Universidade Estadual de Santa Cruz – UESC

Relatório de Implementações de Métodos da Disciplina Análise Numérica - parte II

Relatório de implementações realizadas por Flávia Alessandra S J.

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2023.2

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2023

ÍNDICE

Linguagem Escolhida e justificativas

Python, Visual Studio Code e Geogebra:

A escolha da linguagem Python foi motivada pela sua facilidade em lidar com fórmulas matemáticas e operações de leitura e escrita de arquivos. A vantagem de utilizar Python se destaca principalmente no manuseio de fórmulas matemáticas, pois a função "eval" disponível na linguagem facilita o processamento dessas equações, especialmente quando elas são carregadas a partir de arquivos externos.

No que diz respeito à manipulação de arquivos, Python oferece funções integradas que tornam a leitura e escrita de dados muito simples, graças à sua biblioteca padrão. Não encontramos nenhuma limitação ao usar essa linguagem; ela atendeu a todas as nossas necessidades com eficiência.

A versão do Python utilizada foi a 3.7.8. O interpretador pode ser baixado em: https://www.python.org e a instalação das bibliotecas pode ser feita no terminal do VScode (ou prompt de comando) pelo seguinte comando (após a instalação do interpretador):

- ➔ pip install sympy
- ➔ pip install numpy

Uma vez instalado, não há necessidade de chamar esses comandos novamente.

Foi preferível utilizar o editor de texto Visual Studio Code, que possui um compilador interno para várias linguagens. A única exigência foi instalar o Python na máquina, juntamente com quaisquer bibliotecas externas necessárias, caso existissem. Esse ambiente pode ser baixado em https://code.visualstudio.com.

Testes e Funcionamento do Código:

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

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

Regressão Linear

Estratégia de Implementação:

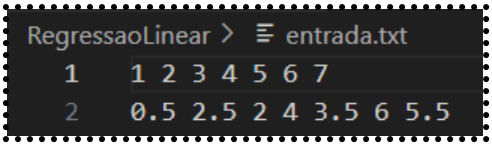
A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *math* apenas para cálculos. Esse método não requer muitos recursos externos para a implementação.

O procedimento consiste em armazenar os valores de x e y, permitindo-nos calcular a soma dos valores de x, y, x*y e x^2. Com esses dados armazenados em variáveis autoexplicativas, aplicamos a fórmula da regressão linear para determinar os valores de A.

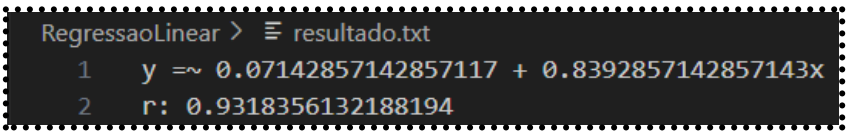
Para os arquivos de saída e entrada foi pensado que por padrão, o polinômio estaria na primeira linha e o coeficiente de correlação na última, enquanto que no arquivo de entrada, os valores de x e y estariam armazenados em linhas diferentes. Essa formatação do arquivo de entrada foi pensada visando para que ocupasse o menor espaço possível, ao mesmo tempo em que fica claro que trata-se de tipos de dados diferentes.

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 em si. Essas informações serão editadas no arquivo “entrada.txt”.



A saída contém na primeira linha, o polinômio e na segunda, o coeficiente de correlação. Essas informações atualizarão o arquivo “resultado.txt”.



Problema teste 8.1:

(a) Tendo como dados de entrada:

x	1980 1985 1990 1993 1994 1996 1998
y	8300 9900 10400 13200 13600 13700 14600

Sendo ***r*** o coeficiente de correlação e ***y*** a equação da reta. A seguir a saída:

y =~	-709699.5332555426 + 362.48541423570595x
r	0.9685408745259411

Substituindo x por 2000. Temos a previsão de acidentes que é 15.271.

(c) No contexto, só posso acreditar no resultado de a, já que não tenho implementações de mmq.

Dificuldades enfrentadas

Para a implementação desse método, não houve dificuldade significativa.

Polinômio de Lagrange

Estratégia de Implementação:

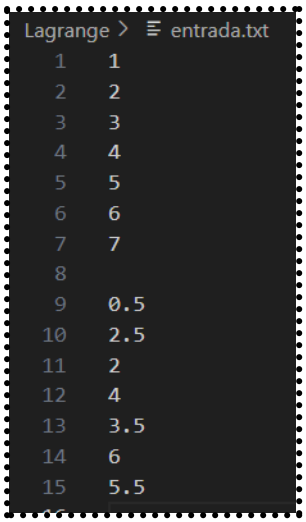
A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *sympy sp* para a conversão das str(x) em variável simbólica.

Primeiramente, realizamos o cálculo dos polinômios de Lagrange utilizando a função "Polinomios" durante a iteração atual do método Lagrange. Isso envolve os valores dos pontos e o tamanho da lista em consideração. A função "Polinomios" é invocada repetidamente dentro da função "lagrange", onde é calculada utilizando os valores de "fx", "pontos" e "Poly" da biblioteca sympy para auxiliar na obtenção dos polinômios.

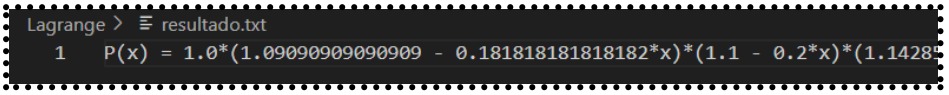
A formatação do arquivo de entrada destoa da maneira anterior para mostrar que esse tipo de variação na leitura dos dados não interfere significativamente no desempenho do algoritmo. Logo, para essa primeira parte pode ser levado em consideração o fator estético/organizacional.

Estrutura dos Arquivos de Entrada/Saída

O arquivo de entrada é organizado de maneira que em cada linha esteja armazenado um dos dados necessários para o cálculo do método. Dessa forma, na primeira linha se encontra o primeiro elemento (de x), na segunda, o segundo elemento (de x), por fim, pula uma linha e usa a mesma formatação para os elementos de y. Essas informações serão editadas no arquivo “entrada.txt”



Para o arquivo de saída, é escrito o polinômio resultado do método em uma única linha:



Problema teste 8.1:

(a) Tendo como dados de entrada:

x	1980 1985 1990 1993 1994 1996 1998
y	8300 9900 10400 13200 13600 13700 14600

O polinômio resultante é:

P(x) = -5.3573e-19*x**6 + 3.7415e-14*x**5 - 1.080e-9*x**4 + 1.6506e-5*x**3 - 0.1406*x**2 + 634.2090*x - 1179098.6565

(b) P(x) = 2.75580107267847e-9*x**6 - 2.45470009401054e-5*x**5 + 0.0910028885587479*x**4 - 179.739074531462*x**3 + 199479.535960257*x**2 - 117954009.141777*x + 29032807088.2871

Problema teste 8.5:

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

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

Diferenças Divididas de Newton

Estratégia de Implementação:

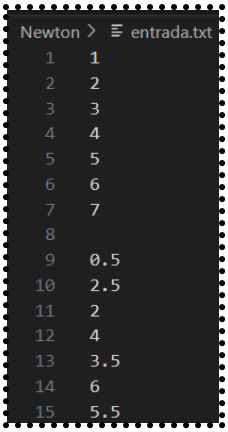
A interação com os dados foi feita utilizando arquivos externos (.txt). Foi necessário o uso da biblioteca *sympy sp* para a conversão das str(x) em variável simbólica e *math* para cálculos.

A princípio, efetuamos o cálculo da função "f" nos pontos utilizando a função "calculaFdosPontos". Esta função é invocada repetidamente dentro da função "newton", onde o resultado é empregado na composição dos valores de "x" na função.

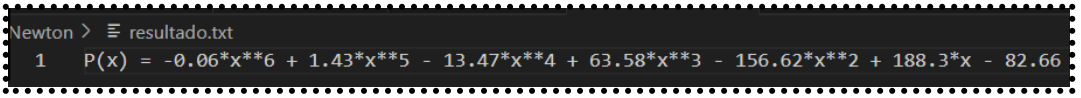
A formatação do arquivo de entrada se manteve como da maneira anterior por tratar-se de operações mais similares.

Estrutura dos Arquivos de Entrada/Saída:

O arquivo de entrada é organizado de maneira que em cada linha esteja armazenado um dos dados necessários para o cálculo do método. Dessa forma, na primeira linha se encontra o primeiro elemento (de x), na segunda, o segundo elemento (de x), por fim, pula uma linha e usa a mesma formatação para os elementos de y. Essas informações serão editadas no arquivo “entrada.txt”



Para o arquivo de saída, é escrito o polinômio resultado do método em uma única linha:



Problema teste 8.11:

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

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

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 o bom funcionamento dos cálculos.

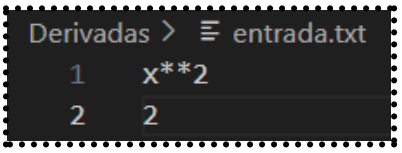
A lógica usada para o cálculo da derivada de primeira ordem da função foi:

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

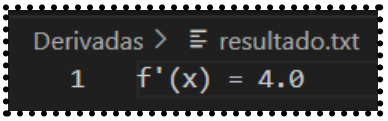
É fundamental destacar a importância de verificar a correspondência adequada entre símbolos e operações matemáticas em relação às funcionalidades de reconhecimento de funções do Python. Por exemplo, uma potenciação x^2, no txt deve ser escrita como: ***x**2***.

Estrutura dos Arquivos de Entrada/Saída:

Para o arquivo de entrada, a formatação é a seguinte: Na primeira linha se encontra a função e na segunda linha está o valor de x.



Para o arquivo de saída, temos a string “f’(x)” e o respectivo valor da derivada:



Problema teste 11.6:

(a) f'(x) = -2.6891076568197336e-05

(b) f'(x) = -4.325802857101102e-05

Problema teste 11.11:

(a) f'(x) = 0.0

Dificuldades enfrentadas:

Para a implementação desse método, não houve dificuldade significativa.

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 o bom funcionamento dos cálculos.

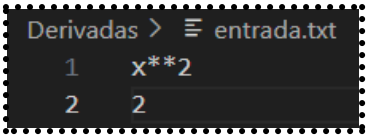
A lógica usada para o cálculo da derivada de primeira ordem da função foi:

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

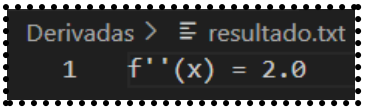
É crucial ressaltar a necessidade de assegurar a correspondência precisa entre símbolos e operações matemáticas quando se trata das capacidades de reconhecimento de funções do Python.

Estrutura dos Arquivos de Entrada/Saída:

Para o arquivo de entrada, a formatação é a seguinte: Na primeira linha se encontra a função e na segunda linha está o valor de x.



Para o arquivo de saída, temos a string “f’’(x)” e o respectivo valor da derivada:



Problema teste 11.6:

(a) f''(x) = -9070677423.524647

(b) f''(x) = -1793023634193238.0

Problema teste 11.11:

(a) f''(x) = -159.5769121605731

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

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.

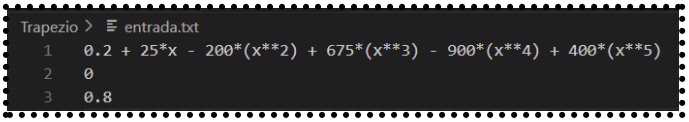
Para os arquivos de saída e entrada foi pensado que, no arquivo de entrada, todos os dados estejam armazenados em linhas diferentes: na primeira linha, a função, na segunda, o limite inferior e por fim, o limite superior. Essa formatação do arquivo de entrada foi pensada visando para que ocupasse o menor espaço possível, ao mesmo tempo em que fica claro que trata-se de tipos de dados diferentes.

Vale ressaltar que é importante consultar a equivalência representativa dos símbolos e operações matemáticas frente aos recursos de reconhecimento de funções do Python.

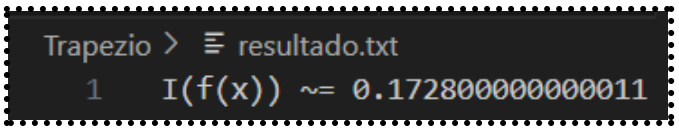
Estrutura dos Arquivos de Entrada/Saída

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 encontra a função, na segunda, o limite inferior, e enfim, o limite superior. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão apenas interfere mais na estética do que na organização em si. Essas informações serão editadas no arquivo “entrada.txt”.



Para o nosso arquivo de saída, temos somente o resultado da integral, sinalizada pela string I(f(x)):



Problema Teste 11.1:

I(f(x)) ~= 1.40084231444575

Problema Teste 11.6:

(a) I(f(x)) ~= 2277649.1031817*pi

Problema Teste 11.11:

I(f(x)) ~= 0.762338063966931/pi**0.5

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

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 e *math* para cálculos.

No início do programa, é calculado o valor de "h", em seguida, a função "intervalos" é chamada para gerar os valores dos intervalos. Posteriormente, o método do trapézio é aplicado utilizando os dados obtidos.

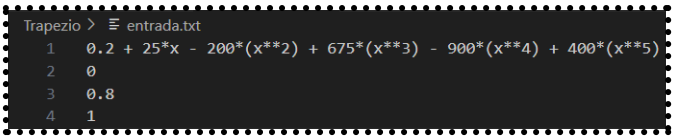
A formatação do arquivo de entrada foi pensada visando para que ocupasse o menor espaço possível, ao mesmo tempo em que fica claro que trata-se de tipos de dados diferentes.

Além disso, é necessário assegurar a correspondência precisa entre símbolos e operações matemáticas quando se trata das capacidades de reconhecimento de funções do Python.

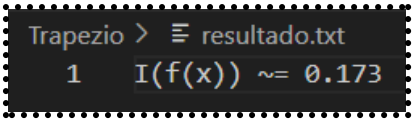
Estrutura dos Arquivos de Entrada/Saída

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 encontra a função, na segunda, o limite inferior,

na terceira, o limite superior, e por fim, o número de subintervalos. Essa estrutura foi pensada de maneira intuitiva, já que a ordem dessa dispersão interfere mais na estética do que na organização em si. Essas informações serão editadas no arquivo “entrada.txt”.



Para o nosso arquivo de saída, temos somente o resultado da integral, sinalizada pela string I(f(x)):



Problema teste 11.1:

I(f(x)) ~= 0.680

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

Simpson 1/3

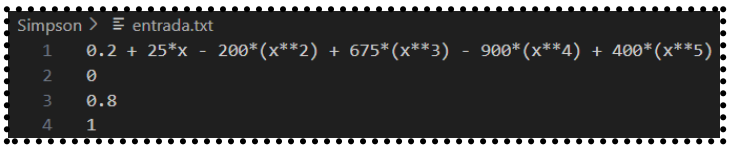
Estratégia de Implementação:

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 e *math* para cálculos.

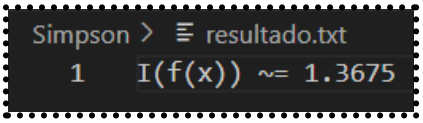
O programa inicialmente calcula o valor de h, depois chama-se a função intervalos, que irá gerar os valores dos intervalos. Logo após é aplicado o método de Simpson 1/3 em si com os dados obtidos.

Estrutura dos Arquivos de Entrada/Saída:

O formato de entrada do arquivo está como entrada.txt, nele será colocado em cada linha um valor. Primeiro a função, segundo o intervalo a, terceiro o intervalo b e por último o número de subintervalos.



O formato de saída está em um arquivo denominado resultado.txt, nele será escrito a resposta em uma única linha como I(f(x)):



Problema Teste 11.1:

I(f(x)) ~= 0.6370

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

Simpson 3/8

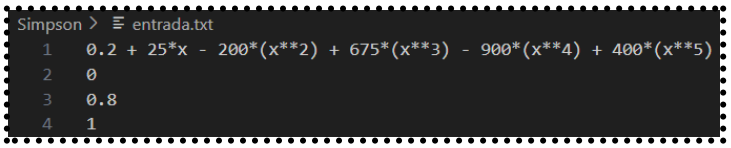
Estratégia de Implementação:

A manipulação dos dados envolveu o uso de arquivos externos no formato .txt. Para realizar essa tarefa, fez-se uso da biblioteca sympy para converter as strings "x" em variáveis simbólicas, além da biblioteca math para efetuar os cálculos necessários.

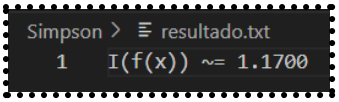
O programa inicialmente calcula o valor de h, depois chama-se a função intervalos, que irá gerar os valores dos intervalos. Logo após é aplicado o método de Simpson 3/8 em si com os dados obtidos.

Estrutura dos Arquivos de Entrada/Saída:

O formato de entrada do arquivo está como entrada.txt, nele será colocado em cada linha um valor. Primeiro a função, segundo o intervalo a, terceiro o intervalo b e por último o valor do subintervalo.



Já o formato de saída está em um arquivo denominado resultado.txt, nele será escrito a resposta em uma única linha como I(f(x)).



Problema Teste 11.1:

I(f(x)) ~= 0.6350

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

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.

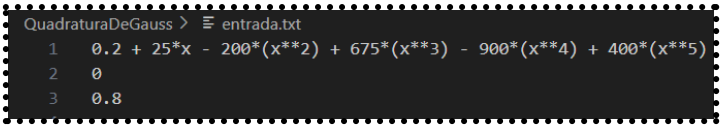
Para os arquivos de saída e entrada foi pensado que, no arquivo de entrada, todos os dados devem ser armazenados em linhas diferentes: na primeira linha, a função, na segunda o limite inferior, na terceira o limite superior.

O método consiste em usar a fórmula para a integral:

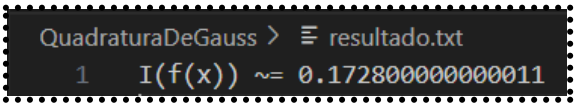
$$I = \frac{(b - a)f(a) + (b - a)f(b)}{2}$$

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 forma, na primeira linha se encontra a função, na segunda, o limite inferior, e enfim, o limite superior. Essas informações serão editadas no arquivo “entrada.txt”.



Para o arquivo de saída, temos o resultado da integral com a string identificadora I(f(x)):



Problema Teste 11.1:

I(f(x)) ~= 1.40084231444575

Dificuldades enfrentadas:

A maior dificuldade foi na compreensão dos exercícios. Para a implementação desse método, não houve dificuldade significativa.

Considerações Finais

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

Além disso, é fundamental destacar a relevância da comparação entre os resultados de diferentes métodos, o que contribui para uma compreensão mais aprofundada do seu uso.

Ao longo das implementações, observa-se que alguns métodos se destacaram em termos de precisão, enquanto outros foram mais rápidos. No entanto, é importante ressaltar que todos os métodos que foram desenvolvidos e abordados nesse relatório produziram resultados satisfatórios.