****

**Universidade Estadual de Santa Cruz – UESC**

**Relatórios de Implementações de Métodos da Disciplina Análise Numérica**

**Relatório de implementações realizadas por Iago Gomes Santana**

**Disciplina Análise Numérica.**

**Curso Ciência da Computação**

**Semestre 2031.1**

**Professor Gesil Sampaio Amarante II**

**Ilhéus – BA**

**2023**

ÍNDICE

[**Linguagem Escolhida e justificativas: 4**](#_heading=h.evje1lcwdaly)

[**Arquitetura: 5**](#_heading=h.oivl2yubs3pk)

[**Regressão Linear 6**](#_heading=h.2z5bfzhao93i)

[Estratégia de Implementação: 6](#_heading=h.591oldvrkm39)

[Estrutura dos Arquivos de Entrada/Saída 6](#_heading=h.e3zz9zq3c1cg)

[Problema teste 7](#_heading=h.cekoel9npo7y)

[Dificuldades enfrentadas 8](#_heading=h.hao5zwmylzps)

[**Aproximação Polinomial 8**](#_heading=h.yg2kblq4ndnb)

[Estratégia de Implementação: 8](#_heading=h.7me2dm5okr6f)

[Estrutura dos Arquivos de Entrada/Saída 10](#_heading=h.7mkxxg4vf8qf)

[Problema teste 11](#_heading=h.qrp60m9qifbv)

[Dificuldades enfrentadas 12](#_heading=h.613c9v57c6pa)

[**Interpolação por Polinômios de Lagrange 12**](#_heading=h.siiakp9r3ec)

[Estratégia de Implementação: 12](#_heading=h.s7bhzwa0jdx4)

[Estrutura dos Arquivos de Entrada/Saída 13](#_heading=h.on2vazq7gsie)

[Problema teste 14](#_heading=h.cmad08j11qtc)

[Dificuldades enfrentadas 15](#_heading=h.dw1249swp4jt)

[**Interpolação por diferenças divididas de Newton 15**](#_heading=h.i24bxp106htb)

[Estratégia de Implementação: 15](#_heading=h.ozpufinp1q7t)

[Estrutura dos Arquivos de Entrada/Saída 16](#_heading=h.bfz43n1zpchm)

[Problema teste 1, 2, 3... 17](#_heading=h.7uql4m51603b)

[Dificuldades enfrentadas 17](#_heading=h.uo68wftieh7)

[**Derivação Numérica 17**](#_heading=h.wymk4vby9url)

[Estratégia de Implementação: 17](#_heading=h.mdoe2fx0vr6t)

[Estrutura dos Arquivos de Entrada/Saída 19](#_heading=h.t3r5jkvi588j)

[Problema teste 1, 2, 3... 20](#_heading=h.v1bfa5abk73h)

[Dificuldades enfrentadas 20](#_heading=h.x4q9md5g03hv)

[**Integração por trapézio 20**](#_heading=h.5sjgxyejc0i4)

[Estratégia de Implementação: 20](#_heading=h.qqp01ogyosuw)

[Estrutura dos Arquivos de Entrada/Saída 22](#_heading=h.zhxvkbrufez9)

[Problema teste 1, 2, 3... 23](#_heading=h.aw9hk729wneg)

[Dificuldades enfrentadas 23](#_heading=h.9otumt7984vt)

[**Simpson de 1/3 23**](#_heading=h.xymvv2utmpq7)

[Estratégia de Implementação: 23](#_heading=h.si5qsyb11w7o)

[Estrutura dos Arquivos de Entrada/Saída 24](#_heading=h.2y7hj2qkhpiv)

[Problema teste 1, 2, 3... 25](#_heading=h.dmpry9s5mmw9)

[Dificuldades enfrentadas 25](#_heading=h.8tbgt5o6pmex)

[**Simpson de 3/8 25**](#_heading=h.ftjc3xjl0sau)

[Estratégia de Implementação: 25](#_heading=h.i3buqz89ff3s)

[Estrutura dos Arquivos de Entrada/Saída 26](#_heading=h.oy1qj5lcro9j)

[Problema teste 1, 2, 3... 27](#_heading=h.twzsg71nxmv6)

[Dificuldades enfrentadas 27](#_heading=h.cqkdzit3tbxi)

[**Extrapolação de Richards 27**](#_heading=h.pcwixy5ng1yi)

[Estratégia de Implementação: 27](#_heading=h.uelwkfh6okhz)

[Estrutura dos Arquivos de Entrada/Saída 28](#_heading=h.yxilg5ad7w58)

[Problema teste 1, 2, 3... 29](#_heading=h.omjmkufyratq)

[Dificuldades enfrentadas 29](#_heading=h.w56i5nyb8q8l)

[**Considerações Finais 29**](#_heading=h.qekoipljz12j)

# Linguagem Escolhida e justificativas:

A linguagem escolhida foi o Python (versão 3.8.10) por sua facilidade em lidar com tipos diferentes de dados e praticidade ao programar algumas funções matemáticas. A lib “math” me foi muito útil tal como foi a “numpy” em alguns casos.

Outro ponto relevante para a escolha foi o domínio da linguagem, uma vez que as implementações exigem não só um bom conhecimento do método, mas também da ferramenta usada para programá-lo, python foi a escolha óbvia para mim.

E por fim, praticidade e agilidade. Por estar conciliando trabalho com estudos, meu tempo é muito mais curto do que gostaria, e o python agiliza muito na velocidade de programação, o que para mim é vital, visto que preciso render o dobro em metade do tempo para dar conta das matérias.

Uma desvantagem que percebi no python foi que, em alguns casos, quando envolvem operações com números decimais com muitos dígitos após a vírgula, os resultados podem ser imprecisos. Porém é possível contornar esse déficit utilizando a libs específicas para cálculo de alta precisão.

Outro ponto negativo para o python é que talvez ele não se adeque em alguns cenários de implantação para estes métodos. Isso pois python não é conhecido por ser uma linguagem otimizada, este título pertence a C. Porém, a perda de performance do python não afetará em nada a resolução dos problemas propostos tendo em vista o escopo e a necessidade.

No mais, python é uma boa escolha no geral por ser uma linguagem de sintaxe enxuta e ágil para se programar, não chega a ser tão eficiente quanto C, mas para o nosso escopo de problemas, essa perda na eficiência não será impactante.

# Arquitetura:

Por ser um projeto com cada método tendo sua especificidade a ser implementada, ao mesmo tempo que cada um dispõe de semelhanças tais quais a leitura e escrita em arquivo e o método que calcula f(x), optei por uma estrutura que se baseia em:

* Ter um diretório raiz chamado “metodos\_numericos” e dentro dele tendo mais 3 diretórios
* implementations:
  + Aqui eu tenho 1 arquivo python para cada método implementado. Nesses arquivos temos apenas a implementação das especificidades do método em si.
  + Temos também o diretório “utils” que possui 2 arquivos python, com a implementação de uma classe para leitura e escrita em arquivo de texto, tal como um pacote para realizar o cálculo de uma função de X (ou f(x) ), á partir de uma string e um valor para X
* Inputs:
  + Diretório onde devem ser criados os arquivos de inputs para cada método. A princípio cada input deve ter o nome de seu respectivo método, mas isso pode ser mudado no decorrer do desenvolvimento.
* Outputs:
  + Diretório de saída onde o programa escreverá os resultados dos cálculos. Novamente cada arquivo terá o nome de seu respectivo método, mas isso também poderá mudar até a data de entrega do relatório.

Esta é a estrutura básica do projeto, vale salientar mais 2 arquivos na raiz que podem ser úteis, o ‘README.md’, que irá conter tutoriais e explicações. E o ‘makefile’, que auxilia na execução dos programas.

**OBS.:** os comandos make só funcionam no linux, caso use windows, pode aproveitar a linha do comando dentro do makefile para executar o código.

# Regressão Linear

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

A estratégia de implementação para a regressão linear apresentada segue os seguintes passos:

1. Cálculo de algumas somas estatísticas:
   1. **sum\_x** é a soma dos valores da variável independente x.
   2. **sum\_y** é a soma dos valores da variável dependente y.
   3. **sum\_exp\_x** é a soma dos quadrados dos valores de x.
   4. **sum\_xy** é a soma dos produtos dos valores de x e y.
2. Cálculo dos coeficientes da regressão linear:
   1. **a1** é o coeficiente angular da reta de regressão e é calculado utilizando a fórmula: (lengh \* sum\_xy - sum\_x \* sum\_y) / (lengh \* sum\_exp\_x - sum\_x \*\* 2).
   2. **a0** é o coeficiente linear da reta de regressão e é calculado utilizando a fórmula: sum\_y / lengh - a1 \* (sum\_x / lengh).
3. Retorno dos coeficientes calculados: A função retorna os coeficientes **a0** e **a1**, que representam os coeficientes linear e angular da reta de regressão linear, respectivamente.

A estratégia de implementação utiliza as fórmulas matemáticas da regressão linear para calcular os coeficientes diretamente a partir dos dados fornecidos. A função linear\_regression não faz uso de bibliotecas externas e implementa a regressão linear de forma simples e direta.

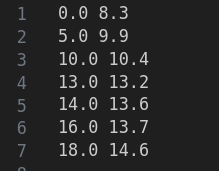
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, o arquivo de entrada e de saída devem se chamar “regression.txt”

**Entrada**:

Para este método, a entrada deve conter uma sequência de números semelhante a em uma tabela, desde que siga o padrão “x y” por linha, como no exemplo abaixo. O final de uma tabela é delimitado por uma linha em branco



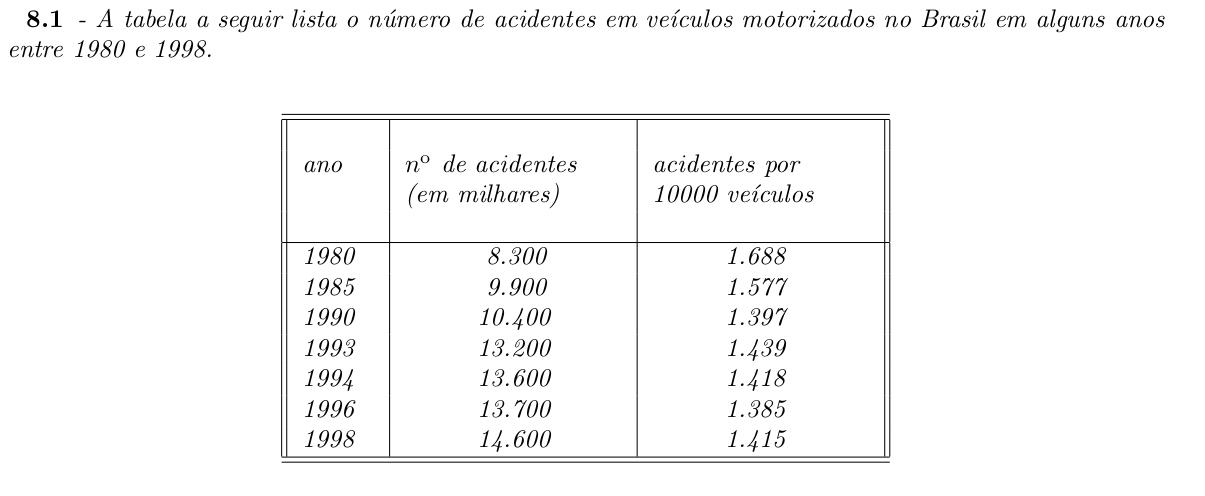
**Saída:**

E a saída é basicamente N linhas compostas por uma reta no padrão “y = a + bX”, como no exemplo abaixo:



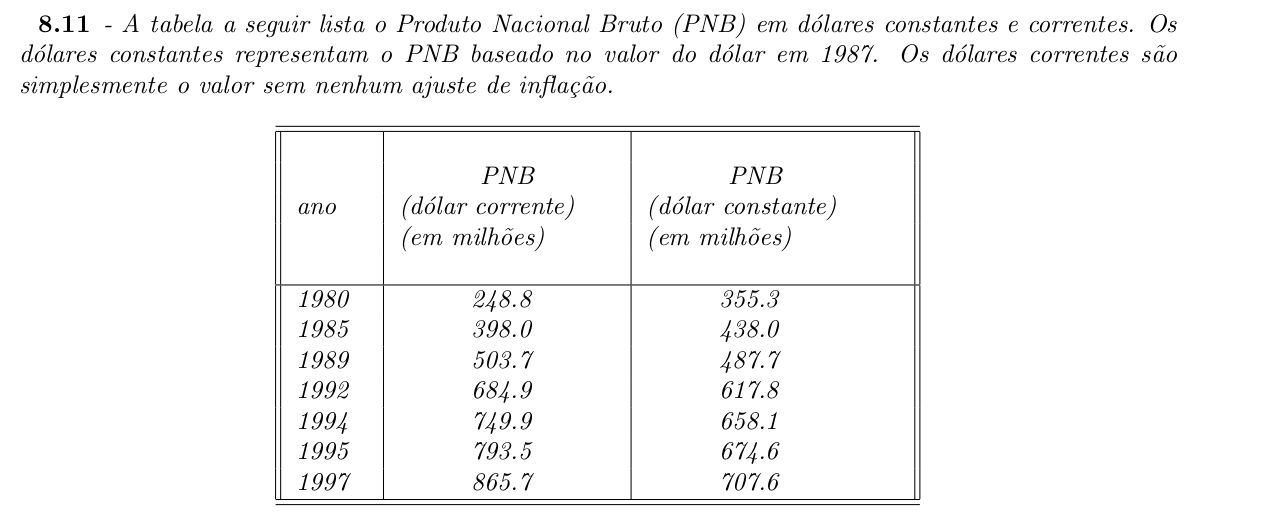
## Problema teste

1. 8.1 - A)



Ao executar o meu algoritmo de regressão linear, obtive a seguinte reta:  
y = -709.6995332555966 + 0.3624854142357331X

Substituindo 2000 em X, obtive o resultado de: 15,271295216 acidentes

1. 8.11 A)  
     
   Ficou um pouco confuso, mas interpretei que a questão pede duas retas que descrevam o comportamento do PNB ao longo do tempo, e após executar o algorítimo, cheguei aos resultados:  
   y = -73791.8445312442 + 37.38066406249709X  
   y = -43319.83203125579 + 22.048372395836243X
2. 8.11 B)  
   Em 2000, a previsão para PNB (dólar corrente) é de 969,48  
   Já para PNB (constante) é de 776,91

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Aproximação Polinomial

## Estratégia de Implementação:

Para este tópico, implementei 2 métodos diferentes, um para o caso discreto, outro para o caso contínuo. O que difere os dois é basicamente o método principal que explicarei separadamente, mas de resto, o que não for explicitamente separado serve para ambos.

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

**MMQ Discreta:**

A estratégia de implementação para o método dos mínimos quadrados (escopo discreto) apresentada no código segue os seguintes passos:

1. Cálculo dos vetores de base:
   1. **u0** é um vetor composto por 1's, com o mesmo comprimento que os vetores de entrada x e y.
   2. **u1** é um vetor composto pelos valores do vetor x.
2. Cálculo de algumas somas estatísticas:
   1. **sum\_x** é a soma dos valores do vetor x.
   2. **sum\_y** é a soma dos valores do vetor y.
   3. **sum\_xy** é a soma dos produtos dos valores correspondentes de x e y.
   4. **sum\_xx** é a soma dos quadrados dos valores do vetor x.
   5. **sum\_u0** é a soma dos valores do vetor u0 (que é igual ao comprimento de x).
   6. **sum\_u1** é a soma dos valores do vetor u1 (que é igual à soma dos valores de x).
3. Cálculo dos coeficientes da regressão linear pelo método dos mínimos quadrados:
   1. **a** é o coeficiente angular da reta de regressão e é calculado utilizando a fórmula: (lengh \* sum\_xy - sum\_x \* sum\_y) / (sum\_xx \* lengh - sum\_x\*\*2).
   2. **b** é o coeficiente linear da reta de regressão e é calculado utilizando a fórmula: (sum\_xx \* sum\_y - sum\_xy \* sum\_x) / (sum\_xx \* lengh - sum\_x\*\*2).
4. Retorno dos coeficientes calculados: A função retorna os coeficientes **b** e **a**, que representam os coeficientes linear e angular da reta de regressão linear pelo método dos mínimos quadrados.

A estratégia de implementação utiliza as fórmulas matemáticas do método dos mínimos quadrados para calcular os coeficientes diretamente a partir dos vetores de entrada x e y. A função mmq não faz uso de bibliotecas externas e implementa o método dos mínimos quadrados de forma simples e direta.

**MMQ Continua:**

A estratégia de implementação para o método dos mínimos quadrados (escopo contínuo) apresentada no código segue os seguintes passos:

1. Definição do número de pontos e das matrizes:
   1. **n** é o número de pontos (tamanho dos vetores x e y).
   2. **A** é uma matriz de tamanho n x (m + 1) inicializada com zeros. Cada elemento de A é calculado como x[i] \*\* j, onde i percorre os pontos e j percorre as potências de x até m.
   3. **b** é um vetor de tamanho n inicializado com zeros. Cada elemento de b é igual ao valor correspondente de y[i].
2. Resolução do sistema de equações:
   1. Utiliza-se a função np.linalg.lstsq para encontrar os coeficientes a e b que minimizam o erro quadrático entre os pontos e a função aproximada.
   2. O resultado da função np.linalg.lstsq retorna uma tupla contendo os coeficientes, sendo [0] utilizado para obter o coeficiente a.
3. Retorno dos coeficientes calculados: A função retorna os coeficientes a e b, que representam os coeficientes da função aproximada pelo método dos mínimos quadrados.

A estratégia de implementação utiliza a construção de uma matriz A com as potências de x e a resolução de um sistema de equações lineares utilizando a função np.linalg.lstsq para encontrar os coeficientes da função aproximada. A função mmq utiliza a biblioteca NumPy para manipulação de matrizes e cálculos lineares.

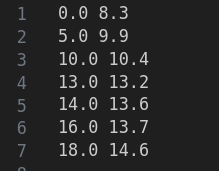
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída do caso discreto devem se chamar “mmq\_discreet.txt”. Já para o caso contínuo o nome é “mmq\_continuos.txt”

**Entrada**:

Para este método, a entrada deve conter uma sequência de números semelhante a em uma tabela, desde que siga o padrão “x y” por linha, como no exemplo abaixo. O final de uma tabela é delimitado por uma linha em branco.

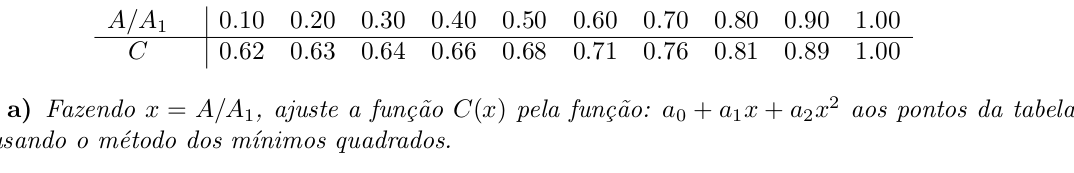
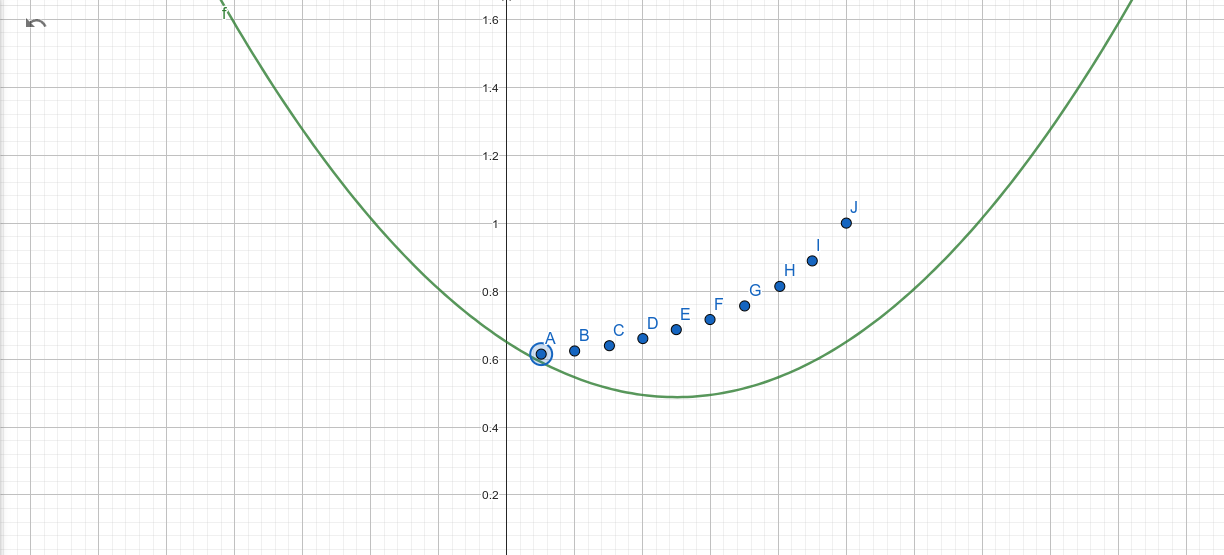


**Saída:**

E a saída é basicamente N linhas compostas por uma reta no padrão “y = a + bX”, como no exemplo abaixo:



## Problema teste

1. 8.5 A)   
     
   Ao fazer as devidas adaptações e executar o método mmq\_continuos, obtive o seguinte resultado:  
   f2(x) = 0.65050 - 0.65050x^1 + 0.65050x^2
2. 8.5 B)  
   A questão pede um gráfico e aqui está ele:  
     
   E acredito que o polinômio apesar de descrever relativamente bem o comportamento, não descreve bem os valores, precisaria ser um polinômio de grau maior para alcançar tal feito.

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Interpolação por Polinômios de Lagrange

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

A estratégia de implementação para o método de interpolação através de polinômios de Lagrange, apresentada no códig, segue os seguintes passos:

1. Construção dos polinômios de Lagrange:
   1. A função build\_polynomial recebe uma lista de pontos list\_x e itera sobre eles para construir os polinômios de Lagrange correspondentes.
   2. Para cada ponto list\_x[i], calcula-se o polinômio de Lagrange lk dividindo o resultado da avaliação de evaluate\_lk pela avaliação de evaluate\_lk no próprio ponto list\_x[i].
   3. Os polinômios de Lagrange são armazenados na lista result.
2. Avaliação dos polinômios de Lagrange para obter o resultado da interpolação:
   1. A função lagrange recebe os valores y correspondentes à função nos pontos list\_x e a lista de polinômios de Lagrange list\_lk.
   2. Inicializa-se a variável result como zero para armazenar a soma dos produtos dos valores de y pelos polinômios de Lagrange.
   3. Itera-se sobre os índices i dos valores de y e soma-se y[i] \* list\_lk[i] ao resultado.
   4. O resultado é um polinômio obtido pela multiplicação dos valores de y pelos polinômios de Lagrange.
   5. Utiliza-se a função Poly(result, X).coeffs() para obter os coeficientes do polinômio.
3. Construção da representação do polinômio interpolador:
   1. Itera-se sobre os coeficientes do polinômio e para cada coeficiente polynomial[i], constrói-se um termo multiplicando-o pela potência correspondente de X.
   2. Verifica-se o sinal do coeficiente e adiciona-se o termo ao resultado response na forma apropriada.

A estratégia de implementação utiliza os polinômios de Lagrange para interpolar uma função em uma lista de pontos. A função lagrange retorna uma representação do polinômio interpolador, enquanto a função build\_polynomial constrói os polinômios de Lagrange para cada ponto. A biblioteca SymPy é utilizada para manipulação simbólica e cálculos de polinômios.

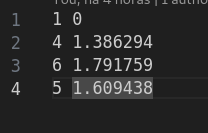
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída devem se chamar “lagrange.txt”.

**Entrada**:

Para este método, a entrada deve conter uma sequência de números semelhante a em uma tabela, desde que siga o padrão “x y” por linha, como no exemplo abaixo. O final de uma tabela é delimitado por uma linha em branco.



**Saída:**

E a saída é basicamente N linhas compostas por um polinômio, como no exemplo abaixo:



## Problema teste

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Interpolação por diferenças divididas de Newton

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

A estratégia de implementação para o método de Interpolação por diferenças divididas de Newton, presente no código, segue os seguintes passos:

1. Cálculo dos coeficientes:
   1. A função calc\_bn calcula os coeficientes de diferenças divididas de Newton para uma lista de pontos list\_x e seus respectivos valores list\_y.
   2. O cálculo é feito de forma recursiva, verificando se os coeficientes já foram calculados e armazenados em REFERENCE\_TABLE.
   3. Se os coeficientes não estiverem disponíveis em REFERENCE\_TABLE, são calculados a partir das diferenças divididas dos pontos subsequentes e anteriores.
   4. O resultado é armazenado em REFERENCE\_TABLE para reutilização futura.
2. Resolução dos coeficientes:
   1. A função resolve\_coefficients recebe os coeficientes calculados e a lista de pontos list\_x.
   2. Utiliza a biblioteca SymPy para construir a expressão polinomial interpoladora.
   3. Itera sobre os coeficientes e para cada coeficiente coefficients[i], constrói-se um termo multiplicando-o pela potência correspondente de x - list\_x[j], onde j percorre os pontos anteriores a i.
   4. A expressão final é a soma de todos os termos construídos.
3. Construção da representação do polinômio interpolador:
   1. A função ddn recebe as listas de pontos x e seus respectivos valores y.
   2. Inicialmente, os coeficientes iniciais são definidos como os primeiros valores de y.
   3. Inverte-se as listas x e y para facilitar os cálculos das diferenças divididas.
   4. Itera-se sobre os índices de length - 2 até 0, calculando os coeficientes de diferenças divididas para cada ponto.
   5. Utiliza-se a função resolve\_coefficients para obter a expressão polinomial interpoladora simplificada.
   6. A representação final do polinômio interpolador é retornada como uma string.

A estratégia de implementação utiliza o método das diferenças divididas de Newton para calcular os coeficientes e construir o polinômio interpolador. A biblioteca SymPy é utilizada para manipulação simbólica e cálculos de polinômios. O armazenamento em REFERENCE\_TABLE evita cálculos repetidos e melhora a eficiência do algoritmo.

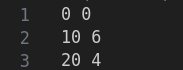
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída devem se chamar “newton\_dd.txt”.

**Entrada**:

Para este método, a entrada deve conter uma sequência de números semelhante a em uma tabela, desde que siga o padrão “x y” por linha, como no exemplo abaixo. O final de uma tabela é delimitado por uma linha em branco.



**Saída:**

E a saída é basicamente N linhas compostas por um polinômio, como no exemplo abaixo:



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Derivação Numérica

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

Para este tópico, foi implementado para um mesmo caso, 4 formas de responder, sendo elas a derivada simples retardada, a derivada simples centrada, a derivada simples progressiva e a derivada de segunda ordem. para tanto, fiz todo o código em um único arquivo e separei seus outputs organizadamente no arquivo de saída.

A estratégia de implementação para o método de derivação numérica, presente no código, segue os seguintes passos:

1. Cálculo da derivada de primeira ordem:
   1. A função derivate calcula a derivada de primeira ordem de uma função function em um ponto central middle\_x.
   2. A função utiliza a diferença progressiva (forward) para calcular a derivada no ponto middle\_x.
   3. A diferença progressiva é obtida subtraindo o valor da função avaliada em left\_x do valor da função avaliada em middle\_x, dividido pelo passo h.
   4. A função solve\_func é utilizada para calcular o valor da função em um ponto específico.
2. Cálculo da derivada de primeira ordem (progressiva, retardada e centrada):
   1. A função first\_order calcula a derivada de primeira ordem utilizando três pontos: left\_x, middle\_x, e right\_x.
   2. O ponto central é obtido como a média de left\_x e right\_x.
   3. O passo h é definido como a diferença entre middle\_x e left\_x.
   4. O passo h2 é definido como a diferença entre right\_x e left\_x.
   5. São calculadas as derivadas progressiva (forward), retardada (backward) e centrada utilizando a função derivate.
3. Cálculo da derivada de segunda ordem:
   1. A função second\_order calcula a derivada de segunda ordem utilizando três pontos: left\_x, middle\_x, e right\_x.
   2. O ponto central é obtido como a média de left\_x e right\_x.
   3. O passo h é definido como a diferença entre middle\_x e left\_x.
   4. São calculadas as derivadas de primeira ordem progressiva e retardada utilizando a função derivate.
   5. A derivada de segunda ordem é calculada utilizando a diferença progressiva entre as derivadas de primeira ordem.
4. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função eval.

A estratégia de implementação utiliza o conceito de diferenças finitas para calcular as derivadas numéricas. A função derivate implementa a diferença progressiva, enquanto as funções first\_order e second\_order utilizam combinações de diferenças progressivas e centrais para calcular as derivadas de primeira e segunda ordem. A função solve\_func permite a avaliação da função em um ponto específico.

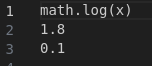
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída devem se chamar “derivation.txt”.

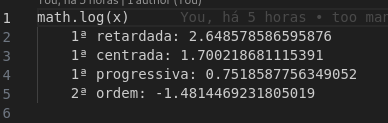
**Entrada**:

Por padrão, cada caso de teste deve conter 3 linhas e uma linha em branco. A primeira linha contém a função f(x), a segunda contém o segundo ponto do intervalo e a terceira contém o primeiro ponto do intervalo, como no exemplo abaixo:



**Saída:**

Já a saída foi organizada de modo intuitivo, sendo composta por 5 linhas, sendo a primeira a função original, e as 4 linhas posteriores as derivadas com legenda como no exemplo abaixo.



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Integração por trapézio

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

Para este tópico , optei por implementar a integral de trapézio simples em um arquivo separado da integral de múltiplos trapézios.  
  
**Trapézio simples:**

A estratégia de implementação para o método de integração por trapézio simples, presente no código, é a seguinte:

1. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função func\_parser.
2. Integração por trapézio:
   1. A função simple\_trapezoidal\_integration calcula a integral de uma função function no intervalo [a, b] utilizando o método dos trapézios.
   2. O valor da função nos pontos extremos do intervalo é obtido chamando a função solve\_func para avaliar a função nos pontos a e b, armazenando os resultados nas variáveis fa e fb, respectivamente.
   3. O tamanho do intervalo é calculado como a diferença entre b e a e armazenado na variável h.
   4. A integral é calculada utilizando a fórmula dos trapézios: a média dos valores da função nos pontos extremos multiplicada pelo tamanho do intervalo.
   5. O resultado da integração é retornado como uma string formatada contendo o valor da integral.

A estratégia de implementação utiliza a fórmula dos trapézios para estimar a integral de uma função no intervalo [a, b]. A função solve\_func permite a avaliação da função em um ponto específico substituindo o valor de x na expressão. A função simple\_trapezoidal\_integration aplica a fórmula dos trapézios utilizando os valores da função nos pontos extremos do intervalo, multiplicando pela largura do intervalo para obter a estimativa da integral.

**Múltiplos trapézios:**

A estratégia de implementação para o método de integração por trapézio múltiplo, presente no código, é a seguinte:

1. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função func\_parser.
2. Integração por trapézio múltiplo:
   1. A função multi\_trapezoidal\_integration calcula a integral de uma função function no intervalo [a, b] utilizando o método dos trapézios múltiplos.
   2. A variável h é calculada como a largura total do intervalo dividida por 10, o que resulta em 10 subintervalos iguais.
   3. Uma lista points é criada para armazenar os pontos dos subintervalos. O primeiro ponto é o limite inferior a, e os pontos subsequentes são calculados somando-se h ao ponto anterior.
   4. Em seguida, um loop é executado 10 vezes para calcular a contribuição de cada subintervalo para a integral total.
   5. Dentro do loop, o valor da função nos pontos extremos do subintervalo é obtido chamando a função solve\_func para avaliar a função nos pontos points[i] e points[i + 1], armazenando os resultados nas variáveis fa e fb, respectivamente.
   6. A contribuição de cada subintervalo é calculada utilizando a fórmula dos trapézios: a média dos valores da função nos pontos extremos multiplicada pelo tamanho do subintervalo h.
   7. A contribuição de cada subintervalo é somada à variável response, que acumula a integral total.
   8. O resultado da integração é retornado como uma string formatada contendo o valor da integral.

A estratégia de implementação utiliza a fórmula dos trapézios múltiplos para estimar a integral de uma função no intervalo [a, b] dividido em subintervalos de tamanho igual. A função solve\_func permite a avaliação da função em um ponto específico substituindo o valor de x na expressão. A função multi\_trapezoidal\_integration itera sobre os subintervalos, calculando a contribuição de cada subintervalo para a integral total utilizando a fórmula dos trapézios, e acumulando o resultado final.

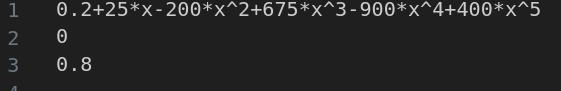
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída do método simples devem se chamar “itegration.txt”. Já para o caso de múltiplos trapézios, o nome é “multi\_itegration.txt”

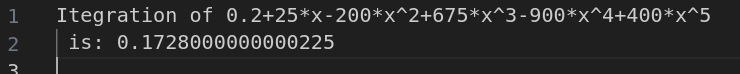
**Entrada**:

Por padrão, cada caso de teste deve conter 3 linhas e uma linha em branco. A primeira linha contém a função f(x), a segunda contém o segundo ponto do intervalo e a terceira contém o primeiro ponto do intervalo, como no exemplo abaixo:



**Saída:**

A saída é composta por uma resposta por 2 linhas de modo bem intuitivo como mostrado abaixo:



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Simpson de 1/3

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

A estratégia de implementação para o método de integração de Simpson de 1/3, com base no código, é a seguinte:

1. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função func\_parser.
2. Integração de Simpson de 1/3:
   1. A função simpson\_1\_3 calcula a integral de uma função function no intervalo [a, b] utilizando o método de Simpson de 1/3.
   2. O parâmetro n representa o número de subintervalos em que o intervalo [a, b] será dividido.
   3. O tamanho de cada subintervalo h é calculado como (b - a) / n.
   4. Uma lista points é criada para armazenar os pontos de divisão dos subintervalos. O primeiro ponto é o limite inferior a, e os pontos subsequentes são calculados adicionando-se h / n ao ponto anterior.
   5. Em seguida, um loop é executado n + 1 vezes para calcular os valores da função nos pontos extremos dos subintervalos.
   6. A função nos pontos extremos é calculada chamando a função solve\_func para avaliar a função nos pontos points[i], e os resultados são armazenados na lista fx.
   7. A função solve\_points é chamada para calcular a soma ponderada dos valores da função nos pontos extremos, utilizando os coeficientes 1, 4 e 2.
   8. A integral é calculada multiplicando-se h pela soma ponderada obtida, dividida por 3 vezes o número de subintervalos.
   9. O resultado da integração é retornado como uma string formatada contendo o valor da integral.

A estratégia de implementação utiliza a fórmula de integração de Simpson de 1/3 para estimar a integral de uma função no intervalo [a, b] dividido em n subintervalos. A função solve\_func permite a avaliação da função em um ponto específico substituindo o valor de x na expressão. A função simpson\_1\_3 calcula os pontos de divisão dos subintervalos, avalia a função nos pontos extremos, calcula a soma ponderada dos valores da função e, em seguida, calcula a integral utilizando a fórmula de Simpson de 1/3.

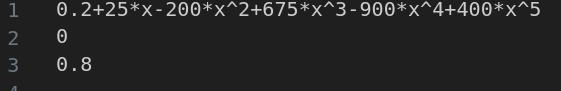
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída devem se chamar “simpson\_13.txt”.

**Entrada**:

Por padrão, cada caso de teste deve conter 3 linhas e uma linha em branco. A primeira linha contém a função f(x), a segunda contém o segundo ponto do intervalo e a terceira contém o primeiro ponto do intervalo, como no exemplo abaixo:



**Saída:**

A saída é composta por uma resposta por 2 linhas de modo bem intuitivo como mostrado abaixo:



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Simpson de 3/8

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

A estratégia de implementação para o método de integração de Simpson de 3/8, com base no código, é a seguinte:

1. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função func\_parser.
2. Integração de Simpson de 1/3:
   1. Explicada anteriormente.
3. Integração de Simpson de 3/8:
   1. A função simpson\_3\_8 calcula a integral de uma função function no intervalo [a, b] utilizando o método de Simpson de 3/8.
   2. O parâmetro n representa o número total de subintervalos em que o intervalo [a, b] será dividido.
   3. O valor de n deve ser múltiplo de 3 para que o método de Simpson de 3/8 seja aplicado corretamente.
   4. A variável drain é calculada como o resto da divisão de n por 3.
   5. O tamanho de cada subintervalo h é calculado como (b - a) / n.
   6. Uma lista points é criada para armazenar os pontos de divisão dos subintervalos. O primeiro ponto é o limite inferior a, e os pontos subsequentes são calculados adicionando-se h / n ao ponto anterior.
   7. A integral parcial é calculada chamando a função simpson\_1\_3 para integrar a função nos subintervalos [a, points[drain]].
   8. O tamanho do último subintervalo é atualizado para h = b - points[drain].
   9. Em seguida, um loop é executado n + 1 vezes para calcular os valores da função nos pontos extremos dos subintervalos.
   10. A função nos pontos extremos é calculada chamando a função solve\_func para avaliar a função.

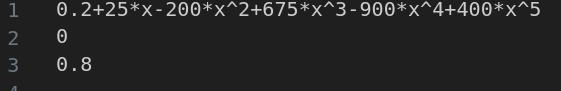
## Estrutura dos Arquivos de Entrada/Saída

Como mencionado anteriormente, o projeto dispõe de uma estrutura de arquivos na qual, todos os arquivos de entrada de dados ficam na pasta “/inputs” e todos os arquivos de saída na pasta “/outputs”.

Para este método, os arquivos de entrada e de saída devem se chamar “simpson\_38.txt”.

**Entrada**:

Por padrão, cada caso de teste deve conter 3 linhas e uma linha em branco. A primeira linha contém a função f(x), a segunda contém o segundo ponto do intervalo e a terceira contém o primeiro ponto do intervalo, como no exemplo abaixo:



**Saída:**

A saída é composta por uma resposta por 2 linhas de modo bem intuitivo como mostrado abaixo:



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

# Extrapolação de Richards

## Estratégia de Implementação:

A implementação dispõe de um método principal chamado “run()”. Este método fica responsável por chamar a função responsável pela leitura do arquivo de entrada, chamar a função responsável para executar o método e por fim, chamar a função responsável por escrever as respostas no arquivo de saída.

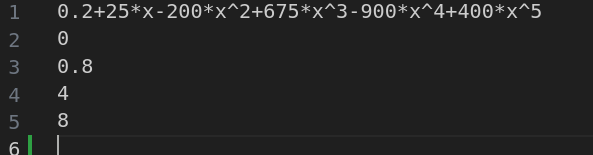
A estratégia de implementação para o método da extrapolação de Richardson, com base no código, é a seguinte:

1. Cálculo da função:
   1. A função solve\_func recebe um valor x e uma string func que representa a função.
   2. Substitui o valor x na string func e retorna o resultado da avaliação da expressão utilizando a função func\_parser.
2. Integração pelo método do trapézio:
   1. Já explicado anteriormente.
3. Extrapolção de Richardson:
   1. A função richards calcula a integral de uma função function no intervalo [a, b] utilizando o método da extrapolação de Richardson.
   2. Os parâmetros n1 e n2 representam o número de subintervalos utilizados nas integrações com o método do trapézio.
   3. O tamanho de cada subintervalo h1 e h2 é calculado como (b - a) / n1 e (b - a) / n2, respectivamente.
   4. A integral i1 é calculada chamando a função trapezoidal\_integration com os parâmetros function, a, b e n1.
   5. A integral i2 é calculada chamando a função trapezoidal\_integration com os parâmetros function, a, b e n2.
   6. A extrapolação de Richardson é realizada utilizando a fórmula: response = i2 + (i2 - i1) / ((h1 / h2) \*\* 2 - 1).
   7. O resultado da integração é retornado como um valor numérico, juntamente com uma mensagem indicando a função integrada.

## Estrutura dos Arquivos de Entrada/Saída

**Entrada**:

Por padrão, cada caso de teste deve conter 5 linhas e uma linha em branco. A primeira linha contém a função f(x), a segunda contém o primeiro ponto do intervalo, a terceira contém o segundo ponto do intervalo, a quarta linha tem o número de vezes que o intervalo deve ser dividido em i1 e a última linha o número de vezes que o intervalo deve ser dividido em i2, como no exemplo abaixo:



**Saída:**

A saída é composta por uma resposta por 2 linhas de modo bem intuitivo como mostrado abaixo:



## Problema teste 1, 2, 3...

## Dificuldades enfrentadas

Quanto à implementação do método, foi tudo tranquilo.

## 

# Considerações Finais

Durante os testes, cada método foi aplicado a diferentes conjuntos de dados e funções de teste. A regressão linear mostrou-se eficaz na modelagem de relacionamentos lineares entre variáveis, fornecendo uma boa aproximação da tendência geral dos dados. No entanto, seu desempenho foi limitado quando os dados apresentaram uma relação não linear.

Para aproximação polinomial, descobrimos que quanto maior o grau do polinômio utilizado, maior a capacidade de ajuste aos dados. No entanto, é importante considerar o equilíbrio entre a complexidade do modelo e a qualidade da aproximação, a fim de evitar overfitting.

A interpolação por polinômios de Lagrange mostrou-se uma técnica poderosa para reconstruir funções entre pontos conhecidos. Ela produziu resultados precisos e suaves, mas exigiu um número significativo de pontos de dados para obter bons resultados.

Já a interpolação por diferenças divididas de Newton apresentou resultados semelhantes à interpolação de Lagrange, porém com um número menor de cálculos necessários. Isso a torna uma opção atraente para interpolação quando a quantidade de dados é grande.

Nos testes de derivada numérica, observamos que a escolha do método depende do equilíbrio entre precisão e estabilidade. A derivada de primeira ordem progressiva forneceu uma boa estimativa em problemas suaves, enquanto a derivada de segunda ordem central foi mais precisa, mas menos estável para funções com maior variação.

Em relação à integração numérica, o método do trapézio simples mostrou-se razoavelmente preciso, porém sua precisão aumentou significativamente quando aplicado à integração múltipla. O método de Simpson de 1/3 forneceu resultados ainda mais precisos, especialmente quando o número de pontos foi aumentado. Já o método de Simpson de 3/8 mostrou-se vantajoso em casos onde o número de pontos não era múltiplo de 3.

Por fim, a extrapolação de Richards, baseada no método do trapézio, permitiu melhorar a precisão da integração ao realizar uma estimativa mais precisa do valor exato. Essa abordagem mostrou-se útil quando a precisão era fundamental e os métodos tradicionais não eram suficientes.