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 da Regressão Linear

1.1 Estratégia de Implementação

De início implementei o método **regressaoLinear()** que recebe quantidade **n** de pontos e duas listas **x** e **y** com os pontos a ser calculado.

```
def regressaoLinear(n,x,y):
```

No método de início faço um laço para calcular os somatórios para realizar o cálculo da regressão linear.

```
for i in range(n):
    somatorioXY += x[i]*y[i]
    somatorioX += x[i]
    somatorioY += y[i]
    quadradoX += (x[i]**2)
```

Ao final do laço faço o cálculo da regressão e insiro nas variáveis **a** e **b** e retorno-as ao final do método**.**

```
a=((n*somatorioXY)-(somatorioX*somatorioY))/((n*quadradoX)
- (somatorioX**2))
```

```
b=((somatorioY*quadradoX)-(somatorioX*somatorioXY))/((n*qua
dradoX) - (somatorioX**2))
```

```
return a,b
```

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 dos valores de **x** e **y**. O arquivo texto de saída é dividido em os valores de **x**, **y**, valores calculados em **a** e **b**, respectivamente.

1.3 Problema teste 1, 2, 3…

- Problema 8.1 letra a página 268

Pontos(x): [1980.0, 1985.0, 1990.0, 1993.0, 1994.0, 1996.0, 1998.0]

Pontos(y): [8300.0, 9900.0, 10400.0, 13200.0, 13600.0, 13700.0, 14600.0]

Resultado obtido:

a = 362.48541423570595

b = -709699.5332555426

- Problema 8.1 letra b página 268

Pontos(x): [1980.0, 1985.0, 1990.0, 1993.0, 1994.0, 1996.0, 1998.0]

Pontos(y): [1688.0, 1577.0, 1397.0, 1439.0, 1418.0, 1385.0, 1415.0]

Resultado obtido:

a = -16.298716452742124

b = 33922.55892648775

- Problema 8.5 letra a página 269

Pontos(x): 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Pontos(y): 0.62 0.63 0.64 0.66 0.68 0.71 0.76 0.81 0.89 1

Resultado obtido:

a = 0.3890909090909092

b = 0.5259999999999997

- Problema 8.11 letra a página 273

Pontos(x): [1980.0, 1985.0, 1989.0, 1992.0, 1994.0, 1995.0, 1997.0]

Pontos(y): [248.8, 398.0, 503.7, 684.9, 749.9, 793.5, 865.7]

Resultado obtido:

a = 37.38066406249709

b = -73791.84453124802

- Problema 8.11 letra b página 273

Pontos(x): [1980.0, 1985.0, 1989.0, 1992.0, 1994.0, 1995.0, 1997.0]

Pontos(y): [355.3, 438.0, 487.7, 617.8, 658.1, 674.6, 707.6]

Resultado obtido:

a = 22.048372395836243

b = -43319.83203125

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 da Aproximação Polinomial (Parte 1)

Caso discreto

2.1 Estratégia de Implementação

De início tem a função **calculoMatriz()** que recebe matriz m e os valores de y vindo do arquivo texto de entrada. Dentro da função faço somatório do produto de y pela matriz e insiro em uma lista temporário.

```
def calculoMatriz(m, y):
    # Crio lista temporária.
    temp = []
    for i in range(len(m)):
            temp.append(sum([y[j] * m[i][j] for j in range(len(y))]))

    return temp
```

Na função **resolveMatriz()** que recebe matriz m é utilizada para montar uma nova matriz para determinar a expressão, no qual é utilizada outra função produto() para fazer produto dos valores das colunas e linhas da matriz e assim retornando o somatório.

```
def resolveMatriz(m):
    temp = []
    for i in range(len(m)):
            temp.append([produto(m, j, i) for j in range(len(m))])
    return temp
```

```
def produto(m, x, y):
    # Cria lista auxiliar.
    aux = []
    for i in range(len(m[0])):
        aux.append(m[x][i] * m[y][i])
    return sum(aux)
```

Determinando a matriz e o vetor resultante uso o método de fatoração LU para resolver o sistema e devolver o vetor com valores definidos.

```
a = fatorLu(resolveMatriz(matriz), v)
```

Resolvendo o sistema é a feita a junção do x elevado ao expoente correspondente e após gravando no arquivo de saída.

```
for i in range(grauM + 1):
    s += a[i]*x**i
saida.write(f"P(x) = {s}")
```

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 dos valores de x, y e o grau M. O arquivo texto de saída contém a expressão encontrada.

2.3 Problema teste 1, 2, 3...

- Problema 8.1 letra a página 268

Pontos(x): [1980.0, 1985.0, 1990.0, 1993.0, 1994.0, 1996.0, 1998.0]

Pontos(y): [8300.0, 9900.0, 10400.0, 13200.0, 13600.0, 13700.0, 14600.0]

Aplicando para grau 2:

P(x) = 6.45740201013633*x**2 - 25324.2535484718*x + 24834710.9198017

Substituindo x na função por 2000:

15811.863403420895

- Problema 8.1 letra b página 268

Pontos(x): [1980.0, 1985.0, 1990.0, 1993.0, 1994.0, 1996.0, 1998.0]

Pontos(y): [8300.0, 9900.0, 10400.0, 13200.0, 13600.0, 13700.0, 14600.0]

Aplicando para grau 2:

P(x) = 1.15006132108461*x**2 - 4591.09906176698*x + 4583374.24024525

Substituindo x na função por 2000:

1421.4010497294366

- Problema 8.5 letra a página 269

Pontos(x): 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Pontos(y): 0.62 0.63 0.64 0.66 0.68 0.71 0.76 0.81 0.89 1

Aplicando para grau 2:

P(x) = **0.564393939393961*x**2 - 0.231742424242448*x + 0.650166666666672**

- Problema 8.11 letra a página 273

Pontos(x): [1980.0, 1985.0, 1989.0, 1992.0, 1994.0, 1995.0, 1997.0]

Pontos(y): [248.8, 398.0, 503.7, 684.9, 749.9, 793.5, 865.7]

Aplicando para grau 2:

P(x) = 0.625542790137219*x2 - 2450.42989111147*x + 2399718.96862641**

- Problema 8.11 letra b página 273

Pontos(x): [1980.0, 1985.0, 1989.0, 1992.0, 1994.0, 1995.0, 1997.0]

Pontos(y): [355.3, 438.0, 487.7, 617.8, 658.1, 674.6, 707.6]

Aplicando para grau 2:

P(x) = 0.395179238541631*x2 - 1549.59643964623*x + 1519291.28759129**

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.

2. Método da Aproximação Polinomial (Parte 2)

Caso contínuo

2.1 Estratégia de Implementação

De início tem a função **sistema()** que recebe o grau m, o intervalo [a,b] e a função matemática a ser feita a aproximação. Nessa função forma sistema e fazendo a resolução do mesmo utilizando o produto escalar usual com integração para C[a,b].

```
def sistema(grauM, a, b, funcao):

    x = Symbol('x')

    for coluna in range(grauM + 1):

        linha_L = [0.0] * (grauM + 1)

        for linha in range(grauM + 1):

            linha_L[linha] = integrate(((x ** linha) * (x ** coluna)), (x, a, b))

        matriz2[coluna] = integrate((sympify(funcao) * (x ** coluna)), (x, a, b))

        matriz1.append(linha_L)
```

Reaproveito código da implementação da eliminação de gauss e utilizo a função **eliminaçãoGauss()** para fazer a triangulação da matriz.

```
def eliminacaoGauss(A, b):

    # n é a ordem da matriz A
```

```
    n = len(A)
    # Para cada etapa k
    for k in range(0, n-1):
        # Para cada linha i
        for i in range(k+1, n):
            #Calcula o fator m
            m = -A[i][k]/A[k][k]
                # Atualiza a linha i da matriz, percorrendo todas as colunas j
            for j in range(k+1, n):
                A[i][j] = (m * A[k][j]) + A[i][j]
            # Atualiza o vetor b na linha i
            b[i] = m * b[k] + b[i]
```

E por fim resolvendo o sistema na função **solucao()**

```
def solucao(grauM, matriz1, matriz2, funcao):
    sistema(grauM, a, b, funcao)
    eliminacaoGauss(matriz1, matriz2)
    X = [0.0] * (grauM + 1)
    X[grauM] = matriz2[grauM] / matriz1[grauM][grauM]
    for i in range(grauM, -1, -1):
        s = 0.0
```

```
        for j in range(i + 1, grauM + 1):
            s = s + (matriz1[i][j] * X[j])
            X[i] = (matriz2[i] - s) / matriz1[i][i]

    return X
```

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 da função, dos valores do intervalo e do grau M. O arquivo texto de saída contém a expressão encontrada.

2.3 Problema teste 1, 2, 3...

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 Interpolação Lagrange

3.1 Estratégia de Implementação

O método **polinomioLagrange()** que recebe como parâmetro os pontos x e y. No método fazemos montagem dos polinômios, continuando no método utilizamos a fórmula de lagrange para encontrar o polinômio de interpolação. Ao final agrupamos os termos semelhantes e fazemos a simplificação da expressão encontrada retornando e gravando assim no arquivo de saída.

```
def polinomioLagrange(x,y):

    X = Symbol('x')

    coeficientes = []

    for i in range(len(x)):

        L = 1

        for j in range(len(x)):

            if i ≠ j:

                L *= (X - x[j]) / (x[i] - x[j])

        coeficientes.append(L)


    polinoInterpo = 0

    for i in range(len(coeficientes)):

        polinoInterpo += y[i]*coeficientes[i]


    p = simplify(polinoInterpo)
```

```
    p = expand(p)

    return p
```

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 dos pontos x e dos pontos y. O arquivo texto de saída terá a expressão obtida.

3.3 Problema teste 1, 2, 3...

- Problema 10.2 letra a página 316

Pontos x e y:

x: 0.0 10.0 20.0

y: 0.0 6.0 4.0

Resultado obtido:

P(x): -0.04*x2 + 1.0*x**

- Problema 10.6 letra a página 318

Pontos x e y:

x: 1000 1500 2000

y: 5.48 7.9 11

Resultado obtido:

P(x): 1.35999999999999e-6*x2 + 0.00144000000000001*x + 2.67999999999999**

- Problema 10.6 letra b página 318

Pontos x e y:

x: 1000 1500 2000 2500

y: 5.48 7.9 11 13.93

Resultado obtido:

P(x): -1.13333333333333e-9*x3 + 6.45999999999996e-6*x**2 - 0.00592666666666658*x + 6.08000000000001**

- Problema 10.9 letra a página 318

Pontos x e y:

x: 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5

y: 0.0 0.0125 0.06 0.195 0.5 1.0875 2.1 3.71 6.12 9.5625

Resultado obtido:

P(x): 2.77555756156289e-17*x9 + 5.6843418860808e-14*x**5 + 0.0200000000000955*x**4 + 0.00999999999987722*x**3 + 0.0199999999999818*x**2 + 0.0100000000000122*x**

- Problema 10.9 letra b página 318

Pontos x e y:

x: 0.5 1.0875 2.1

y: 2 2.5 3

Resultado obtido:

P(x): -0.223272918308379*x2 + 1.20550958760179*x + 1.4530634357762**

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 Interpolação Newton

4.1 Estratégia de Implementação

De início temos o método **interpolacaoNewtob()** que recebe os pontos x e pontos y como parâmetro. Dentro do método fazemos o cálculo da diferença dividida para os pontos de x e os pontos de y, seguindo de acordo com a fórmula disponibilizada no slide. Após fazer o cálculo, calculamos o restante da expressão somando o resultado da diferença dividida encontrada anteriormente para fazer a interpolação linear. Em seguida fazemos a simplificação da expressão encontrada e retornando-a do método.

```
def interpolacaoNewton(x, y):

    tabela = []

    tabela.append(y)

    pontos = x

    print(pontos)

    passo = 1

    for i in range(len(x) - 1):

        ordem = []

        for j in range(len(tabela[i]) - 1):

            difDividida = (tabela[i][j + 1] - tabela[i][j])
/ (pontos[j + passo] - pontos[j])

            ordem.append(difDividida)

        tabela.append(ordem)

        passo = passo + 1
```

```
    aprox = 0
    grau = 0
    X = Symbol('x')
    for i in range(len(tabela)):
        fator = tabela[i][0]
        for j in range(grau):
            fator = fator * (X - pontos[j])
        grau= grau +1
        aprox = aprox + fator
    p = simplify(aprox)
    p = expand(p)
    return p
```

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 dos pontos x e dos pontos y. O arquivo texto de saída terá a expressão encontrada.

4.3 Problema teste 1, 2, 3...

- Problema 10.2 letra a página 316

Pontos x e y:

x: 0.0 10.0 20.0

y: 0.0 6.0 4.0

Resultado obtido:

P(x): -0.04*x2 + 1.0*x**

- Problema 10.6 letra a página 318

Pontos x e y:

x: 1000 1500 2000

y: 5.48 7.9 11

Resultado obtido:

P(x) = 1.36e-6*x2 + 0.00144*x + 2.68**

- Problema 10.6 letra b página 318

Pontos x e y:

x: 1000 1500 2000 2500

y: 5.48 7.9 11 13.93

Resultado obtido:

P(x) = -1.13333333333333e-9*x3 + 6.45999999999999e-6*x**2 - 0.00592666666666666*x + 6.08**

- Problema 10.9 letra a página 318

Pontos x e y:

x: 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5

y: 0.0 0.0125 0.06 0.195 0.5 1.0875 2.1 3.71 6.12 9.5625

Resultado obtido:

P(x) = -2.88423018566641e-17*x9 + 5.67072843708837e-16*x**8 - 4.66641716449492e-15*x**7 + 2.08701111947818e-14*x**6 - 5.51049175425946e-14*x**5 + 0.0200000000000872*x**4 + 0.00999999999991993*x**3 + 0.0200000000000384*x**2 + 0.00999999999999275*x**

- **Problema 10.9 letra b página 318**

Pontos x e y:

x: 0.5 1.0875 2.1

y: 2 2.5 3

Resultado obtido:

P(x) = -0.223272918308379*x2 + 1.20550958760179*x + 1.4530634357762**

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 Derivação Numérica de Primeira Ordem

5.1 Estratégia de Implementação

O script está organizado para fazer o cálculo das derivadas de primeira ordem a partir dos dados obtidos do arquivo texto de entrada, sendo respectivamente a função, x e o h. Utilizando função nativa do python **eval()**,função que recebe a expressão a ser calculada e o x a ser substituído na expressão, fazendo assim as derivadas progressiva, retardada e centrada; e ao final gravando os resultados obtidos no arquivo de saída.

```
funcao = entrada.readline()

x = int(entrada.readline())

h = float(entrada.readline())

progressiva = (f(funcao, x + h) - f(funcao, x)) / h

retardada = (f(funcao, x) - f(funcao, x - h)) / h

centrada = (f(funcao, x + h) - f(funcao, x - h)) / (2 * h)

saida.write(f"Progressiva: {progressiva}\n")

saida.write(f"Retardada: {retardada}\n")

saida.write(f"Centrada: {centrada}\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 da função, x e o h. O arquivo texto de saída terá o resultado obtido das derivadas progressiva, retardada e centrada.

5.3 Problema teste 1, 2, 3…

Exemplo 1

Dados de entrada:

Função: ((sin(x+2)-e**-x**2)/(x**2+log(x+2)))+x

x: 2.5

h: 0.01

Resultado obtido:

Progressiva: 1.0594907626876982

Retardada: 1.0587660615604566

Centrada: 1.0591284121240774

Exemplo 2

Dados de entrada:

Função: log(x)

x: 1.8

h: 0.01

Resultado obtido:

Progressiva: 0.5540180375615322

Retardada: 0.5571045049455381

Centrada: 0.5555612712535352

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 Derivação Numérica de Segunda Ordem

6.1 Estratégia de Implementação

O script está implementado para fazer o cálculo das derivadas de segunda ordem a partir dos dados obtidos do arquivo texto de entrada, sendo respectivamente a função, x e o h. Utilizando função nativa do python **eval()**,função que recebe a expressão a ser calculada e o x a ser substituído na expressão, fazendo assim as derivadas regressiva, centrada e avançada; e ao final gravando os resultados obtidos no arquivo de saída.

```
# Faz a leitura da função,x e h do arquivo texto.

funcao = entrada.readline()

x = float(entrada.readline())

h = float(entrada.readline())

regressiva = (f(x, funcao) - 2 * f(x - h, funcao) + f(x - 2 * h, funcao)) / h**2

centrada = (f(x + h, funcao) - 2 * f(x, funcao) + f(x - h, funcao)) / h**2

avancada = (f(x + 2 * h, funcao) - 2 * f(x + h, funcao) + f(x, funcao)) / h**2

saida.write(f"Funcao: {funcao}")

saida.write(f"Regressiva: {regressiva}\n")

saida.write(f"Centrada: {centrada}\n")

saida.write(f"Avançada: {avancada}\n")
```

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, x e o h. O arquivo texto de saída terá o resultado obtido das derivadas regressiva, centrada e avançada.

6.3 Problema teste 1, 2, 3…

Exemplo 1

Dados de entrada:

Função: ((sin(x+2)-e**-x**2)/(x**2+log(x+2)))+x

x: 2.5

h: 0.01

Resultado obtido:

Progressiva: 1.0594907626876982

Retardada: 1.0587660615604566

Centrada: 1.0591284121240774

Exemplo 2

Dados de entrada:

Função: log(x)

x: 1.8

h: 0.01

Resultado obtido:

Regressiva: -0.31210499214506804

Centrada: -0.30864673840058643

Avançada: -0.30524564645695307

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.

7. Integração por método do trapézio

7.1 Estratégia de Implementação

O script está implementado para fazer o cálculo a partir dos dados obtidos do arquivo texto de entrada, recebendo do arquivo a **função**,**a** e **b**. A partir dos pontos a e b conseguimos obter o h para fazer o cálculo utilizando a regra do trapézio,utilizo a função nativa do python **eval()**,função que recebe a expressão a ser calculada e o x a ser substituído na expressão, e gravamos o resultado obtido no arquivo de saída.

```
funcao = entrada.readline()

a = float(entrada.readline())

b = float(entrada.readline())

h = b - a

i = h/2*(f(funcao,a) + f(funcao,b))

saida.write(f"Função: {funcao}")

saida.write(f"I: {i}\n\n")
```

7.2 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, pontos a e b. O arquivo texto de saída terá o resultado obtido juntamente com a função.

7.3 Problema teste 1, 2, 3…

- Problema 11.1 i página 366

Dados obtidos a partir da questão:

Para T: 2000, 2250, 2500, 2750 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))**.**

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 324872701217.3752

Função: 1/(x**5*(e**(1.432/(2250*x))-1))

I: 1022672024025.1923

Função: 1/(x**5*(e**(1.432/(2500*x))-1))

I: 2582772476512.8135

Função: 1/(x**5*(e**(1.432/(2750*x))-1))

I: 5575106820199.431

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10725486189997.248

- Problema 11.1 ii página 366

Dados obtidos a partir da questão:

Para T: 2000, 2200, 2400, 2600, 2800 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 324872701217.3752

Função: 1/(x**5*(e**(1.432/(2200*x))-1))

I: 829632729417.4371

Função: 1/(x**5*(e**(1.432/(2400*x))-1))

I: 1822256033145.394

Função: 1/(x**5*(e**(1.432/(2600*x))-1))

I: 3570707043171.352

Função: 1/(x**5*(e**(1.432/(2800*x))-1))

I: 6406540055757.339

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10725486189997.248

- Problema 11.6 i página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

Intervalo: 3.9933666e-5 e 5.895923e-5

Resultado obtido:

I: 2561718.1296939827

- Problema 11.6 ii página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

Intervalo: 3.9933666e-5 e 5.895923e-5

Resultado obtido:

I: 17370606133.08559

- Problema 11.11 página 371

Dados obtidos a partir do problema:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

Intervalos: 4.991 5

I: 0.4301031948321018

7.4 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.

8. Integração por múltiplos trapézios

8.1 Estratégia de Implementação

O script está implementado para fazer o cálculo a partir dos dados obtidos do arquivo texto de entrada, recebendo do arquivo a **função** e **os pontos** . A partir dos pontos conseguimos obter o h, fazemos o somatório dos intervalos internos multiplicado por 2 e somamos o ponto inicial com final, utilizo a função nativa do python **eval()** para fazer o cálculo, função que recebe a expressão a ser calculada e o x a ser substituído na expressão, e retorno o resultado obtido para a gravação no arquivo de saída.

```
def trapezioMultiplo(x, func):

    n = len(x)

    somatorio = sum(f(func, x[i]) for i in range(1, n - 1))

    h = x[-1] - x[0]

     I = h * ((f(func, x[0]) + 2 * somatorio + f(func,
x[-1]))/(2*(n-1)))

    return I
```

8.2 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**, os **pontos** e o **n**. O arquivo texto de saída terá o resultado obtido juntamente com a função.

8.3 Problema teste 1, 2, 3…

- Problema 11.1 i página 366

Dados obtidos a partir da questão:

Para T: 2000, 2250, 2500, 2750 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))**.**

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199347969205.3777

Função: 1/(x**5*(e**(1.432/(2250*x))-1))

I: 724306926301.1903

Função: 1/(x**5*(e**(1.432/(2500*x))-1))

I: 2067451748677.5588

Função: 1/(x**5*(e**(1.432/(2750*x))-1))

I: 4941957059403.032

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10325036713963.906

- Problema 11.1 ii página 366

Dados obtidos a partir da questão:

Para T: 2000, 2200, 2400, 2600, 2800 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199347969205.3777

Função: 1/(x**5*(e**(1.432/(2200*x))-1))

I: 571942603229.4377

Função: 1/(x**5*(e**(1.432/(2400*x))-1))

I: 1392429725002.087

Função: 1/(x**5*(e**(1.432/(2600*x))-1))

I: 2984477907021.784

Função: 1/(x**5*(e**(1.432/(2800*x))-1))

I: 5782237309900.986

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10325036713963.906

- Problema 11.6 i página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

I: 1631227.141016913

- Problema 11.6 ii página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

I: 18462931204.992973

- Problema 11.11 página 371

Dados obtidos a partir do problema:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**

Intervalos: 4.991 5

Resultado obtido:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

I: 0.46397375956990305

8.4 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.

9. Método Simpson de 1/3

9.1 Estratégia de Implementação

O script está implementado para fazer o cálculo a partir dos dados obtidos do arquivo texto de entrada, recebendo do arquivo a **função** e **os pontos** . A partir dos pontos conseguimos obter o h, fazemos o somatório dos intervalos internos , em seguida fazemos o somatório do intervalos com algumas condições, caso o i for ímpar multiplica por 4 o resultado do somatório, se for i for par multiplica por 2 o resultado do somatório, utilizo a função nativa do python **eval()** para fazer o cálculo, função que recebe a expressão a ser calculada e o x a ser substituído na expressão, em seguida obtenho o I multiplicando o h pelo cálculo função para o primeiro ponto e para o último somando com o somatório obtido e dividindo por 3 vezes número de intervalos. Ao final, gravo o resultado obtido no arquivo de saída,juntamente com a função.

```
funcao = entrada.readline()

x  =  [float(i)  for  i  in  entrada.readline().replace('\n',
'').split(' ')]

n = int(entrada.readline().replace('\n', ''))

h = (x[-1] - x[0])

p = [x[0]]

for i in range(1, n): p.append(p[i - 1] + (h / n))

p.append(x[-1])

somatorio = 0
```

```
for i in range(1, n):
    if i % 2 =/= 0:
        somatorio += 4 * f(funcao, p[i])
    else:
        somatorio += 2 * f(funcao, p[i])
I = h * ((f(funcao, x[0]) + somatorio + f(funcao, x[-1])) / (3 * n))
saida.write(f"Função: {funcao}")
saida.write(f"I: {I}\n\n")
```

9.2 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**, os **pontos** e o **n**(intervalo). O arquivo texto de saída terá o resultado obtido juntamente com a função.

9.3 Problema teste 1, 2, 3…

- Problema 11.1 i página 366

Dados obtidos a partir da questão:

Para T: 2000, 2250, 2500, 2750 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))**.**

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199054229843.0942

Função: 1/(x**5*(e**(1.432/(2250*x))-1))

I: 723625248828.5453

Função: 1/(x**5*(e**(1.432/(2500*x))-1))

I: 2066272741023.7283

Função: 1/(x**5*(e**(1.432/(2750*x))-1))

I: 4940458197274.489

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10323927279020.596

- Problema 11.1 ii página 366

Dados obtidos a partir da questão:

Para T: 2000, 2200, 2400, 2600, 2800 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199054229843.0942

Função: 1/(x**5*(e**(1.432/(2200*x))-1))

I: 571352239387.6508

Função: 1/(x**5*(e**(1.432/(2400*x))-1))

I: 1391449920942.723

Função: 1/(x**5*(e**(1.432/(2600*x))-1))

I: 2983125228097.7017

Função: 1/(x**5*(e**(1.432/(2800*x))-1))

I: 5780738530862.437

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10323927279020.596

- Problema 11.6 i página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

I: 1628928.111721991

- Problema 11.6 ii página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

I: 18465694870.43107

- Problema 11.11 página 371

Dados obtidos a partir do problema:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**

Intervalos: 4.991 5

Resultado obtido:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

I: 0.46406945937239535

9.4 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.

10. Método Simpson de 3/8

10.1 Estratégia de Implementação

O script está implementado para fazer o cálculo a partir dos dados obtidos do arquivo texto de entrada, recebendo do arquivo a **função** e **os pontos** . A partir dos pontos conseguimos obter o h, fazemos o somatório dos intervalos internos , em seguida calculamos os intervalos e a multiplicação dos intervalos, multiplicando por 3, utilizo a função nativa do python **eval()** para fazer o cálculo, função que recebe a expressão a ser calculada e o x a ser substituído na expressão, o cálculo final é o produto de h pela soma do f(x0),f(xn) e o resultado obtido do somatório anterior, dividindo por 8 . Ao final, gravo o resultado obtido no arquivo de saída,juntamente com a função.

```
h = (x[-1] - x[0])

somatorio = 0

p = [x[0] + (h / n)]

for i in range(1, n - 1): p.append(p[i - 1] + (h / n))

for i in p: somatorio += 3 * f(funcao, i)

I = h * ((f(funcao, x[0]) + somatorio + f(funcao, x[-1])) /
8)

saida.write(f"Função: {funcao}")

saida.write(f"I: {I}\n\n")
```

10.2 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**, os **pontos** e o **n**(intervalo). O arquivo texto de saída terá o resultado obtido juntamente com a função.

10.3 Problema teste 1, 2, 3…

- Problema 11.1 i página 366

Dados obtidos a partir da questão:

Para T: 2000, 2250, 2500, 2750 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))**.**

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 197976201976.75372

Função: 1/(x**5*(e**(1.432/(2250*x))-1))

I: 720204278274.9144

Função: 1/(x**5*(e**(1.432/(2500*x))-1))

I: 2060408978869.4988

Função: 1/(x**5*(e**(1.432/(2750*x))-1))

I: 4935742174401.562

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10329044960626.96

- Problema 11.1 ii página 366

Dados obtidos a partir da questão:

Para T: 2000, 2200, 2400, 2600, 2800 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 197976201976.75372

Função: 1/(x**5*(e**(1.432/(2200*x))-1))

I: 568483207347.3004

Função: 1/(x**5*(e**(1.432/(2400*x))-1))

I: 1386390058054.5056

Função: 1/(x**5*(e**(1.432/(2600*x))-1))

I: 2977014383682.415

Função: 1/(x**5*(e**(1.432/(2800*x))-1))

I: 5777130589825.424

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10329044960626.96

- Problema 11.6 i página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

I: 1627202.9584554269

- Problema 11.6 ii página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

I: 18464416631.26609

- Problema 11.11 página 371

Dados obtidos a partir do problema:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**

Intervalos: 4.991 5

Resultado obtido:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

I: 0.4635279927736623

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

I: 0.4641479868683157

10.4 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.

11. Método Extrapolação de Richardson

11.1 Estratégia de Implementação

O script está implementado para fazer o cálculo a partir dos dados obtidos do arquivo texto de entrada, recebendo do arquivo a **função**, **os pontos** e **intervalo** . A partir dos pontos conseguimos obter o **h** com subtração entre último e primeiro ponto, em seguida gerando outros pontos com o intervalo determinado. Continuando chamo o método **richards()** passando os pontos gerados juntamente com a função a ser calculada, no método uso como base a integrações por trapézio para obter a integral, a partir dos valores retornado utilizo a extrapolação de Richardson e retorno o valor obtido para a gravação no arquivo de saída, juntamente com a função utilizada.

```
def richards(x0, func):
    # Integracao entre dois pontos consecutivos utilizando regra do trapezio.
    I1 = trapezioSimples(x0, func)
    # Integracao com intervalo 2 dos pontos.
    I2 = trapezioMultiplo(x0[::2], func)
    # Integracao com n sub-intervalos.
    I3 = trapezioMultiplo(x0, func)
    I4 = 4 / 3 * I2 - 1 / 3 * I1
    I5 = 4 / 3 * I3 - 1 / 3 * I2
    I = 16 / 15 * I5 - 1 / 15 * I4
```

```
    return I

def trapezioSimples(x0, fun):

    h = x0[-1] - x0[0]

    I = h * ((f(fun, x0[0]) + f(fun, x0[-1])) / 2)

    return I

def trapezioMultiplo(x0, fun):

    n = len(x0)

    somatorio = sum(f(fun, x0[i]) for i in range(1, n - 1))

    h = x0[-1] - x0[0]

    I = h * ((f(fun, x0[0]) + 2 * somatorio + f(fun,
x0[-1])) / (2 * (n - 1)))

    return I
```

11.2 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**, os **pontos** e o **n**(intervalo). O arquivo texto de saída terá o resultado obtido juntamente com a função.

11.3 Problema teste 1, 2, 3...

- Problema 11.1 i página 366

Dados obtidos a partir da questão:

Para T: 2000, 2250, 2500, 2750 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))**.**

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199113357143.76105

Função: 1/(x**5*(e**(1.432/(2250*x))-1))

I: 723777997445.4268

Função: 1/(x**5*(e**(1.432/(2500*x))-1))

I: 2066389786703.6804

Função: 1/(x**5*(e**(1.432/(2750*x))-1))

I: 4940162930097.32

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10322731046356.354

- Problema 11.1 ii página 366

Dados obtidos a partir da questão:

Para T: 2000, 2200, 2400, 2600, 2800 e 3000.

Função: 1/(x**5*(e**(1.432/(T*x))-1))

a: 0.00004

b: 0.00007

Resultado obtido:

Função: 1/(x**5*(e**(1.432/(2000*x))-1))

I: 199113357143.76105

Função: 1/(x**5*(e**(1.432/(2200*x))-1))

I: 571489028045.5298

Função: 1/(x**5*(e**(1.432/(2400*x))-1))

I: 1391613038361.8315

Função: 1/(x**5*(e**(1.432/(2600*x))-1))

I: 2983134350085.1113

Função: 1/(x**5*(e**(1.432/(2800*x))-1))

I: 5780300893762.988

Função: 1/(x**5*(e**(1.432/(3000*x))-1))

I: 10322731046356.354

- Problema 11.6 i página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 2000)) - 1))

I: 1628974.7951055057

- Problema 11.6 ii página 368

Dados obtidos a partir do problema:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

Intervalos: 3.9933666e-5 5.895923e-5

Resultado obtido:

Função: (2 * pi * 6.6256 * 10 ** -27 * (2.99793 * 10 ** 10) ** 2) / (x ** 5 * (e ** (6.6256 * 10 ** -27 * 2.99793 * 10 ** 10 / (1.38054 * 10 ** -16 * x * 6000)) - 1))

I: 18465879222.61467

- Problema 11.11 página 371

Dados obtidos a partir do problema:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**

Intervalos: 4.991 5

Resultado obtido:

Função: 1/(0.005*sqrt(2*pi))*e**(-1/2*((x-4.991)/0.005)**2)

I: 0.4641479868683157

11.4 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

Execute também o comando abaixo para verificar se o pip foi instalado, sendo ele o gerenciador de pacotes do python:

pip --version

Instalar as bibliotecas numpy e sympy.

Execute no seu terminal o seguinte comando:

pip install numpy

pip install sympy

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

Execução dos códigos com os testes

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