UNIVERSIDADE ESTADUAL DE SANTA CRUZ – UESC

ISRAEL SANTANA DOS ANJOS

RELATÓRIO DE IMPLEMENTAÇÃO DE MÉTODOS PARA A DISCIPLINA DE ANÁLISE NÚMERICA

ILHÉUS – BAHIA

2024

ISRAEL SANTANA DOS ANJOS

RELATÓRIO DE IMPLEMENTAÇÃO DE MÉTODOS PARA A DISCIPLINA DE ANÁLISE NÚMERICA

Relatório da análise de implementação dos métodos apresentado ao Curso de graduação em Ciência da computação da Universidade Estadual de Santa Cruz, como parte de um dos créditos da disciplina de análise numérica.

Docente: Gesil Sampaio Amarantes

ILHÉUS – BAHIA

2024

Sumário

Linguagem escolhida e justificativa

Para um melhor desenvolvimento foi escolhido o python como linguagem de programação, pois, essa linguagem é bastante simples implementar métodos numéricos, como também, possui uma vasta biblioteca e comunidade grande. A versão foi a 3.11.7, o interpretador pode ser baixado através do link: www.python.org.

A instalação das bibliotecas utilizadas pode ser feita pelo prompt de comando, ou terminal da IDE, pelo seguinte comando:

1. pip install sympy
2. pip install numpy

Uma vez instalado não se faz necessário chamar os comandos novamente.

Como ambiente de desenvolvimento foi escolhido o visual studio code, por facilitar a escrita e depuração de códigos, sendo também, uma opção de fácil identificação de erros de sintaxe e formatação concisa. A versão utilizada foi a 1.87.2 e o download poder ser feito pelo link: code.visualstudio.com.

Outra ferramenta utilizada na realização dos cálculos e plotar os gráficos das funções foi o geoGebra. Essa ferramenta, foi de grande auxílio na escolha dos intervalos para a resolução dos problemas. O geoGebra pode ser acessado pelo link: www.geogebra.org.

Teste e funcionamento do código

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

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

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

Regressão Linear

Estratégias de implementação

A entrada dos dados foi feita através de arquivos externos (.txt). Foi necessário a utilização da biblioteca math apenas para facilitar alguns cálculos e tornar o código mais limpo e legível. Um exemplo da utilização da biblioteca pode ser visto na linha 31, onde é feito o cálculo da raiz quadrada.

Esse código implementa funções para realizar regressao linear e calcular o coeficiente de correlação de Pearson.

A função, *regressao_linear(x, y)*, recebe duas listas *x* e *y* como entrada. Ela calcula os parâmetros da reta de regressão linear que melhor se ajusta aos pontos *(x, y)*. Para isso, acumula somas e médias dos valores de *x* e *y*, além de somas de produtos e quadrados de *x*.

Com esses valores, a função calcula o coeficiente angular *(a1)* e o coeficiente linear *(a0)* da reta de regressão. Por fim, retorna esses coeficientes como uma tupla.

A segunda função, *coeficiente_correlacao(x, y),* também recebe duas listas *x* e *y* como entrada. Ela calcula o coeficiente de correlação de Pearson entre *x* e *y*, esse coeficiente mede a força e o sentido da relação linear entre duas variáveis. Similar a função anterior, ela acumula somas e médias dos valores de *x* e *y*, além de somas de produtos e quadrados de *x* e *y*. Usando essas somas e a biblioteca *math*, ela calcula o coeficiente de correlação e o retorna como saída.

Estrutura dos arquivos de entrada\saída

O arquivo de entrada é organizado de maneira que em cada linha esteja armazenado uma das sequências dos dados necessários para o cálculo do método. Dessa forma, na primeira linha se encontram os valores de x, na segunda, os valores de y. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização. Essas entradas podem ser alteradas no arquivo *entrada_Regressão* que se encontra na pasta entradas teste.

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

```
Nome do método
Resultados
```

Problema

Exemplo 1

Como dados de entrada temos os seguintes:

1980 1985 1990 1993 1994 1996 1998

1688 1577 1397 1439 1418 1385 1415

Como já supracitado, a primeira linha corresponde a lista dos valores de x e a segunda a de y, respectivamente.

Após a rodar o algoritmo foi obtida a seguinte saída:

```
Resultado Regressão
y =~ 33922.55892648775 + -16.298716452742124x
r: -0.9126263729121098
```

Exemplo 2

Foi executado o código novamente com as entradas seguintes:

1980 1985 1990 1993 1994 1996 1998

8300 9900 10400 13200 13600 13700 14600

E foi observado o seguinte resultado como saída:

```
Resultado Regressão
y =~ -709699.5332555426 + 362.48541423570595x
r: 0.9685408745259411
```

Análise das saídas

Para o primeiro exemplo a equação da linha de melhor ajuste é $y = 33922.56 - 16.30x$ e o coeficiente de correlação é: -0.91. Para o segundo exemplo, a equação de melhor ajuste é $y = -709699.53 + 362.49x$ e o coeficiente de correlação é 0.97.

O primeiro exemplo indica uma forte correlação negativa. Isso significa que à medida que aumentamos os valores de x, os valores de y tendem a diminuir. Porém, no segundo exemplo,

percebe-se uma forte relação positiva, indicando que à medida que aumentamos os valores de x, os valores de y também aumentam.

Dificuldades enfrentadas

Não houve dificuldades

Polinômio de Lagrange

Estratégia de implementação

A iteração entrada dos dados foi feita através de arquivos externo (.txt). Foi preciso o uso da biblioteca *sympy sp* para a conversão das strings *(x)* em variáveis simbólicas.

A função *polinômio(k, pontos, n)* recebe três argumentos:

- k: índice do ponto em relação ao qual o polinômio de Lagrange será construído.
- pontos: lista de pontos de interpolação (coordenadas x).
- n: número total de pontos de interpolação.

A função retorna um polinômio de Lagrange que passa pelo ponto pontos[k].

Já a função *lagrange(fx, pontos)*, recebe dois argumento, onde:

- fx: lista de valores de função (coordenadas y) correspondentes aos pontos de interpolação.
- pontos: lista de pontos de interpolação (coordenadas x).

A função retorna a string que representa o polinômio interpolador de Lagrange completo. Para isso, calcula o polinômio de Lagrange Poli usando a função Polinomios, que já foi citada anteriormente e então:

    - Multiplica Poli pelo valor de função *fx[k]* correspondente ao ponto pontos[k].
    - Soma o produto *fx[k] * Poli* a *P*.

E então, retorna a string que representa o polinômio final.

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 maneira, nas primeiras linhas até o algoritmo de leitura encontrar o “\n” estarão todos os valores para as coordenadas de y

(valores de função) e na subsequente ao “\n”, os valores de entrada para as coordenadas x (pontos de interpolação).

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

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para este método foram utilizados os seguintes pontos:

0.1	0.62
0.2	0.63
0.3	0.64
0.4	0.66
0.5	0.68
0.6	0.71
0.7	0.76
0.8	0.81
0.9	0.89
1	1

Onde, a primeira coluna se encontra os valores relacionados a y e na segunda coluna os valores referentes a x.

Após executar o código obtivemos a seguinte saída, que se encontra no arquivo “resultado.txt”, como já mencionado anteriormente.

> P(x) = -1146464516.22677*x**9 + 7745494791.27448*x**8 - 23181005492.5823*x**7 + 40341226885.1545*x**6 - 44992037364.3078*x**5 + 33352089178.8184*x**4 - 16434127889.0092*x**3 + 5190943324.09316*x**2 - 953799193.625119*x + 77680277.410669.

Podemos então inferir que o resultado da interpolação de Lagrange mostrou um bom resultado, encontrando um polinômio que passa pelos pontos que foram disponibilizados para o método.

Porém, a interpolação de Lagrange pode sofrer de um fenômeno chamado Runge, onde o erro da interpolação pode aumentar de uma forma significativa ao passo que aumentamos o número de pontos.

Exemplo 2

Para o segundo exemplo foi utilizada a seguinte estrada:

1980	
1985	248.8
1989	398
1992	503.7
1994	684.9
1995	749.9
1997	793.5
	865.7

Com a execução do código com as novas entradas de teste, podemos observar o seguinte resultado:

P(x) = 2.71622370816285e-14*x**6 - 9.77355788072824e-11*x**5 + 1.42622253631237e-7*x**4 - 0.0001075448945572 7*x**3 + 0.0439410386947188*x**2 - 9.13131319084323*x + 2728.39962182633

Dificuldades enfrentadas

Não houve dificuldades

Interpolação por diferenças divididas de Newton

Estratégias de implementação

A iteração entrada dos dados foi feita através de arquivos externo (.txt). Foi preciso o uso da biblioteca *sympy sp* para a conversão das strings *(x)* em variáveis simbólicas.

A função *calculaFdosPontos* recebe dois parâmetros:

- fx: Uma expressão simbólica que representa a função a ser interpolada.
- pontos: Uma lista contendo os pontos de interpolação no formato [x0, x1, ..., xn].

Então a função devolve, derivada enésima da função fx no ponto x0, calculada utilizando diferenças divididas de Newton.

Já a função *newton* recebe, também, dois parâmetros:

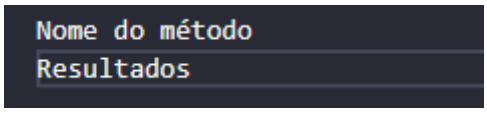
- fx: Uma expressão simbólica que representa a função a ser interpolada.
- pontos: Uma lista contendo os pontos de interpolação no formato [x0, x1, ..., xn].

E retorna uma *string* representando o polinômio de interpolação de Newton. A função utiliza da calculaFdosPontos para calcula o coeficiente do termo atual.

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 maneira, nas primeiras linhas até o algoritmo de leitura encontrar o “\n” estarão todos os valores para os dados de x (valores da função) e na subsequente ao “\n”, os valores de entrada para os dados de y (pontos de interpolação). As informações sobre as entradas podem ser verificadas no arquivo “*entrada_Newton.txt*”

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



Problema

Exemplo 1

Para demonstrar o código como também fazer uma comparação com a interpolação de Lagrange, foram utilizadas as seguintes entradas testes.

0.1	0.62
0.2	0.63
0.3	0.64
0.4	0.66
0.5	0.68
0.6	0.71
0.7	0.76
0.8	0.81
0.9	0.89
1	1

E então, conseguimos observar o seguinte resultado com a execução do código:

```
P(x) = -1267.64*x**9 + 5877.99*x**8 - 11554.254*x**7 + 12565.9906*x**6 - 8280.454332*x**5 + 3405.204419*x**4 - 867.4441686*x**3 + 131.18585124*x**2 - 10.5359163504*x + 0.9599933152
```

Exemplo 2

Para o exemplo dois, temos as seguintes entradas de teste:

1980	248.8
1985	398
1989	503.7
1992	684.9
1994	749.9
1995	793.5
1997	865.7

Após a troca da entrada com as supracitadas, obtivemos a seguinte saída:

```
P(x) = 0.03*x**5 - 298.31*x**4 + 1186515.81*x**3 - 2359654503.03*x**2 + 2346348293844.54*x - 933245055565048.0
```

Análise das saídas – Lagrange e Newton

O polinômio de Lagrange obtido é de nono grau, isso mostra que ele passa pelos 10 pontos de interpolação fornecido pelo exemplo um. Já o segundo exemplo, podemos observar que obtemos um polinômio de sexto grau, isso significa que ele passa pelos 7 pontos de interpolação fornecido.

Ao executar o método de diferenças divididas de Newton, com esses mesmos dados de entradas, foi observado que os polinômio resultantes foram de nono e oitavo graus assim como os de Lagrange, isso mostra que o método de Newton passou pelos mesmos 10 pontos e 7 pontos respectivamente.

Derivada de primeira ordem

Estratégia de Implementação

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

A função *setFuncao* recebe apenas um *f* como parâmetro. E "*f*" uma *string* representando a expressão matemática da função a ser utilizada, ela também, define uma variável global com o nome *funcao* e atribui a ela a string *f*.

Na função denominada *func* ela recebe como parâmetro, também, apenas um argumento. Sendo ele um *x,* que é um valor numérico que representa o ponto de entrada para a função. Então, retorna o valor da função f no ponto x.

A lógica usada para o cálculo da derivada de primeira ordem da função foi:

$$\frac{f(x+1) - f(x-1)}{(x+1) - (x-1)}$$

Neste sentido, foi implementada a seguinte função *derivadaPrimeira* que recebe como parâmetro um valor numérico *x*. Esse valor numérico representa o ponto no qual se deseja calcular a derivada primeira. E por fim, retorna uma *string* representando a derivada primeira da função *f* no ponto *x*.

Antes de executar o código vale destacar que, é preciso passar a função para a *setFuncao()*, como no exemplo abaixo.

```
fun, x = leitura.lerArquivoDeri()
derivadas.setFuncao(fun)
```

É de fundamental importância verificar a correspondência adequada entre símbolos e operações matemáticas em relação às funcionalidades de reconhecimento de funções do Python. Por exemplo, uma potenciação x^2, no arquivo de entrada deve ser escrita como: x**2.

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *x*. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Derivada.txt*". O mesmo arquivo também é utilizado para o cálculo da derivada segunda ordem, com uma pequena diferença.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está na vigésima sexta linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problema

Exemplo 1

Para o primeiro problema temos a seguinte equação com o respectivo x, para ser analisada:

$$\frac{2 * \pi * 0.00000595481384}{x^5 * \left(e^{\frac{0.000239798497}{x}-1}\right)}$$

$$X = 0.00003933666$$

Após a execução do código obtivemos a seguinte saída:

$$I(f(x)) \sim = -4.325802857101102e-05$$

Para $x = 0.00005895923$ temos a seguinte saída:

$$I\big(f(x)\big) \cong -5.550468452396318e-05$$

Exemplo 2

O segundo problema foi utilizado a equação descrita a seguir:

$$\frac{1}{0.005 * 2 * \pi^{\frac{1}{2}}} * e^{-(\frac{1}{2})*(\frac{x-4.991}{0.005})^2}$$

$$X = 4.991$$

Após a execução do código obtivemos a seguinte saída:

$$I(f(x)) \sim = 0.0$$

Para $x = 5$ temos o seguinte resultado:

$$I(f(x)) \sim = 0.0$$

Dificuldades enfrentadas

Não houve grandes dificuldades, o método é bastante simples para se fazer a implementação.

Derivada de segunda ordem

Estratégia de Implementação

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

Foi utilizado o mesmo arquivo da derivada de primeira ordem para fazer a implementação da derivada de segunda ordem. Sendo assim, as funções *setFuncao* e func são as mesmas, cujas explicação está descrita a seguir, como também na derivada de primeira ordem.

A função *setFuncao* recebe apenas um *f* como parâmetro. E "*f*" uma *string* representando a expressão matemática da função a ser utilizada, ela também, define uma variável global com o nome *funcao* e atribui a ela a string *f*.

Na função denominada *func* ela recebe como parâmetro, também, apenas um argumento. Sendo ele um *x,* que é um valor numérico que representa o ponto de entrada para a função. Então, retorna o valor da função f no ponto x.

A lógica utilizada para implementar a derivada de segunda ordem foi:

$$(f(x+1) - f(x)) - (f(x) - f(x-1))$$

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *x*. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Derivada.txt*". O mesmo arquivo também é utilizado para o cálculo da derivada de primeira ordem, com uma pequena diferença.

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está na vigésima nona linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para o primeiro problema temos a seguinte equação com o respectivo x, para ser analisada:

$$\frac{2 * \pi * 0.00000595481384}{x^5 * \left(e^{\frac{0.000239798497}{x}} - 1\right)}$$

$$X = 0.00003933666$$

Após a execução do código obtivemos a seguinte saída:

$$I(f(x)) \sim = -24656657330.96032$$

Para $x = 0.00005895923$ com a mesma equação, temos a seguinte saída:

$$I(f(x)) \sim = -1433958741163.9412$$

Exemplo 2

O segundo problema foi utilizado a equação descrita a seguir:

$$\frac{1}{0.005 * 2 * \pi^{\frac{1}{2}}} * e^{-(\frac{1}{2}) * (\frac{x - 4.991}{0.005})^2}$$

$$X = 4.991$$

Após a execução do código obtivemos a seguinte saída:

$$I(f(x)) \sim = -159.5769121605731$$

Para $x = 5$ temos a respectiva saída:

$$I(f(x)) \sim = -31.580063320353798$$

Dificuldades enfrentadas

Não houve grandes dificuldades, o método é bastante simples para se fazer a implementação.

Trapézio simples

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy sp para a conversão das *str(x)* em variável simbólica.

Para esse método foi implementado uma função que recebe três parâmetros. A e B que corresponde aos pontos extremos do intervalo e a função. Foi implementado a fórmula para o trapézio simples:

$$T(f) = I_1(f) = \frac{f(a) + f(b)}{2} \times (b - a)$$

Essa fórmula corresponde exatamente ao valor da área do trapézio definido pela reta interpoladora.

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *A e B* que correspondem aos intervalos de integração. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Trapezio.txt*".

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está na décima quarta linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para este método foi utilizada a seguinte função:

$$\frac{1}{0.005 * (2 * \pi)^{\frac{1}{2}} * e^{\left(-\left(\frac{1}{2}\right)*\left(\frac{x-4.99}{0.005}\right)^2\right)}}$$

Com um $A = 4.991$ e o $B = 5$.

Após adicionar os dados no arquivo de entrada obtemos a saída a seguir:

$$I\big(f(x)\big) \cong \frac{0.762338063966931}{\pi^{0.5}}$$

Dificuldades enfrentadas

Não houve dificuldades

Trapézio Múltiplo

Estratégia de Implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy sp para a conversão das *str(x)* em variável simbólica.

Para este método foi implementado uma função *intervalos()* que recebe como parâmetro três valore, sendo eles:

- Início: Representa o limite inferior de integração.
- Fim: Representa o limite superior de integração.
- H: Valor que representa o tamanho do subintervalo.

Com esses parâmetros passados a função retorna uma lista com os pontos de corte dos subintervalos, incluindo os limites de integração.

Por fim foi implementada a função *trapézio_Multiplo()* que recebe como argumento três parâmetros:

- Função: Uma expressão simbólica que representa a função.
- A: Valor que representa o limite inferior.
- B: Valor que representa o limite superior.
- N: Representa o número de subintervalos.

A função chama a função intervalos para obter a lista de subintervalos, passando como parâmetro os valores de A, B e H, que é obtido através do cálculo entre a diferença entre b e a dividido pelo número de subintervalos.

A função retorna o valor numérico que representa a integral definida da função no intervalo especificado.

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *A e B* que correspondem aos intervalos de integração e na última linha o valor de n que corresponde ao número de subintervalos. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_TrapezioM.txt*".

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está na decima sétima linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para este método foi utilizada a seguinte função:

$$0.2 + 25 * x - 200(x^2) + 675(x^3) - 900 * (x^4) + 400 * (x^5)$$

Com $A = 0$, $B = 0.8$ e $N = 3$. Ao executar o código obtemos a seguinte saída:

$$I\big(f(x)\big) \cong 1.379$$

Dificuldades enfrentadas

Não houve dificuldade.

Simpson 1/3

Estratégia de implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy sp para a conversão das *str(x)* em variável simbólica.

Assim como no método do trapézio, neste, foi implementado uma função intervalos () que recebe três parâmetros; início, fim e h. Abaixo uma breve descrição do que cada parâmetro representa:

- Início: Valor numérico que representa o limite inferior de integração.
- Fim: Valor numérico que representa o limite superior de integração.
- H: O tamanho do subintervalo.

Com esses parâmetros passados a função retorna uma lista com os pontos de corte dos subintervalos, incluindo os limites de integração.

Foi então implementada a função "*Simpson1_3*". Na chamada da função se faz necessário passar como parâmetro 4 argumentos, sendo eles:

- Função: Que é uma expressão simbólica que representa a função que vai ser integrada.
- A: Representa o limite inferior de integração.
- B: Representa o limite superior de integração.
- N: Representa o número de subintervalos.

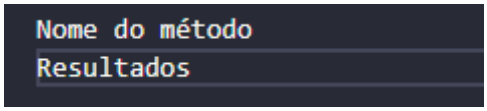
Para obter os pontos de corte a função chama a função "*intervalos ()*" passando como parâmetro; o limite de integração inferior e superior, como também, o tamanho de subintervalos que é calculado pela diferença entre o limite superior e inferior (b e a) dividido pelo dobro do número de subintervalos (n).

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *A e B* que correspondem aos intervalos de integração e na última linha o valor de n que corresponde ao número de subintervalos. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Simpson.txt*".

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está

na vigésima primeira linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:



Problemas

Exemplo 1

A função a seguir é a mesma utilizada no método do trapézio:

$$0.2 + 25 * x - 200(x^2) + 675(x^3) - 900 * (x^4) + 400 * (x^5)$$

Com os mesmos valores numéricos para $A = 0$, $B = 0.8$ e $N = 3$. Ao executar o código obtemos a seguinte saída:

$$I\big(f(x)\big) \cong 1.7106$$

Dificuldades enfrentadas

Não houve dificuldade.

Simpson 3/8

Estratégia de implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy sp para a conversão das *str(x)* em variável simbólica.

Assim como no método anterior, neste, foi implementado uma função intervalos () que recebe três parâmetros; início, fim e h.

- Início: Valor numérico que representa o limite inferior de integração.
- Fim: Valor numérico que representa o limite superior de integração.
- H: O tamanho do subintervalo.

Com esses parâmetros passados a função retorna uma lista com os pontos de corte dos subintervalos, incluindo os limites de integração.

Foi então implementada a função "*Simpson3_8*". Na chamada da função se faz necessário passar como parâmetro 4 argumentos, sendo eles:

- Função: Que é uma expressão simbólica que representa a função que vai ser integrada.
- A: Representa o limite inferior de integração.
- B: Representa o limite superior de integração.
- N: Representa o número de subintervalos.

Para obter os pontos de corte a função chama a função "*intervalos ()*" passando como parâmetro; o limite de integração inferior e superior, como também, o tamanho de subintervalos que é calculado pela diferença entre o limite superior e inferior (b e a) dividido pelo dobro do número de subintervalos (n).

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *A e B* que correspondem aos intervalos de integração e na última linha o valor de n que corresponde ao número de subintervalos. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Simpson.txt*".

Todas as saídas para os problemas que vão ser apresentados nesse relatório estão no mesmo arquivo de saída denominado *resultado*. Este método para ser mais específico, está na vigésima terceira linha, contando a partir dos títulos. As saídas de cada método estão organizadas da seguinte maneira:

```
Nome do método
Resultados
```

Problemas

Exemplo 1

A função a seguir é a mesma utilizada no método do trapézio:

$$0.2 + 25 * x - 200(x^2) + 675(x^3) - 900 * (x^4) + 400 * (x^5)$$

Com os mesmos valores numéricos para $A = 0$, $B = 0.8$ e $N = 3$. Ao executar o código obtemos a seguinte saída:

$$I\big(f(x)\big) \cong 1.7112$$

Dificuldades enfrentadas

Não houve dificuldade.

Quadratura de Gauss

Estratégia de implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca sympy sp para a conversão das *str(x)* em variável simbólica.

O método foi implementado usando a fórmula da integral:

$$I = \frac{(b-a)f(a) + (b-a)f(b)}{2}$$

A função *quadratura_gauss()* recebe três parâmetros:

- Função: Uma expressão simbólica que representa a função a ser integrada.
- A: Representa o limite inferior da integração.
- B: Representa o limite superior da integração.

A função assume implicitamente que a quadratura de Gauss usa apenas um ponto de integração localizado no centro do intervalo.

Após os cálculos usando a fórmula da integral a função retorna o valor de I que representa a aproximação da integral definida da função no intervalo especificado.

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 maneira, a primeira linha contém a função a ser analisada e a segunda o valor a ser atribuído à *A e B* que correspondem aos intervalos de integração inferior e superior respectivamente. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Gauss.txt*".

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

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Para este método foi utilizada a seguinte função:

$$\frac{1}{x^5 * e^{\frac{1.432}{2600}-1}}$$

Com $A = 1$ e $B = 2$.

Ao executar o código obtemos a seguinte saída:

$$I(f(x)) \cong 1.40084231444575$$

Dificuldades enfrentadas

Não houve dificuldade.

Método de Extrapolação de Richards

Estratégia de implementação

A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *numpy* para realizar operações matemáticas.

A função *extrapolação_richards()* recebe três parâmetros:

- X: Vetor contendo os pontos no eixo x.
- y: Vetor contendo os pontos no eixo y.
- x_extrapolar: Valor no eixo x que deseja extrapolar o valor de y.

Para que seja feito os cálculos é preciso que os vetores tenhas o mesmo tamanho.

A função retorna o valor extrapolado de y para o ponto x_extrapolar.

É importante lembrar que antes de executar o arquivo exemolo.py, para este método, é preciso informar o x_extrapolar na linha 52.

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 maneira, as primeiras linhas até o "*/n*" contêm os valores para o primeiro vetor com os valores de x e nas linhas subsequentes os

valores para y. As informações sobre as entradas podem ser verificadas no arquivo "*entrada_Richards.txt*".

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

```
Nome do método
Resultados
```

Problemas

Exemplo 1

Como dados de entrada temos os seguintes:

0.1	0.62
0.2	0.63
0.3	0.64
0.4	0.66
0.5	0.68
0.6	0.71
0.7	0.76
0.8	0.81
0.9	0.89
1	1

Como já supracitado, a primeira coluna corresponde a lista dos valores de *x* e a segunda a de *y*, respectivamente.

Após a rodar o algoritmo com valor da variável x_extrapolar = 6 foi obtida a seguinte saída:

> Valor extrapolado de y para x = 6: [-1.78141117e+06 9.09876168e+20 1.53586348e+05 9.09876168e+20
> 4.46814981e+05 2.89995618e+05 -4.54938084e+20 3.40488995e+05
> 6.74528141e+05]

Exemplo 2

Para este segundo exemplo foi utilizado o seguinte dado de entrada:

1980	248.8
1985	398
1989	503.7
1992	684.9
1994	749.9
1995	793.5
1997	865.7

Mantendo os valores de extrapolação do exemplo anterior, foram obtidos os seguintes resultados:

> Valor extrapolado de y para x = 6: [1.38605727e+20 4.07394412e+20 -1.50117920e+19 1.51112489e+20
> -6.75714719e+19 4.19061990e+20]

Dificuldades enfrentadas

Não foi possível ler o valor da variável x_extrapolar no arquivo de entrada. Infelizmente, sempre que a função ler do respectivo método era executada pelo script, era retornado um erro que não consegui resolver em tempo hábil.

Considerações finais

Programar requer uma atenção meticulosa par lidar com variáveis, gerenciar a memória e criar iterações em loops. Na criação de códigos que gerem resultados precisos, percebe-se que a programação em si é apenas umas das etapas desse processo, uma vez que uma compreensão profunda do problema abordado é essencial para alcançar resultados relevantes.

Para além disso, é de grande importância destacar a relevância da comparação dos resultados de diferentes métodos, que contribui para uma compreensão mais aprofundada do uso. Observa-se que alguns métodos se destacam no requisito velocidade e alguns outros na precisão. Entretanto, ressalto que todos os métodos que foram desenvolvidos ao longo deste relatório produziram resultados satisfatórios.