Cálculo Numérico

com Python no Google Colaboratory

Fábio José da Costa Alves
Cínthia Cunha Maradei Pereira

UNIVERSIDADE DO ESTADO DO PARÁ
CENTRO DE CIÊNCIAS SOCIAIS E EDUCAÇÃO

Clay Anderson Nunes Chagas
REITOR

Ilma Pastana Ferreira
VICE-REITOR

Anderson Madson Oliveira Maia
DIRETORA DO CENTRO DE CIÊNCIAS SOCIAIS E EDUCAÇÃO

Fabrício Martins da Costa
CHEFE DO DEPARTAMENTO DE MATEMÁTICA, ESTATÍSTICA E INFORMÁTICA.

Carlos Alberto de Miranda Pinheiro
COORDENADOR DO CURSO DE MATEMÁTICA

Pedro Franco de Sá
PROGRAMA DE PÓS-GRADUAÇÃO EM ENSINO DE MATEMÁTICA

Marta Genu Soares
PROGRAMA DE PÓS-GRADUAÇÃO EM EDUCAÇÂO

GPEMT
GRUPO DE PESQUISA EM ENSINO DA MATEMÁTICA E TECNOLOGIAS

ALVES, Fábio José Costa da; PEREIRA, Cinthia Cunha Maradei. Cálculo Numérico com Python no Google Colaboratory. Grupo de Pesquisa em Ensino da Matemática e Tecnologias, Universidade do Estado do Pará (UEPA), Belém-Pa, 2023.

ISBN: 978-65-84998-59-9

Cálculo Numérico; Python; Google Colaboratory.

SUMÁRIO

APRESENTAÇÃO

Prezado(a) Leitor(a)

O livro Cálculo Numérico com Python no Google Colaboratory foi desenvolvido para o desenvolvimento de habilidades de do pensamento computacional, importante para o futuro desenvolvimento profissional, ao mesmo tempo que se estuda os métodos do cálculo numérico, em uma abordagem prática-experimental.

Na abordagem de cada método, é apresentada a demonstração das expressões e/ou o detalhamento do algoritmo recursivo, cuja exemplificação é apresentada com detalhes, havendo um programa de Python para ser executado na plataforma Google Colaboratory, possibilitando verificar os resultados das atividades, bem como aprofundar conhecimentos na programação em Python.

É importante destacar que o Google Colaboratory é um ambiente de web, pode ser executado tanto no computador quanto no celular, e quem não há necessidade de instalação de nenhum programa ou app. Além disso, os programas Python são executados nos processadores dos computadores da Google, viabilizando a programação em qualquer tipo de computador ou celular, desde que os mesmos possam internet.

Esperamos que com o livro Cálculo Numérico com Python no Google Colaboratory, você consiga adquirir conhecimentos e desenvolver habilidades que possam ajudá-los no desenvolvimento de atividades futuras tanto no âmbito da pós-graduação quanto no dia a dia sua vida profissional em sala de aula.

Os Autores

1. SISTEMA NUMÉRICO E ERROS

Você já observou que no dia a dia estamos cercados de várias situações que representam problemas das mais diversas origens, como física, química, estatística, etc, e que alguns destes problemas são: queda livre de um objeto de cima de um prédio, o crescimento de uma população de uma cidade, o consumo de combustível de um carro, o consumo de energia de nossa casa, entre outros. Quando um problema é representado por uma fórmula ou procedimento matemático, que expressam as características principais deste problema, temos o modelo matemático do problema.

Para que você compreenda melhor a seqüência lógica da solução de um problema, observe o diagrama a baixo.

Observe que para resolvermos um problema, devemos primeiro fazer a modelagem deste problema, isto é, produzir um modelo matemático que descreva todo o comportamento do problema, em seguida devemos buscar a resolução numérica do modelo matemático, que representará a solução do problema.

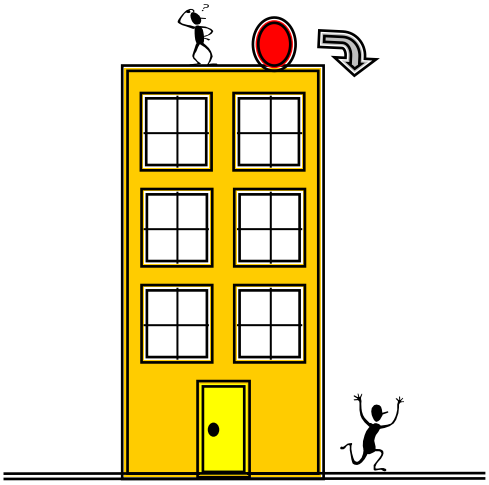
Você sabia que podemos obter a solução de um problema (físico), através de métodos numéricos. Porém, é importante ressaltar, que em certas situações a solução estimada, pelos métodos numéricos, se afasta da verdadeira solução do problema. Isto ocorre devido a presença de fontes de erro que podem ocorrer na fase de modelagem do problema ou na fase resolução do problema.

Para que você possa compreender a fonte de erro no processo de modelagem, observe o exemplo a seguir.

Exemplo: Uma bola cai de cima de um prédio e sua velocidade em cada instante é descrita pela equação horária:

$$s = s_o + v_o t + \frac{a}{2} t^2$$

onde s_o é a altura do prédio, v_o é a sua velocidade inicial e a representa, neste caso, a gravidade.



Se a altura do prédio for de 30 m ($s_o = 30$), a velocidade inicial da bola for zero ($v_o = 0$) e considerando a gravidade igual a 10 m/s2 ($a = 10$). A posição após 3 s após a queda é:

$s = 30 + 0.1 - \frac{10}{2}2^2 \Rightarrow s = 10m$

Você acha que este resultado é confiável?

É bem provável que não, pois no modelo matemático não foram consideradas outras forças, como, por exemplo, a resistência do ar, a velocidade do vento, etc.

Já na fase de resolução, o erro é gerado no momento que se faz os cálculos na calculadora ou computador devido aos processos de arredondamentos. Para exemplificar observe o exemplo a seguir.

Exemplo: Erros na fase de Resolução

Observe que $\sqrt{2} = 1{,}41421356237310$. Ao se resolver esta equação $\frac{x}{10^5} - \sqrt{2} = 0$, cuja solução é $x = 10^5\sqrt{2}$. Observe que a resposta desta equação dependerá do número de dígitos significativos.

se $\sqrt{2} = 1{,}41 \Rightarrow x = 141.000$

se $\sqrt{2} = 1{,}4142 \Rightarrow x = 141.420$

se $\sqrt{2} = 1{,}414213 \Rightarrow x = 141.421{,}30$

MUDANÇA DE BASE

Para você compreender melhor a fonte de erro na fase de resolução, e necessário nos compreendermos como funciona de mudança de base. Você sabia sábia que os

números que usamos no nosso dia a dia estão na base 10. Para uma melhor compreensão observe a decomposição do seguinte número

$8052 = 8 * 10^3 + 0 * 10^2 + 5 * 10^1 + 2 * 10^0$

é assim que se decompõem um número na base dez. Se o numero tiver dígitos atrás da vírgula a decomposição fica da seguinte forma

$8052{,}406 = 8 * 10^3 + 0 * 10^2 + 5 * 10^1 + 2 * 10^0 + 4 * 10^{-1} + 0 * 10^{-2} + 6 * 10^{-3}$

de uma forma compacta podemos dizer que os números na base dez pode ser escritos por:

$$\sum_{i=n}^{m} a_i . 10^i = a_m . 10^m + \ldots + a_2 . 10^2 + a_1 . 10^1 + a_0 . 10^0 + a_{-1} . 10^{-1} a_{-2} . 10^{-2} + \ldots + a_n . 10^n$$

onde: a_i — é 0 ou 1 e n, m — números inteiros, com $n \leq 0$ e $m \geq 0$

Um número na base 2 pode ser escrito como

$$\sum_{i=n}^{m} a_i . 2^i = a_m . 2^m + \ldots + a_2 . 2^2 + a_1 . 2^1 + a_0 . 2^0 + a_{-1} . 2^{-1} + a_{-2} . 2^{-2} + \ldots + a_n . 2^n$$

para compreender melhor observe os exemplos a seguir:

$1011 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0$

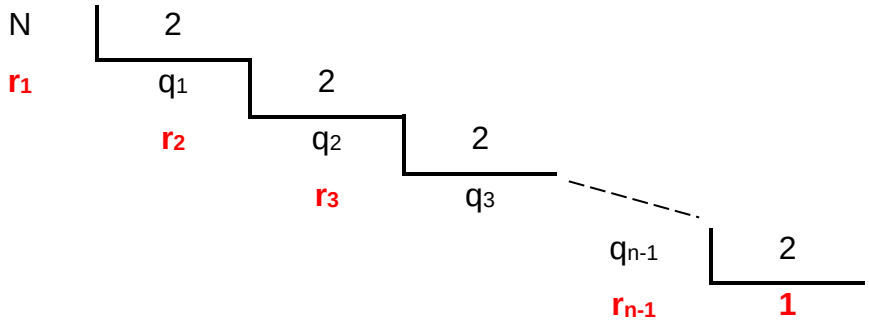
$1011{,}101 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 + 1.2^{-1} + 0.2^{-2} + 1.2^{-3}$

Você sabia, que as calculadoras e os computadores trabalham na base 2, que uma fonte de erro de resolução está nas aproximações que são, as vezes necessárias. Para que você possa entende melhor este problema, vamos, agora, estudar a conversão de um número a base 10 para a base 2. Para isto devemos decompô-lo com fizemos anteriormente, e em seguida efetuar a multiplicação dos dígitos binários pelas potências de base 2 adequadas.

$1011_2 = 1 . 2^3 + 0 . 2^2 + 1 . 2^1 + 1 . 2^0 \quad \Rightarrow \quad 1011_2 = 11_{10}$

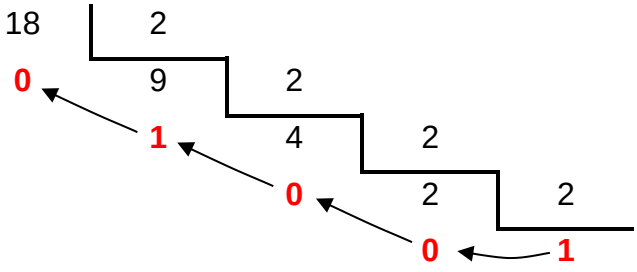
$1011{,}101 = 1 . 2^3 + 0 . 2^2 + 1 . 2^1 + 1 . 2^0 + 1 . 2^{-1} + 0 . 2^{-2} + 1 . 2^{-3} \quad \Rightarrow \quad 1011_2 = 11{,}63_{10}$

Para transformar um número inteiro da base 10 para a base 2, utiliza-se o método de divisões sucessivas, que consiste em dividir o número por 2, a seguir dividi-se por 2 o quociente encontrado e assim o processo é repetido até que o último quociente seja igual a 1. O número binário será, então, formado pela concatenação do último quociente com os restos das divisões lidos em sentido inverso ao que foram obtidos, ou seja,

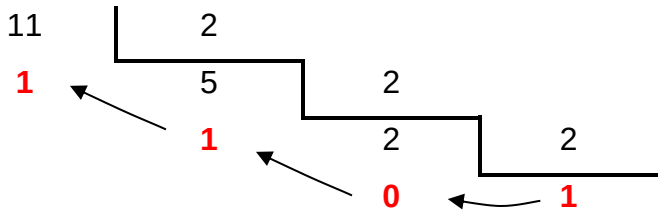


$N_{10} = 1.r_{n-1}.....r_3.r_2.r_1$

Exemplo:



$18_{10} = 10010_2$



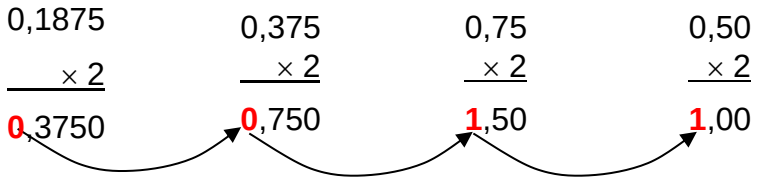
$11_{10} = 1101_2$

Para transformar números fracionários da base 10 para a base 2, utiliza-se o método das multiplicações sucessivas, que consiste em:

1º Passo – multiplicar o numero fracionários por 2;

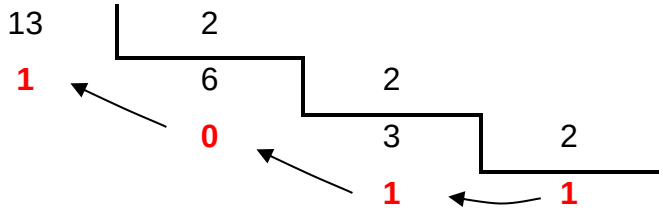
2º Passo – deste resultado, a parte inteira será o primeiro dígito do número na base 2 e a parte fracionária é novamente multiplicada por 2. O processo é repetido até que a parte fracionária do último produto seja igual a zero.

Exemplo: transforme $0{,}1875_{10}$ para a base 2

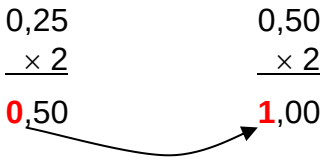


logo $0{,}1875_{10} = 0{,}0011_2$

Exemplo: transforme $13{,}25_{10}$ para a base 2



$13_{10} = 1101_2$



$0{,}25_{10} = 0{,}01_2$, logo $13{,}25_{10} = 1101{,}01_2$

Você sabia que, de maneira geral, o número x em uma base β é representado por:

$$x = \pm\left[\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \ldots + \frac{d_t}{\beta^t}\right].\beta^{exp}$$

onde:

d_i — são os números inteiros contidos no intervalo $0 \leq d_i \leq \beta,\ \ i = 1,2,\ldots,t$

exp — representa o expoente de β e assume valores entre $I \leq exp \leq S$,

I, S — os limites inferior e superior, respectivamente, para a variação do expoente

$\left[\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \ldots + \frac{d_t}{\beta^t}\right]$ — é chamado de mantissa e é a parte do número que representa seus dígitos significativos e t é o número de dígitos significativos do sistema de representação, comumente chamado de precisão da máquina.

Exemplo:

Obs: a mantissa é um número entre 0 e 1.

Sistema decimal

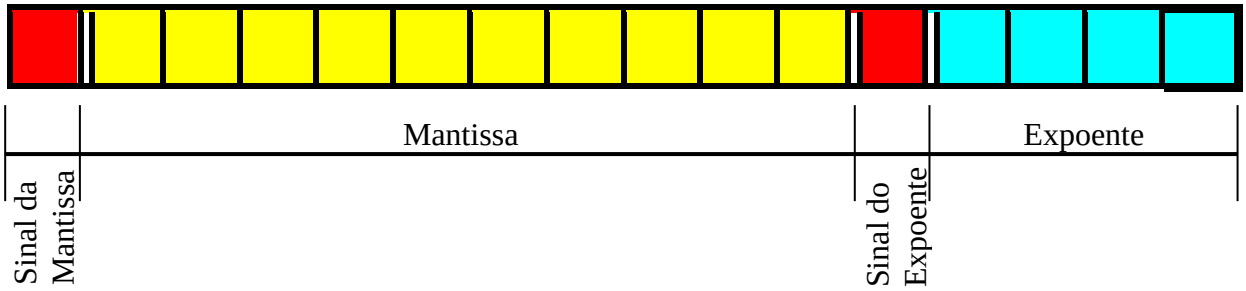
$0{,}357_{10} = \left[\frac{3}{10} + \frac{5}{10^2} + \frac{7}{10^3}\right].10^0$ $\qquad$ $29{,}357_{10} = \left[\frac{2}{10} + \frac{9}{10^2} + \frac{3}{10^3} + \frac{5}{10^4} + \frac{7}{10^5}\right].10^2$

Sistema binário

$11001_2 = \left[\frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{0}{2^4} + \frac{1}{2^5}\right].2^5$

$11001{,}01_2 = \left[\frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{0}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{1}{2^7}\right].2^5$

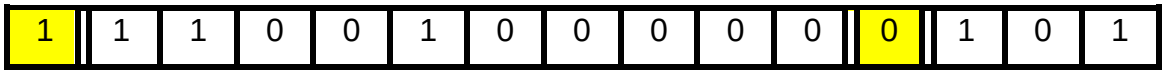
Saiba que cada dígito do computador é chamado de bit. Apresentaremos abaixo uma maquina fictícia de 10 bits para a mantissa, 4 bits para o expoente e 1 bit para o sinal da mantissa e outro bit para o sinal do expoente.



Para você entender melhor, faremos um exemplo numérico.

Exemplo: Numa máquina de calcular cujo sistema de representação utilizado tenha $\beta = 2$, $t = 10$, $I = -15$ e $S = 15$, o número 25 na base decimal é representado por

$$-25_{10} = -11001_2 = -0{,}11001.2^5 = -0{,}11001.2^{101}$$



Observe que utilizamos bit = 0 para positivo e bit = 1 para negativo.

Um parâmetro muito utilizado para avaliar a precisão de um determinado sistema de representação é o número de casas decimais exatas da mantissa e que este valor é

dado pelo valor decimal do último bit da mantissa, ou seja, o bit de maior significado, logo:

$$PRECISÃO \leq \frac{1}{\beta^t}$$

Apresentaremos a seguir, a titulo de curiosidade, os sistemas de representação de algumas máquinas.

Máquinas	β	t	I	S
Burroughs 5500	8	13	- 51	77
Burroughs 6700	8	13	- 63	63
Hewlett-Packard 45	10	10	- 98	100
Texas SR-5X	10	12	- 98	100
PDP-11	2	24	- 128	127
IBM/360	16	6	- 64	63
IBM/370	16	14	- 64	63
Quartzil QI 800	2	24	- 127	127

ATIVIDADE

(01) Os números a seguir estão na base 2, escreva-os na base 10.

(a) $11011_2 =$ (b) $111100_2 =$ (c) $100111_2 =$

(02) Os números a seguir estão na base 10, escreva-os na base 2.

(a) $15_{10} =$ (b) $12_{10} =$ (c) $36_{10} =$

(03) Considere uma máquina de calcular cujo sistema de representação utilizado tenha $\beta = 2, t = 10, I = -15$ e $S = 15$. Represente nesta máquina os números:

(a) 35_{10} (b) $8,2_{10}$ (c) -24_{10}

2. GOOGLE COLABORATORY

Uma pesquisa em 2019 que investigou milhares de listas de empregos para cientistas de dados, verificou-se que entre as 15 tecnologias mais procuradas, Python ficou em primeiro lugar e foi de longe a palavra-chave mais frequente nas listas. Além disso, muitas empresas de tecnologia famosas como Google, Instagram e Netflix fazem uso dessa linguagem, contribuindo para a popularização da mesma.

Python possui uma sintaxe simples se comparada com outras linguagens de programação, com isso a curva de aprendizado é mais rápida do que se comparada com outras opções. O Python é uma linguagem de programação amplamente usada em aplicações da Web, desenvolvimento de software, ciência de dados e machine learning (ML). Os desenvolvedores usam o Python porque é eficiente e fácil de aprender e pode ser executado em muitas plataformas diferentes.

A plataforma Google Colaboratory, ou mais conhecida como “Colab”, abreviadamente, é um produto da Google Research, que permite que qualquer pessoa escreva e execute código Python por meio do navegador e é especialmente adequado para aprendizado, análise de dados e aplicações na educação.

O primeiro passo para usar o Google Colab, para programar em Python, é fazer o login em uma conta do Google, depois acesse o endereço https://colab.research.google.com/ e acessará o Google Colab diretamente pelo navegador, sem precisar instalar nada em seu computador. Ao abrir o Google Colab, você será recebido com uma interface limpa e amigável.

Os programas Python na plataforma no Google Colaboraty são escutados em um arquivo denominado de notebooks do Colab que executam código dos servidores em nuvem do Google, e isso significa que pode tirar proveito da potência de hardware do Google, como GPUs e TPUs, independentemente da potência da sua máquina, e só precisa de um navegador para isso.

3. RESOLUÇÃO NUMÉRICA DE EQUAÇÕES NÃO LINEARES

Você sabia, que nas mais diversas áreas das ciências ocorrem situações que envolvem a resolução de uma equação do tipo $f(x) = 0$ que não possui solução algébrica. Está é a razão porque devemos desenvolver métodos numéricos para resoluções as equações do tipo $f(x) = 0$, podendo ser equações lineares ou não lineares.

Iniciaremos uma nova etapa estudando os métodos para isolar e calcular as raízes de uma equação real. Tais métodos numéricos são usados na busca das raízes das equações, ou os zeros reais de $f(x)$. Embora estes métodos não forneçam raízes exatas, eles podem calcular as raízes com a exatidão que o problema requeira.

Em geral, os métodos, utilizados apresentam duas fases distintas:

Fase I – Localização ou Isolamento das Raízes

Está fase consiste em obter um intervalo que contém a raiz da função $f(x) = 0$, e em seguida iremos para a segunda fase.

Fase II – Refinamento

Nesta fase definimos a precisão que desejamos da nossa resposta e escolhemos as aproximações iniciais dentro do intervalo encontrado na Fase I. Em seguida melhoramos, sucessivamente, a aproximação da raiz da função $f(x) = 0$, até se obter uma aproximação para a raiz dentro de uma precisão pré-fixada.

ISOLAMENTO DE RAÍZES

É importante, que você saiba que os métodos numéricos utilizados para calcular raízes da equação $f(x) = 0$, só calculam uma raiz de cada vez!

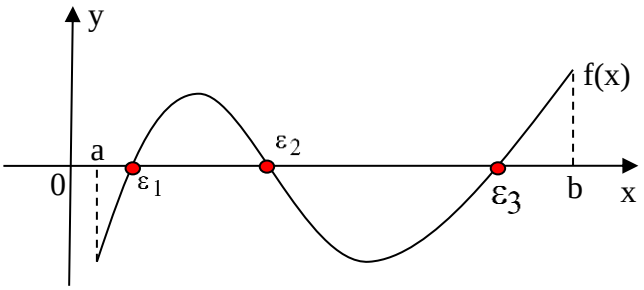
Esta é a razão porque devemos determinar um intervalo para cada raiz que desejamos calcular. Para entendermos melhor como isolar uma raiz de uma equação, nós devemos observar o teorema a seguir.

Teorema

"Se uma função contínua $f(x)$ assume valores de sinais oposto nos pontos extremos do intervalo [a , b] , isto é, $f(a).f(b) < 0$, então o intervalo conterá, no mínimo, uma raiz da equação $f(x) = 0$, em outras palavras haverá no mínimo um número ε, pertencente ao intervalo aberto (a, b), $\varepsilon \in (a, b)$, tal que, $f(\varepsilon) = 0$"

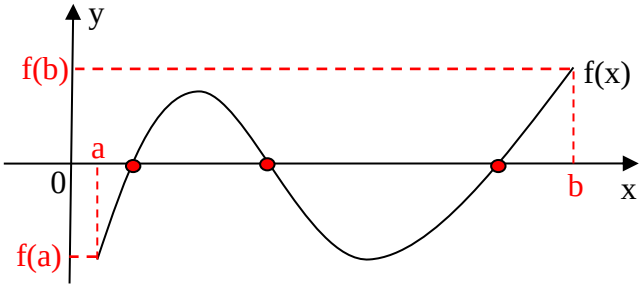
Exemplo:

Neste exemplo apresentamos uma função $f(x)$ que possui dentro do intervalo $[a, b]$ três raízes: ε_1, ε_2 e ε_3. Isto é, são três valores de x, para os quais a função $f(x)$ tem imagem igual a zero, isto é: $f(\varepsilon_1) = 0, f(\varepsilon_2) = 0$ e $f(\varepsilon_3) = 0$.

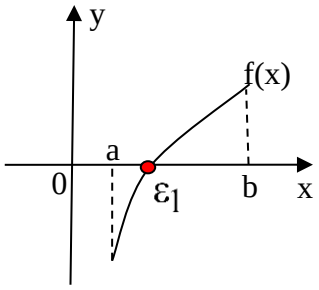


Se a função possui imagem zero nos pontos ε_1, ε_2 e ε_3, o gráfico da função $f(x)$, nestes pontos, intercepta o eixo dos x.

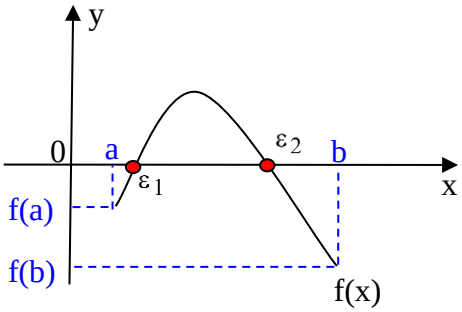
Observe no exemplo que $f(a) < 0$ e $f(b) > 0$, logo o produto $f(a).f(b) < 0$



Observe que toda vez que dentro de um intervalo $[a, b]$, tivermos $f(a).f(b) < 0$, significa que neste intervalo temos pelo menos uma raiz da função $f(x)$, como vemos na figura a seguir.



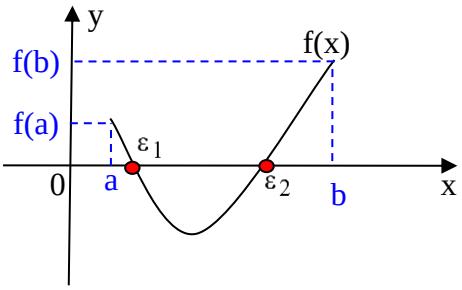
Observe, na figura a seguir, que quando uma função possui um número par de raízes dentro do intervalo $[a, b]$, temos $f(a).f(b) > 0$



$f(a) < 0$

$f(b) < 0$

logo $f(a).f(b) > 0$

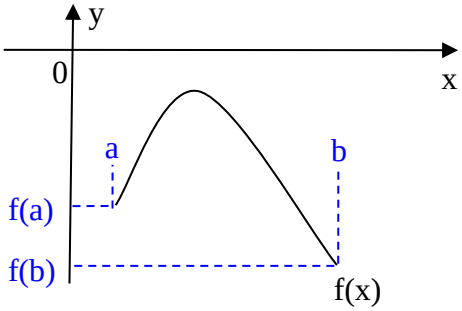


$f(a) > 0$

$f(b) > 0$

logo $f(a).f(b) > 0$

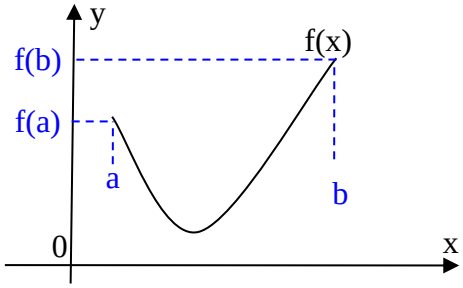
Observe, na figura a seguir, que quando uma função não possui raízes dentro dointervalo $[a, b]$, temos $f(a).f(b) > 0$



$f(a) < 0$

$f(b) < 0$

logo $f(a).f(b) > 0$



$f(a) > 0$

$f(b) > 0$

logo $f(a).f(b) > 0$

O número de raizes de uma função $f(x)$, dentro do intervalo $[a, b]$, que observamos nos exemplos anteriores, é formalmente expressõ pelo teorema que anunciaremos a seguir.

TEOREMA DE BOLZANO

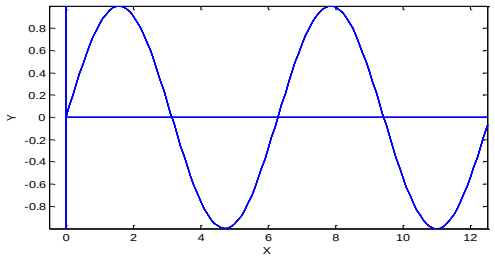
Seja $P(x) = 0$ uma equação algébrica com coeficientes reais e $x \in (a, b)$.

- Se $P(a).P(b) < 0$, então existem um número ímpar de raízes reais no intervalo (a, b).
- Se $P(a).P(b) > 0$, então existem um número par de raízes reais no intervalo (a, b) ou não existem raízes reais no intervalo (a, b).

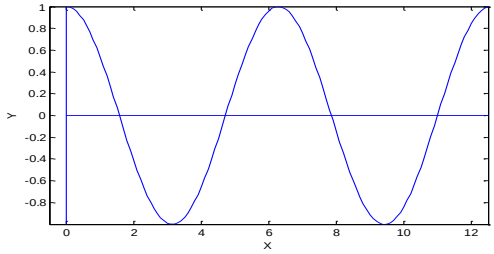
EQUAÇÕES TRANSCENDENTES

Saiba que a determinação do número de raízes de funções transcendentes é quase impossível, pois algumas equações podem ter um número infinito de raízes. Como exemplo temos as funções:

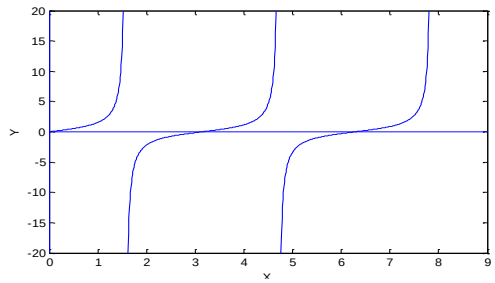
Função Seno



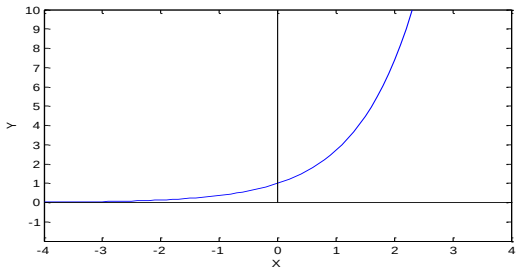
Função Cosseno



Função Tangente



Função Exponencial



O método mais simples de se achar um intervalo que contenha só uma raiz de uma função, ou seja, isolar uma raiz, é o método gráfico que abordaremos a seguir.

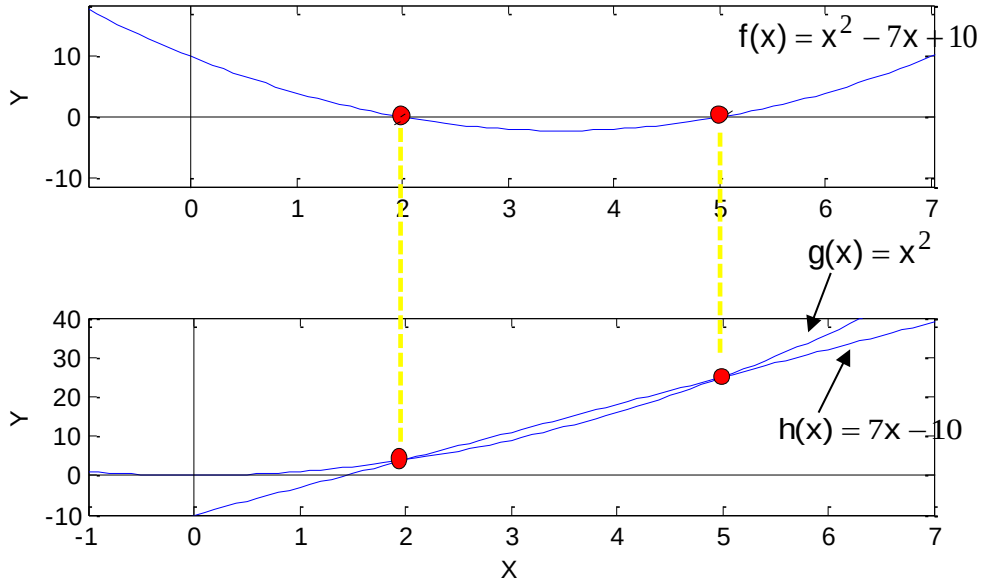
MÉTODO GRÁFICO

Lembre-se que uma raiz de uma equação $f(x) = 0$ é um ponto onde a função $f(x)$ toca o eixo dos x. Observe a função $f(x) = x^2 - 6x + 5$ cujo gráfico está na figura a seguir.

Saiba que uma outra forma de identificarmos as raízes da equação é substituir $f(x) = g(x) - h(x)$, onde $g(x) - h(x) = 0$. As raízes de $f(x) = 0$ corresponderam a interseção das funções $g(x)$ e $h(x)$.

Para você entender melhor, observe o exemplo a seguir, onde utilizamos a função $f(x) = x^2 - 7x + 10$ que possui raízes 2 e 5.

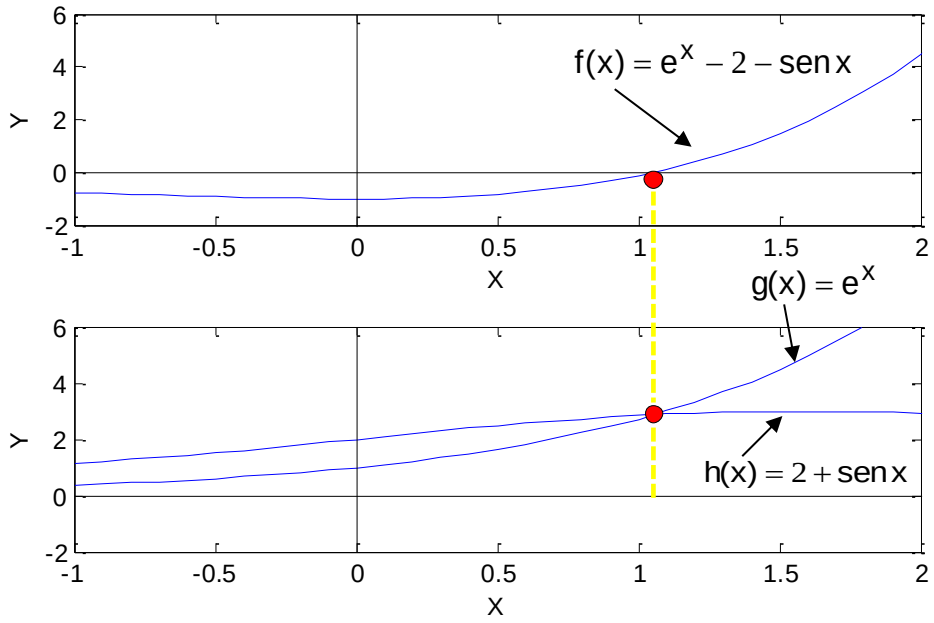
Se fizermos $f(x) = g(x) - h(x)$, onde $g(x) = x^2$ e $h(x) = 7x - 10$ temos a interseção de $g(x)$ com $h(x)$ acontece em 2 e 5.



Observe no próximo exemplo que o método gráfico também funciona com funções mais complexas cujas raízes não são simples de se determinar.

Exemplo:

A aplicação do método utilizaremos a função $f(x) = e^x - 2 - senx$ que possui raízes 2 e 5. Fazendo $g(x) = e^x$ e $h(x) = +2 + senx$, observe que é muito mais fácil fazer o gráfico de $g(x)$ e $h(x)$ do que a fazer o gráfico da função $f(x)$.



Analisando o gráfico podemos afirmar que a nossa raiz esta próxima de 1, então este será nosso valor inicial para os nossos métodos numéricos.

ATIVIDADE

(01) Dada a função $f(x) = 0.2x^2 + senx$, separe está em duas funções e aproxime pelo menos uma de suas raízes pelo método gráfico.

(02) Dada a função $f(x) = x^2 - 4x$, separe está em duas funções e aproxime pelo menos uma de suas raízes pelo método gráfico.

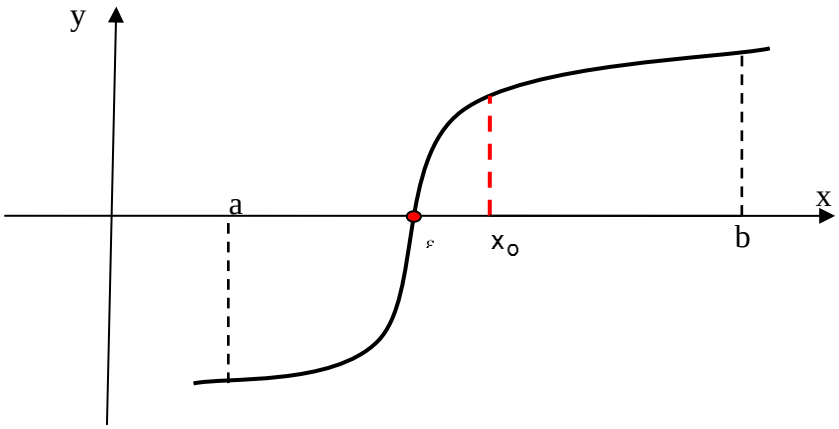
(03) Dada a função $f(x) = x^2 - cos\, x$, separe está em duas funções e aproxime pelo menos uma de suas raízes pelo método gráfico.

(04) Dada a função $f(x) = x^3 + senx$, separe está em duas funções e aproxime pelo menos uma de suas raízes pelo método gráfico.

3.1. MÉTODO DA BISSEÇÃO

Para utilizarmos este método devemos primeiro isolar a raiz dentro de um intervalo $[a, b]$, isto é, devemos utilizar o método gráfico para aproximar visualmente a raiz para em seguida isolá-la pelo intervalo (a, b), onde esta raiz pertença a este intervalo.

Para utilizarmos o método da bisseção é necessário que a função $f(x)$ seja uma continua no intervalo $[a, b]$ e que $f(a).f(b) < 0$. No método da bisseção devemos dividir o intervalo $[a, b]$ ao meio, obtendo assim x_o, com isto temos agora dois intervalos $[a, x_o]$ e $[x_o, b]$

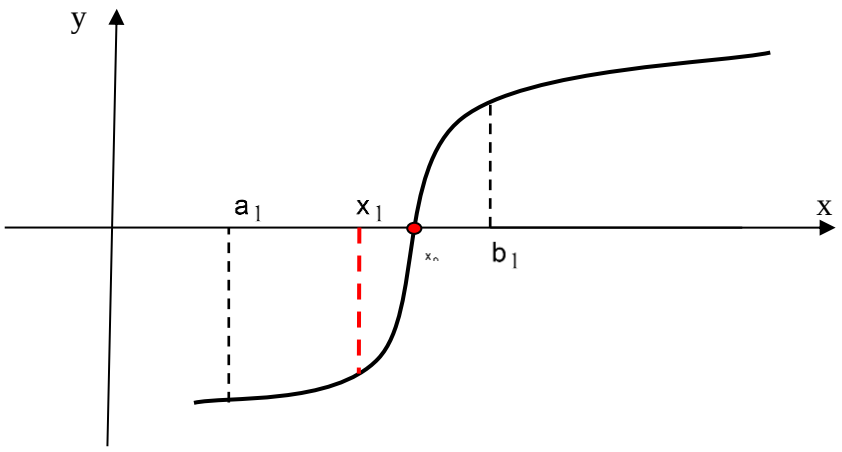


Se $f(x_o) = 0$, então, $\varepsilon = x_o$; Caso contrário, a raiz estará no subintervalo onde a função tem sinais oposto nos pontos extremos, ou seja, se

$f(a).f(x_o) < 0$ implica que a raiz está no intervalo $[a, x_o]$.

$f(x_o).f(b) < 0$ implica que a raiz está no intervalo $[x_o, b]$.

A partir daí construiremos um novo intervalo $[a_1, b_1]$



O novo intervalo $[a_1, b_1]$ que contém ε é dividido ao meio e obtém-se x_1 onde se

$f(a_1).f(x_1) < 0$ implica que a raiz está no intervalo $[a_1, x_1]$.

$f(x_1).f(b_1) < 0$ implica que a raiz está no intervalo $[x_1, b_1]$.

O processo se repete até que se obtenha uma aproximação para a raiz exata ε, com a tolerância $\in$ desejada. Tolerância $(\in)$ é um valor que o calculista define, que define a proximidade que deve ter do valor estimado do valor exato. A partir da tolerância, definimos o critério de parada, onde se para de refinar a solução e se aceita o valor aproximado calculado. A tolerância $\in$, é muitas vezes avaliada por um dos três critérios abaixo:

$$|f(x_n)| \leq E$$

$$|x_n - x_{n-1}| \leq E$$

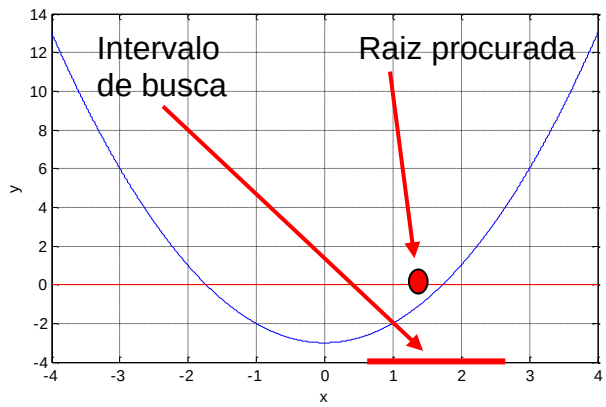
$$\frac{|x_n - x_{n-1}|}{|x_n|} \leq E$$

Para você compreender melhor a aplicação do método da bisseção, observe os próximos exemplos numéricos, onde determinaremos as raízes das funções determinadas.

Exemplo:

(01) Calcular a raiz da equação $f(x) = x^2 - 3$ com $E \leq 0{,}01$.

Solução: Primeiro devemos determinar um intervalo onde está a raiz que desejamos calcular, para isto devemos fazer uma no seu gráfico.



A raiz procurada está próxima de 2 e está dentro do intervalo [1 , 3].

Logo

N	a_n	b_n	x_n	$f(x_n)$	E
0	1.0000	3.0000	2.0000	1.0000	
1	1.0000	2.0000	1.5000	-0.7500	0.5000
2	1.5000	2.0000	1.7500	0.0625	0.2500
3	1.5000	1.7500	1.6250	-0.3594	0.1250
4	1.6250	1.7500	1.6875	-0.1523	0.0625
5	1.6875	1.7500	1.7188	-0.0459	0.0313
6	1.7188	1.7500	1.7344	0.0081	0.0156
7	1.7266	1.7344	1.7266	-0.0190	0.0078

onde N $\Rightarrow$ número da interação

a_n $\Rightarrow$ extremo inferior do intervalo $[a_n , b_n]$.

b_n $\Rightarrow$ extremo superior do intervalo $[a_n , b_n]$.

x_n $\Rightarrow$ ponto médio do intervalo $[a_n , b_n]$.

$f(x_n)$ $\Rightarrow$ valor da função em x_n.

E $\Rightarrow$ erro calculado pela expressão $|x_n - x_{n-1}|$

Construção da tabela

1ª linha: Na iteração inicial (N = 0) temos $[a_o b_o] = [13]$ sendo o ponto médio $x_o = 2$.

2ª linha: (N = 1) Como $f(a_o).f(x_o) < 0$, substituímos $b_1 = x_o$, logo $[a_1 b_1] = [12]$ sendo o ponto médio $x_1 = 1{,}5$.

3ª linha: (N = 2) Como $f(x_1).f(b_1) < 0$, substituímos $a_2 = x_1$, logo $[a_2 b_2] = [1{,}52]$ sendo o ponto médio $x_2 = 1{,}75$.

.....

8ª linha: (N = 7) Como $f(a_6).f(x_6) < 0$, substituímos $a_7 = x_6$, logo $[a_7 b_7] = [1.7188 \quad 1.7344]$ sendo o ponto médio $x_7 = 1.7266$.

Como o erro é menor que tolerância $(0.0078 < E)$ então a aproximação final é $x = 1{,}7266$.

PROGRAMA EM PYTHON

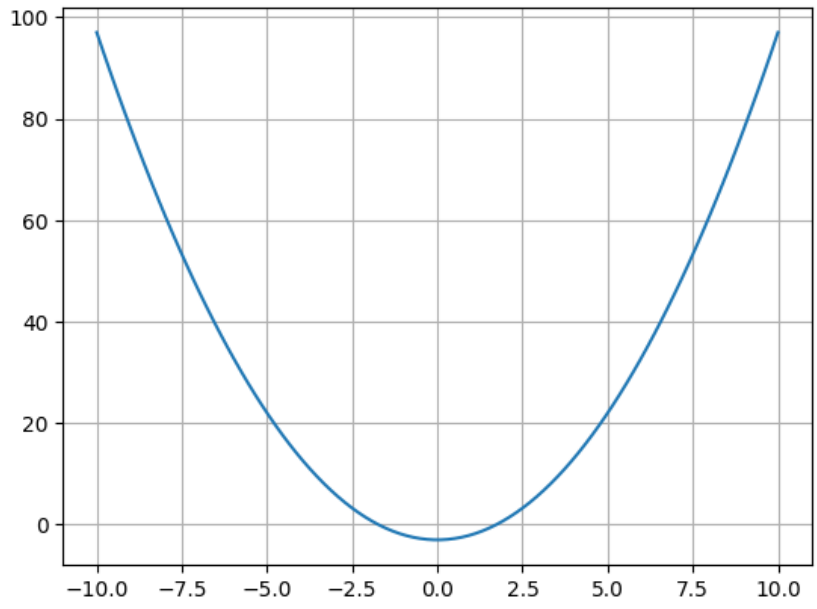
```
# Método da Bisseção
# Entrada
a = 1                # intervalo [a , b]
b = 3
tolerancia = 0.01    # tolerância
nloop = 50           # número máximo de loop

def f(x):
  return x**2 - 3


import math

print("Método da Bisseção")
print("
n      a        b       x       f(a)     f(x)     f(b)    f(a)*f(x)  err
o")

n = 1
fa = f(a)
fb = f(b)
xm2 = (a + b)/2
fxm = f(xm2)
v = fa*fxm
erro = 10
```

```
print("%2d"%n, "%8.4f"%a, "%8.4f"%b, "%8.4f"%xm2, "%8.4f"%fa,
"%8.4f"%fb, "%8.4f"%fxm, "%8.4f"%v, "%8.4f"%erro)

if v < 0: b = xm2
if v > 0: a = xm2
if v == 0: print("o valor da raiz é %4.4f" %xm2)

while(erro > tolerancia):
    xm1 = xm2
    n = n + 1.
    xm2 = (a + b)/2
    fxm = f(xm2)
    erro = math.fabs(xm1 - xm2)
    v = fa*fxm
    if v < 0: b = xm2
    if v > 0: a = xm2
    print("%2d"%n, "%8.4f"%a, "%8.4f"%b, "%8.4f"%xm2, "%8.4f"%fa,
"%8.4f"%fb, "%8.4f"%fxm, "%8.4f"%v, "%8.4f"%erro)
    if(n == nloop):
        break
```

```
print("\nA raiz aproximada é %4.4f" %xm2)
print('Loop para com no máximo 50 interações')

import matplotlib.pyplot as plt
import numpy as np
xi = np.linspace(-10, 10, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Método da Bisseção
 n       a         b         x        f(a)      f(x)      f(b)    f(a)*f(x)  erro
 1    1.0000    3.0000    2.0000   -2.0000    6.0000    1.0000   -2.0000  10.0000
 2    1.5000    2.0000    1.5000   -2.0000    6.0000   -0.7500    1.5000   0.5000
 3    1.5000    1.7500    1.7500   -2.0000    6.0000    0.0625   -0.1250   0.2500
 4    1.6250    1.7500    1.6250   -2.0000    6.0000   -0.3594    0.7188   0.1250
 5    1.6875    1.7500    1.6875   -2.0000    6.0000   -0.1523    0.3047   0.0625
 6    1.7188    1.7500    1.7188   -2.0000    6.0000   -0.0459    0.0918   0.0312
 7    1.7188    1.7344    1.7344   -2.0000    6.0000    0.0081   -0.0161   0.0156
 8    1.7266    1.7344    1.7266   -2.0000    6.0000   -0.0190    0.0380   0.0078

A raiz aproximada é 1.7266
Loop para com no máximo 50 interações
```



ATIVIDADE

(01) Calcular a raiz da equação $f(x) = 2x^2 - 10$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 2.2422

(02) Calcular a raiz da equação $f(x) = 2x^3 - 5$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [0 , 3]) Resposta: 1.3535

(03) Calcular a raiz da equação $f(x) = x^2 - 16 + senx$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [3 , 5]) Resposta: 4.1016

(04) Calcular a raiz da equação $f(x) = x^2 - 5senx$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 2.0000

(05) Calcular a raiz da equação $f(x) = -x^2 + 7$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [2 , 4])

(06) Calcular a raiz da equação $f(x) = x^2 - 4 + cos\, x$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [0 , 2])

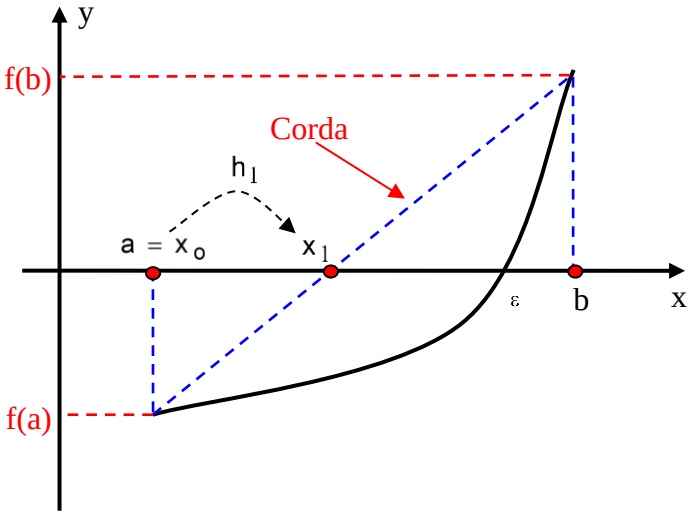
(07) Calcular a raiz da equação $f(x) = x^3 - 12$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3])

3.2. MÉTODO DAS CORDAS

Este será o segundo método numérico para o cálculo de raízes que iremos estudar. Para utilizarmos este método devemos primeiro isolar a raiz dentro de um intervalo $[a, b]$, isto é, devemos, novamente, utilizar o método gráfico para aproximar visualmente a raiz para em seguida isolá-la pelo intervalo $[a\,, b]$, sendo que a raiz pertença ao intervalo $(a\,, b)$.

Para utilizarmos o método das cordas é necessários que a função $f(x)$ seja uma continua no intervalo $[a\,, b]$ e que derivada segunda com sinal constante, sendo $f(a).f(b) < 0$ e que somente um número $\varepsilon \in [a\,, b]$ tal que $f(\varepsilon) = 0$

No método das cordas, ao invés de se dividir o intervalo $[a\,, b]$ ao meio, ele é dividido em partes proporcionais à razão $-f(a)/f(b)$, ou seja

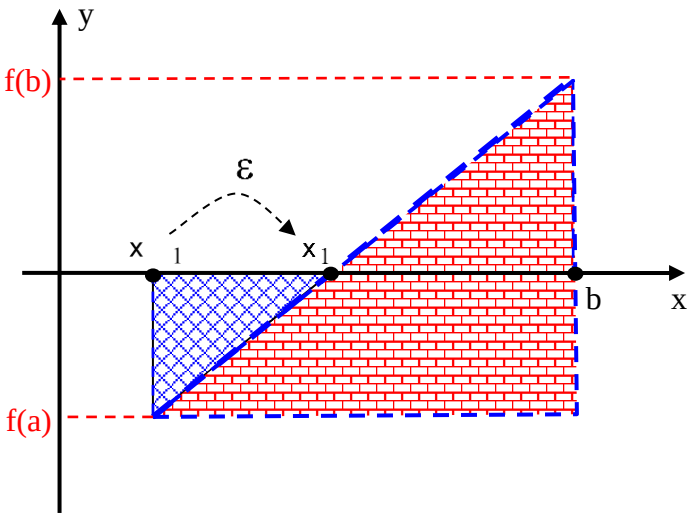


A existência da corda da origem a dois triângulos semelhantes, que permitem estabelecer a seguinte relação:

$$\frac{h_1}{-f(a)} = \frac{b-a}{f(b)-f(a)}$$

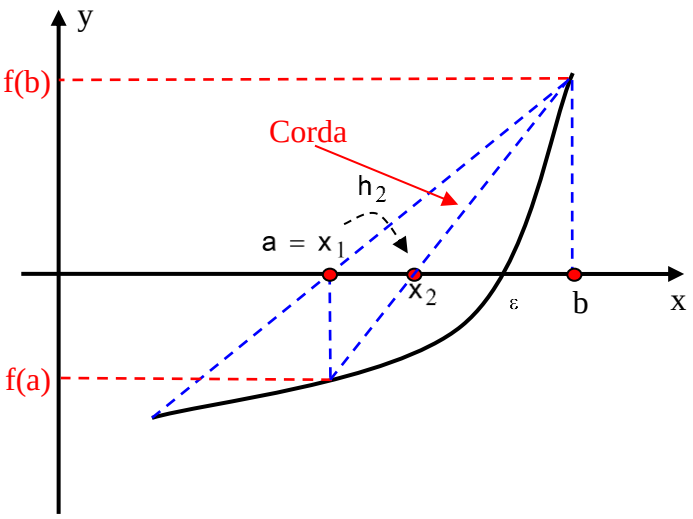
esta relação nos conduz a um valor aproximado da raiz

$$x_1 = a + h_1$$



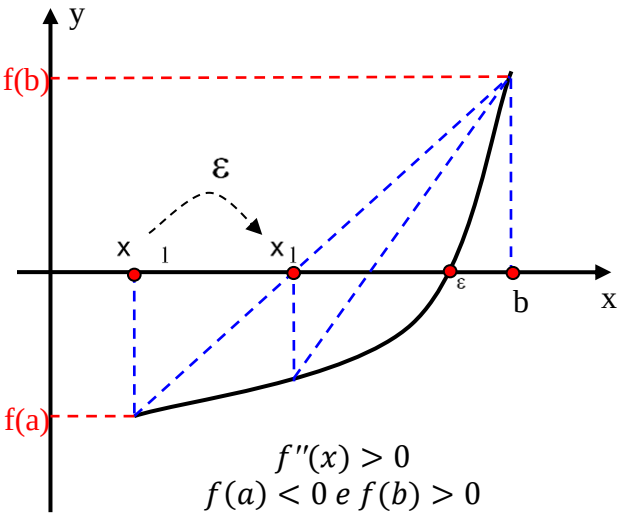
$$x_1 = a - \frac{f(a)}{f(b)-f(a)}(b-a)$$

Ao se aplicar este procedimento ao novo intervalo que contém ε, como mostra a figura a seguir, ($[a\,,x_1]$ *ou* $[x_1\,,b]$), obtém-se uma nova aproximação x_2 da raiz pela aproximação apresentada acima

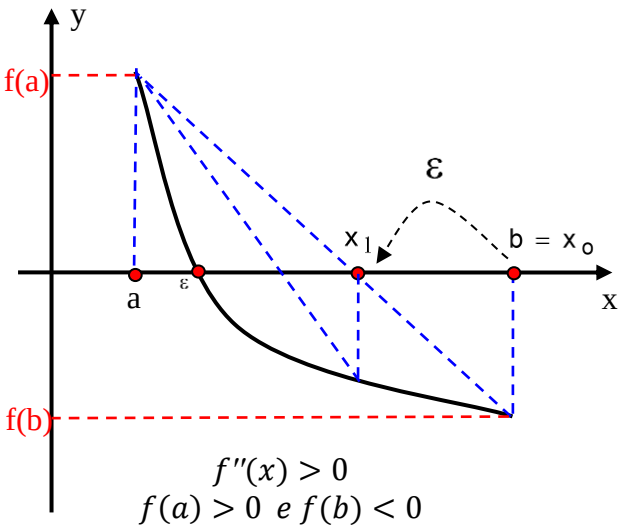


No método das cordas substituímos a curva $y = f(x)$ por uma corda que passa pelos pontos $A(a\,, f(a))$ e $B(b\,, f(b))$

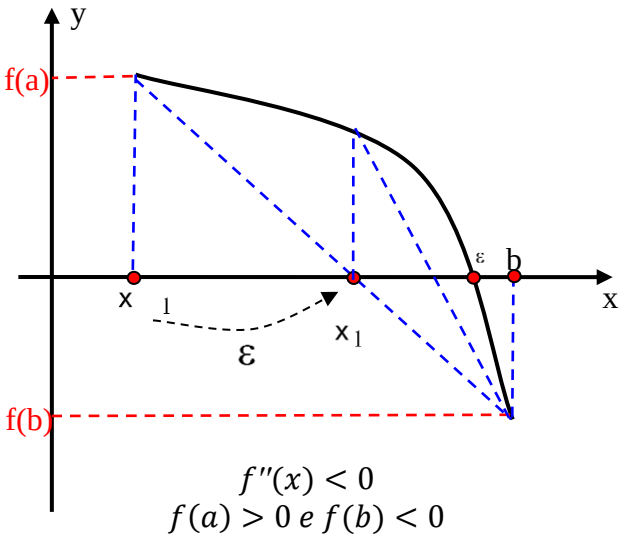
Observe, nas figuras a seguir, como no método das cordas é escolhido o extremo do intervalo $[a\,, b]$ que deve ser igual ao valor x_o.



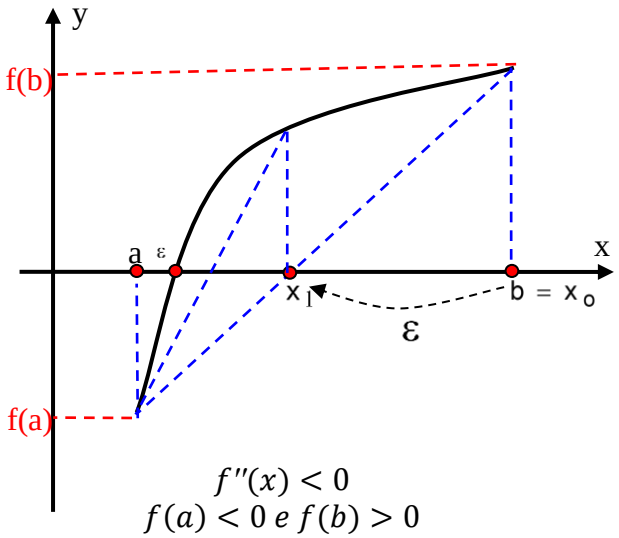
$f''(x) > 0$
$f(a) < 0\ e\ f(b) > 0$
$c = b$



$f''(x) > 0$
$f(a) > 0\ \ e\ f(b) < 0$
$c = a$



$f''(x) < 0$
$f(a) > 0\ e\ f(b) < 0$
$c = b$



$f''(x) < 0$
$f(a) < 0\ e\ f(b) > 0$
$c = a$

A fórmula de recorrência para a aproximação da raiz enésima é

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(c)}(x_n - c), \quad \text{onde } n = 0,1,2,\ldots,$$

onde o ponto fixado c (ou "a" ou "b") é aquele no qual o sinal da função $f(x)$ coincide com o sinal da segunda derivada $f''(x)$, ou seja $f''(c).f(c) > 0$.

$$\frac{|x_n - x_{n-1}|}{|x_n|} \leq E$$

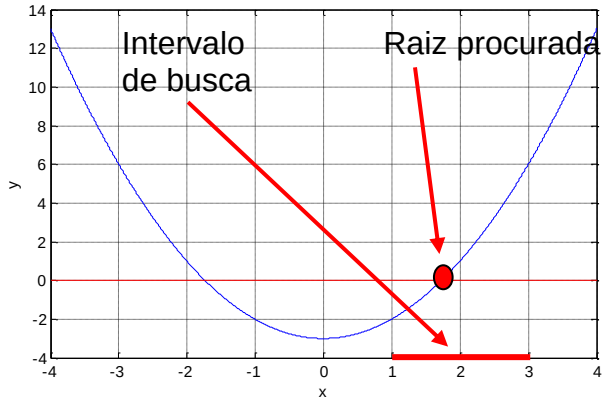
Para você compreender melhor a aplicação do método das cordas, observe os próximos exemplos numéricos, onde determinaremos as raízes das funções.

Exemplo:

(01) Calcular a raiz da equação $f(x) = x^2 - 3$ com $E \leq 0{,}01$.

Solução

Primeiro devemos determinar um intervalo onde esta a raiz que desejamos calcular, para isto devemos fazer uma no seu gráfico.



A raiz procurada está próxima de 2 e está dentro do intervalo [1 , 3]. Logo

N	a_n	b_n	x_n	$f(x_n)$	E
0	1.0000	3.0000	3.0000	6.0000	1.5000
1	1.0000	1.5000	1.5000	-0.7500	0.3000
2	1.0000	1.8000	1.8000	0.2400	0.0857
3	1.0000	1.7143	1.7143	-0.0612	0.0226
4	1.0000	1.7368	1.7368	0.0166	0.0061

onde

N $\Rightarrow$ número da interação

a_n $\Rightarrow$ extremo inferior do intervalo $[a_n , b_n]$.

b_n $\Rightarrow$ extremo superior do intervalo $[a_n , b_n]$.

x_n $\Rightarrow$ ponto médio do intervalo $[a_n , b_n]$.

$f(x_n)$ $\Rightarrow$ valor da função em x_n.

E $\Rightarrow$ erro calculado pela expressão $|x_n - x_{n-1}|$

Construção da tabela

Como $f''(x) = 2 \Rightarrow f''(3) = 2 > 0$ e $f(3) = 3^2 - 3 = 6 > 0$

logo $f''(3).f(3) > 0$ de onde temos que $c = a = 1$ usando a fórmula de recorrência $x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(c)}(x_n - c)$ temos que $x_0 = b = 3$

$x_1 = x_0 - \frac{f(x_0)}{f(x_0)-f(1)}(x_0 - 1) = 1.5000 \Rightarrow [a , b] = [1.0 \quad 1.50]$

$x_2 = x_1 - \frac{f(x_1)}{f(x_1)-f(1)}(x_1 - 1) = 1.8000 \Rightarrow [a , b] = [1.0 \ , \ 1.80]$

$x_3 = x_2 - \frac{f(x_2)}{f(x_2)-f(1)}(x_2 - 1) = 1.7143 \Rightarrow [a , b] = [1.0 \quad 1.7143]$

$x_4 = x_3 - \frac{f(x_3)}{f(x_3)-f(1)}(x_3 - 1) = 1.7368 \Rightarrow [a , b] = [1.0 \quad 1.7368]$

Como o erro é menor que tolerância, então a aproximação é $x = 1{,}7368$.

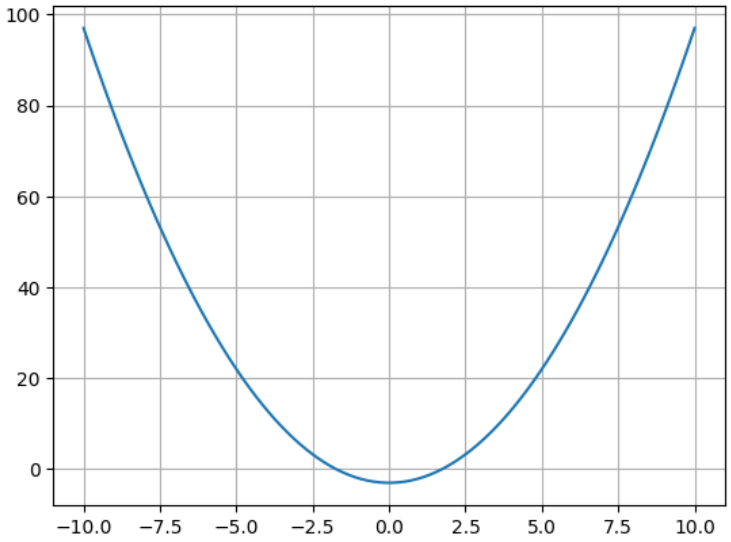
PROGRAMA EM PYTHON

```
# Método das Cordas
# Entrada
a = 1                # intervalo [a , b]
b = 3
tolerancia = 0.01    # tolerância
nloop = 50           # número máximo de loop

def f(x):
  return x**2 - 3

def der1(x):
# Derivada de primeira ordem
  dxd1 = 0.0001
  return ( f(x + dxd1) - f(x) ) / dxd1

def der2(x):
# Derivada de segunda ordem
  dxd2 = 0.0001
  d11 = ( f(x) - f(x - dxd2) ) / dxd2
  d12 = ( f(x + dxd2) - f(x) ) / dxd2
  return ( d12 - d11 ) / dxd2

import math

print("Método das Cordas")
print(" n      a        b       xn      f(xn)     erro")

vfa = 0
vfb = 0
vder2a = 0
vder2b = 0

if (f(a) >= 0):
  vfa = 1

if (f(b) >= 0):
  vfa = 1

if (der2(a) >= 0):
  vder2a = 1

if (der2(b) >= 0):
  vder2b = 1

if (vder2a == vfa):
    xo = a
    c = b
```

```
if (vder2b == vfb):
    xo = b
    c = a

# Variáveis auxiliares
para = 0
xk = 0
h = 0

erro = 10

if (vder2a == vfa):
  print("%2d"%h, "%8.4f"%xo, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
if (vder2b == vfb):
  print("%2d"%h, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)

while(para == 0):
    xk = xo - (f(xo)/(f(xo)-f(c)))*(xo - c);
    erro = abs(xk - xo)
    xo = xk
    h = h + 1
    if (vder2a == vfa):
      print("%2d"%h, "%8.4f"%xo, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
    if (vder2b == vfb):
      print("%2d"%h, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
    if ( erro < tolerancia):
      if (h > 1):
        para = 1

print("\nA raiz aproximada é %4.4f \n" %xo)

import matplotlib.pyplot as plt
import numpy as np
xi = np.linspace(-10, 10, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Método das Cordas
 n       a          b        xn       f(xn)     erro
 0    1.0000    3.0000    1.0000   -2.0000  10.0000
 1    1.5000    3.0000    1.5000   -0.7500   0.5000
 2    1.6667    3.0000    1.6667   -0.2222   0.1667
 3    1.7143    3.0000    1.7143   -0.0612   0.0476
 4    1.7273    3.0000    1.7273   -0.0165   0.0130
 5    1.7308    3.0000    1.7308   -0.0044   0.0035

A raiz aproximada é 1.7308
```



ATIVIDADE

(01) Calcular a raiz da equação $f(x) = 2x^2 - 10$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 2.2308

(02) Calcular a raiz da equação $f(x) = 2x^3 - 5$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 1.3545

(03) Calcular a raiz da equação $f(x) = x^2 - 16 + senx$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [3 , 5]) Resposta: 4.1032

(04) Calcular a raiz da equação $f(x) = x^2 - 5senx$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [2 , 3]) Resposta: 2.0870

(05) Calcular a raiz da equação $f(x) = -x^2 + 7$ com $E \leq 0{,}01$ utilizando o método das cordas. (Sugestão utilizar intervalo de busca [2 , 4])

(06) Calcular a raiz da equação $f(x) = x^2 - 4 + cos\, x$ com $E \leq 0{,}01$ utilizando o método das cordas. (Sugestão utilizar intervalo de busca [1 , 3])

(07) Calcular a raiz da equação $f(x) = x^3 - 12$ com $E \leq 0{,}01$ utilizando o método das cordas. (Sugestão utilizar intervalo de busca [1 , 3])

3.3. MÉTODO DE NEWTON

Iremos estudar agora, o método de Newton para o cálculo de raízes de uma equação que utiliza informação da primeira e segunda derivada. Semelhantes aos métodos da bisseção e da corda, devemos primeiro isolar a raiz que desejamos procurar dentro de um intervalo $[a\ , b]$ utilizando para isto o método gráfico.

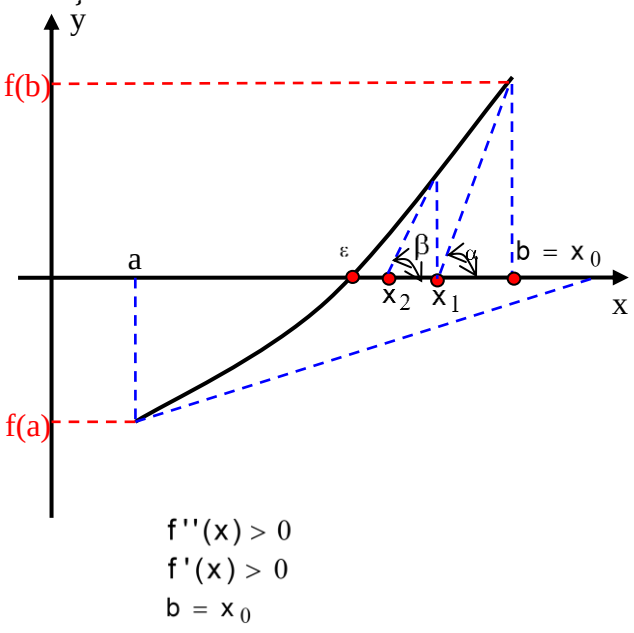
Para utilizarmos o método de Newton é necessários que a função $f(x)$ seja uma continua no intervalo $[a\ , b]$ e que ε o seu único zero neste intervalo; as derivada $f'(x)$ $[f'(x) \neq 0]$ e $f''(x)$ devem também ser contínuas. Para se encontrar a expressão para o cálculo da aproximação x_n para a raiz ε devemos fazer uma expansão em série de Taylor para $f(x) = 0$, de onde temos $f(x) = f(x_n) + f'(x_n)(x - x_n)$ se fizermos $f(x) = f(x_{n+1}) = 0$ obteremos a seguinte expressão $f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0$, isolando o termo x_{n+1} na temos $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, de x_{n+1} é uma aproximação de ε.

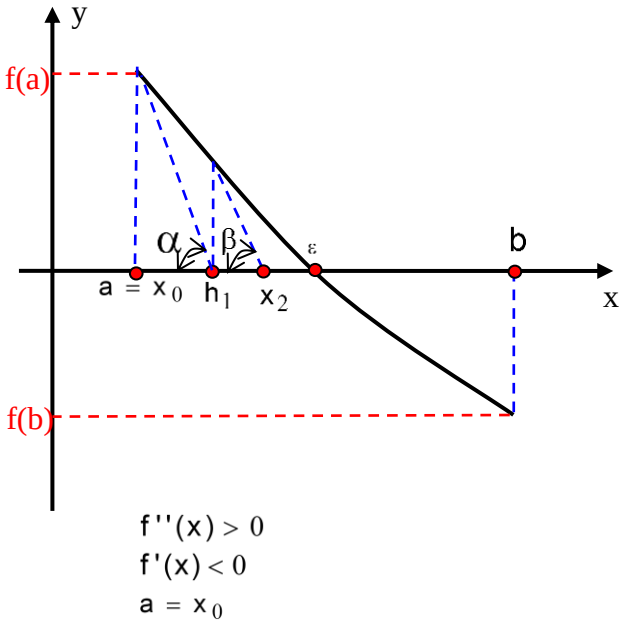
Você sabia, que o método de Newton é equivalente a substituir um pequeno arco de curva $y = f(x)$ por uma reta tangente, traçada a partir de um ponto da curva? Observe, nas figuras a seguir como, no método de Newton, é escolhido o extremo do intervalo $[a\ , b]$ deve ser igual ao valor x_o. Para você compreender melhor a utilização do método de Newton, observe os exemplos numéricos a seguir.

Exemplo:

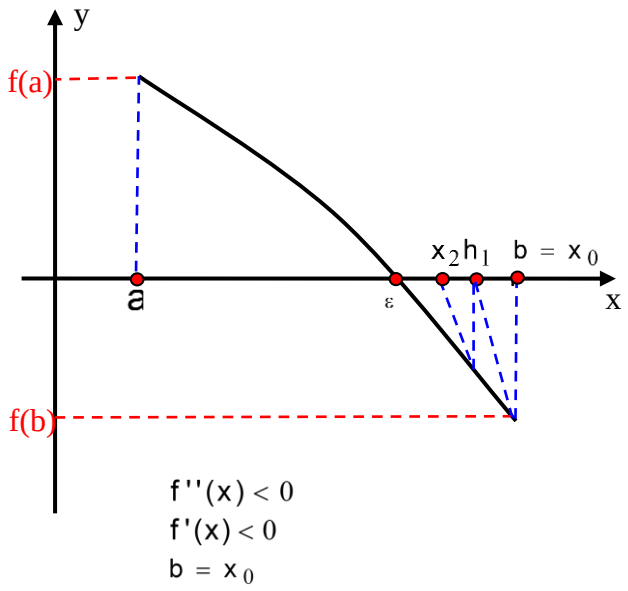
(01) Calcular a raiz da equação $f(x) = x^2 - 3$ com $E \leq 0{,}01$.

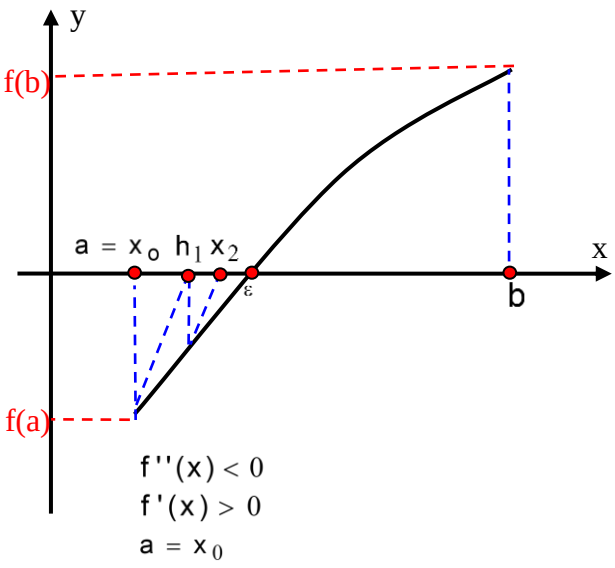
Solução

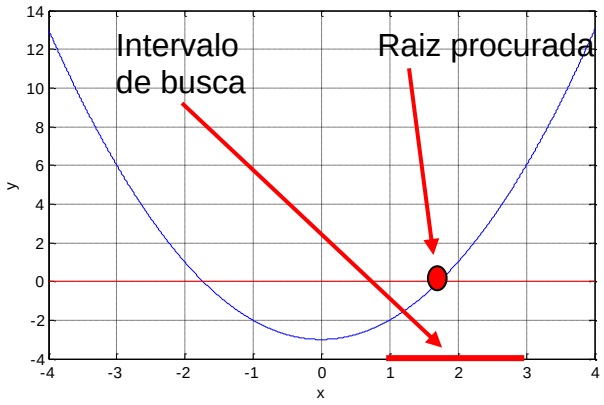




Primeiro devemos determinar um intervalo onde está a raiz que desejamos calcular, para isto devemos fazer uma no seu gráfico.







A raiz procurada está próxima de 2 e está dentro do intervalo [1 , 3]. Logo

N	na	b_n	x_n	$f(x_n)$	E
0	1.0000	3.0000	3.0000	6.0000	
1	1.0000	2.0000	2.0000	1.0000	0.2500
2	1.0000	1.7500	1.7500	0.0625	0.0179
3	1.0000	1.7321	**1.7321**	0.0003	0.0001

Como $f'(x) = 2x \Rightarrow f'(3) = 6 > 0$ e como $f''(x) = 2 > 0$ logo temos $x_0 = b = 3$

usando a expressão $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, temos a seguinte recorrência

$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2.0000 \quad \Rightarrow \quad [a\,, b] = [1.0 \quad 2.0]$

$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.7500 \quad \Rightarrow \quad [a\,, b] = [1.0 \quad 1.75]$

$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 1.7321 \quad \Rightarrow \quad [a\,, b] = [1.0 \quad 1.7321]$

Como o erro é menor que tolerância então a aproximação final é $x = 1{,}7321$.

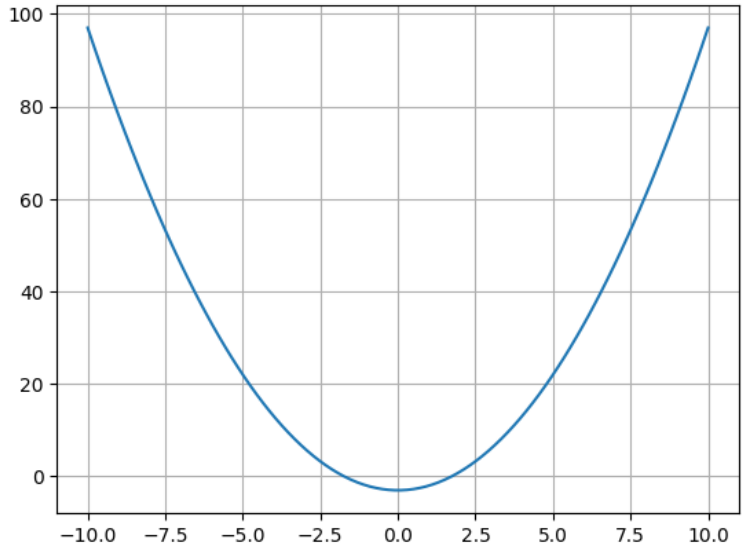
PROGRAMA EM PYTHON

```
# Método de Newton
# Entrada
a = 1                # intervalo [a , b]
b = 3
tolerancia = 0.01    # tolerância
nloop = 50           # número máximo de loop

def f(x):
  return x**2 - 3

def der1(x):
# Derivada de primeira ordem
  dxd1 = 0.0001
  return ( f(x + dxd1) - f(x) ) / dxd1

def der2(x):
# Derivada de segunda ordem
  dxd2 = 0.0001
  d11 = ( f(x) - f(x - dxd2) ) / dxd2
  d12 = ( f(x + dxd2) - f(x) ) / dxd2
  return ( d12 - d11 ) / dxd2

import math

print("Método de Newton")
print(" n      a        b       xn      f(xn)     erro")

vfa = 0
vfb = 0
vder2a = 0
vder2b = 0

if (f(a) >= 0):
  vfa = 1

if (f(b) >= 0):
  vfa = 1

if (der2(a) >= 0):
  vder2a = 1

if (der2(b) >= 0):
  vder2b = 1

if (vder2a == vfa):
    xo = a
    c = b
```

```
if (vder2b == vfb):
    xo = b
    c = a

# Variáveis auxiliares
para = 0
xk = 0
h = 0

erro = 10

if (vder2a == vfa):
  print("%2d"%h, "%8.4f"%xo, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
if (vder2b == vfb):
  print("%2d"%h, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)

while(para == 0):
    xk = xo - (f(xo)/der1(xo));
    erro = abs(xk - xo)
    xo = xk
    h = h + 1
    if (vder2a == vfa):
      print("%2d"%h, "%8.4f"%xo, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
    if (vder2b == vfb):
      print("%2d"%h, "%8.4f"%c, "%8.4f"%xo, "%8.4f"%xo, "%8.4f"%f(xo),
"%8.4f"%erro)
    if ( erro < tolerancia):
      if (h > 1):
        para = 1

print("\nA raiz aproximada é %4.4f \n" %xo)

import matplotlib.pyplot as plt
import numpy as np
xi = np.linspace(-10, 10, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Método de Newton
 n       a          b        xn       f(xn)     erro
 0    1.0000    3.0000    1.0000   -2.0000  10.0000
 1    2.0000    3.0000    2.0000    0.9998   1.0000
 2    1.7500    3.0000    1.7500    0.0625   0.2500
 3    1.7321    3.0000    1.7321    0.0003   0.0179
 4    1.7321    3.0000    1.7321    0.0000   0.0001

A raiz aproximada é 1.7321
```



ATIVIDADES

(01) Calcular a raiz da equação $f(x) = 2x^2 - 10$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 2.2381

(02) Calcular a raiz da equação $f(x) = 2x^3 - 5$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [1 , 3]) Resposta: 1.7150

(03) Calcular a raiz da equação $f(x) = x^2 - 16 + senx$ com $E \leq 0{,}01$ utilizando o método da bisseção. (Sugestão utilizar intervalo de busca [3 , 5]) Resposta: 4.1035

(04) Calcular a raiz da equação $f(x) = -x^2 + 7$ com $E \leq 0{,}01$ utilizando o método de Newton. (Sugestão utilizar intervalo de busca [2 , 4])

(06) Calcular a raiz da equação $f(x) = x^2 - 4 + cos\, x$ com $E \leq 0{,}01$ utilizando o método de Newton. (Sugestão utilizar intervalo de busca [1 , 3])

COMPARAÇÂO DOS MÉTODOS: BISSEÇÃO, CORDAS E NEWTON

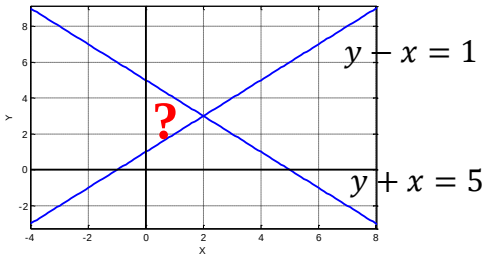
Você observou que os exemplos utilizados nos três métodos (bisseção, cordas e de Newton) são iguais? Fizemos isto, para que você percebesse melhor as diferenças entre os três métodos!

Retorne aos exemplos do método da bisseção e verifique que este método tem convergência lenta, embora este método não necessite de informações da primeira e nem da segunda derivada.

Se você rever os exemplos do método da corda, observará que sua convergência depende da proximidade de x_0 da raiz exata. Você, também irá perceber que este método necessita que sinal da segunda derivada permaneça constante no intervalo, para que haja convergência do resultado. Já o método de Newton necessita da forma analítica da primeira derivada, porém sua convergência e extraordinária.

4. SISTEMAS DE EQUAÇÕES LINEARES

Em nosso dia a dia, a solução de muitos problemas, geralmente está relacionado com a resolução de um sistema. Um exemplo simples é a determinação do ponto de interseção de duas retas: $y + x = 5$ e $y - x = 1$.



Para dar a solução deste problema, devemos resolver o seguinte sistema

$$\begin{cases} y + x = 5 \\ y - x = 1 \end{cases}$$

que tem a seguinte solução é $x = 2$ e $y = 3$. Onde o ponto $(2\,,3)$ corresponde a coordenada da interseção das duas retas.

Para problemas simples, que envolvem um número reduzido de variáveis (2 ou três variáveis), a solução pode ser facilmente obtida com procedimentos simples de substituição ou comparação que aprendemos ao longo do nosso curso primário e secundário. Porém, o grau de dificuldade, na resolução do sistema, aumenta consideravelmente quando aumenta o número de variáveis (acima de 4 variáveis), sendo inclusive necessário o uso de computador para a obtenção de sua solução.

No dia a dia, são vários os problemas que envolvem sistemas com grandes números de incógnitas, como por exemplo, a tomografia médica, onde os sistemas envolvidos chegam a ter mais de 5.000 incógnitas.

Para entendermos os métodos de resolução de sistemas lineares, devemos primeiro compreender que um sistema linear S_n é uma coleção de n equações lineares, como mostraremos a seguir

$$S_n = \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n \end{cases}$$

que pode, também, ser representado por

$S_n = \sum_{i=1}^{n} a_{ij}x_j = b_i$, onde $i = 0,1,2,\ldots,n$

e na forma matricial o sistema S_n pode ser escrito como

$$\underline{Ax} = \underline{b}$$

onde $\underline{A}$ é uma matriz quadrada de ordem n, $\underline{x}$ e $\underline{b}$ não matrizes $n \times 1$, isto é, com n linhas e uma coluna. A matriz $\underline{A}$ tem a seguinte forma

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots. & \dots. & \dots. & \dots. & \dots. \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

onde a_{ij} é chamado coeficiente da incógnita x_j e os b_i são chamados termos independentes. Com a matriz dos coeficientes e a matriz dos termos independentes montamos a matriz $\underline{B}$, denominada de matriz ampliada, que pode ser escrita por

$$\underline{B} = [\underline{A} : \underline{b}]$$

ou seja

$$\underline{B} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n}\, b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n}\, b_2 \\ \dots. & \dots. & \dots. & \dots. & \dots.\dots. \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn}\, b_n \end{bmatrix}$$

Uma solução do sistema S_n , são os valores x_1, x_2, ... , x_n , que constituem a matriz coluna $\underline{x}$, denominada de matriz solução que pode ser escrita por

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Os sistemas lineares S_n podem ser classificados da seguinte forma:

$$S_n = \begin{cases} \text{Homogêneo} \begin{cases} \text{Possível} \begin{cases} \text{Determinado} \\ \text{Indeterminado} \end{cases} \end{cases} \\ \\ \text{Não} - \text{Homogêneo} \begin{cases} \text{Impossível} \\ \text{Possível} \begin{cases} \text{Determinado} \\ \text{Indeterminado} \end{cases} \end{cases} \end{cases}$$

Certamente, você deve estar se questionando sobre alguns itens do diagrama apresentado. Um sistema S_n ($\underline{Ax} = \underline{b}$) é denominado de homogêneo quando a matriz $\underline{b}$, dos termos independentes, é nula, isto é, quando

$$\underline{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Um sistema S_n ($\underline{Ax} = \underline{b}$) é denominado de não-homogêneo quando a matriz $\underline{b}$, não é nula, isto é, existe pelo menos um termo em $\underline{b}$, que não é nulo.

Um sistema é dito impossível quando não há nenhuma solução que satisfaça o sistema, isto é, sua solução é o vazio. Um sistema é dito possível quando há, pelo menos, uma seqüência de valores x_1, x_2, ..., x_n que satisfaça o sistema, isto é, a sua solução nunca é o vazio. Se existir uma única seqüência de valores que satisfaça o sistema S_n, então este sistema é dito Possível e determinado, se existir mais de uma sequência de valores x_1, x_2, ... , x_n que satisfaça o sistema S_n, estão podemos afirmar que o sistema é Possível e indeterminado.

TRANSFORMAÇÕES ELEMENTARES

Você sabia, que o cálculo da solução de sistemas através de métodos interativos, consiste em uma seqüência de transformações, onde um sistema mais complexo é transformado em outro mais simples com a mesma solução.

As transformações utilizadas para modificar os sistemas de equações lineares são formadas pelas seguintes operações elementares:

(1) Trocar a ordem de duas equações do sistema.

(2) Multiplicar uma equação do sistema por uma constante não numa.

(3) Adicionar duas equações do sistema.

A partir das operações apresentadas podemos transformar um sistema S_1 em um sistema S_2. Isto é, S_1 e S_2 são equivalentes.

Para que você possa entender bem estas transformações observe o exemplo a seguir.

Exemplo:

Calcule a solução do sistema $S_1 = \begin{cases} x + y + z = 6 \\ z = 3 \\ y + z = 5 \end{cases}$

Solução

Para obtermos a solução do sistema teremos que fazer uma seqüência de transformações no sistema, observe!

$$S_2 = \begin{cases} x + y + z = 6 \\ y + z = 5 \\ z = 3 \end{cases}$$

O sistema S_2 foi obtido do sistema S_1 a partir da operação: “Trocar a ordem de duas equações do sistema”.

O sistema S_3 foi obtido do sistema S_2 a partir da operação: "Multiplicar uma equação do sistema por uma constante não numa." $\Rightarrow$ Multiplicamos a segunda equação por (-1).

$$S_3 = \begin{cases} x + y + z = 6 \\ -y - z = -5 \\ z = 3 \end{cases}$$

O sistema S_4 foi obtido do sistema S_3 a partir da operação: "Adicionar duas equações do sistema." $\Rightarrow$ Somamos a segunda com a terceira equação de S_3 e colocamos a resposta na segunda equação de S_4.

$$S_4 = \begin{cases} x + y + z = 6 \\ -y = -2 \\ z = 3 \end{cases}$$

Observe que é muito mais fácil calcular a solução do sistema S_4 do que a do sistema S_1. E ambos sistemas possuem a seguinte solução: $x = 1, y = 2$ e $z = 3$.

Se o sistema que você tiver trabalhando tiver 25 incógnitas, como aplicar estas transformações para calcular a solução do seu sistema?

MÉTODO DIRETO

Consiste de métodos que determinam a solução do sistema linear com um número finito de transformações elementares.

4.1. MÉTODO DE GAUSS-JORDAN

Explicaremos o método de Gauss-Jordan com o auxilio do exemplo a seguir.

Exemplo 01 - Calcule a solução do sistema

$$\begin{cases} x + 2y - z = 2 \\ 2x - y + 2z = 6 \\ 3x + 2y - z = 4 \end{cases}$$

Solução

Para facilitar a aplicação do método de Gauss-Jordan devemos, primeiramente, escrever o sistema na forma matricial, isto é:

o sistema $\begin{cases} x + 2y - z = 2 \\ 2x - y + 2z = 6 \\ 3x + 2y - z = 4 \end{cases}$ deve ser escrito por $\begin{bmatrix} 1 & 2 & -1 \\ 2 & -1 & 2 \\ 3 & 2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 4 \end{bmatrix}$

onde

$\underline{A} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & -1 & 2 \\ 3 & 2 & -1 \end{bmatrix}$ é a matriz dos coeficientes e

$\underline{b} = \begin{bmatrix} 2 \\ 6 \\ 4 \end{bmatrix}$ é a matriz dos termos independentes;

Com estas duas matrizes montamos a matriz ampliada $\underline{B}$, onde iremos aplicar as transformações elementares para obtenção da solução do sistema.

$$\underline{B} = [\underline{A}: \underline{b}] = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 2 & -1 & 2 & 6 \\ 3 & 2 & -1 & 4 \end{array}\right]$$

As primeiras transformações que iremos fazer tem como objetivo zerar as posições $a_{21} = 2$ e $a_{31} = 3$ do sistema $\underline{B}$.

$$\underline{B}_0 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 2 & -1 & 2 & 6 \\ 3 & 2 & -1 & 4 \end{array}\right]$$

Para zerar $a_{21} = 2$, usaremos o elemento do pivô desta linha $a_{11} = 1$, para determinar $m_1^{(0)}$:

$m_1^{(0)} = \frac{-a_{21}^{(0)}}{a_{11}^{(0)}} = \frac{-2}{1} = -2$ (0) $\Rightarrow$ significa que tomaremos estes valores da matriz $\underline{B}_0$.

Observe que: $m = \frac{-(valor que se deseja zerar)}{(Valor do pivô nesta coluna)}$

após determinar $m_1^{(0)}$, faremos a seguinte operação

$L_2^{(1)} \rightarrow m_1^{(0)} L_1^{(0)} + L_2^{(0)}$

$L_1^{(0)} \Rightarrow$ tomaremos estes valores da matriz $\underline{B}_0$.

$\Rightarrow$ tomaremos estes valores da linha 1 da matriz $\underline{B}_0$.

$L_2^{(1)} \Rightarrow$ colocaremos estes valores na matriz $\underline{B}_1$.

$\Rightarrow$ colocaremos estes valores na linha 2 da matriz $\underline{B}_1$.

Observe que em todos os cálculos será obedecida esta sequência

$$L_2^{(1)} \rightarrow m_1^{(0)} L_1^{(0)} + L_2^{(0)}$$

onde: $L_1^{(0)}$ é a linha onde está o pivô

$L_2^{(0)}$ é a linha onde está o elemento que queremos zerar

isto é, cada elemento da linha $L_2^{(1)}$ é obtido da combinação linear das linhas $L_1^{(0)}$ e $L_2^{(0)}$ uma matriz $\underline{B}_0$, da seguinte forma:

$a_{21}^{(1)} = m_1^{(0)}.a_{11}^{(0)} + a_{21}^{(0)} = -2 * 1 + 2 = 0$

$a_{22}^{(1)} = m_1^{(0)}.a_{12}^{(0)} + a_{22}^{(0)} = -2 * 2 + (-1) = -5$

$a_{23}^{(1)} = m_1^{(0)}.a_{13}^{(0)} + a_{23}^{(0)} = -2 * (-1) + 2 = 4$

$a_{24}^{(1)} = m_1^{(0)}.a_{14}^{(0)} + a_{24}^{(0)} = -2 * 2 + 6 = 2$

$$\underline{\underline{B}}_1 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 3 & 2 & -1 & 4 \end{array}\right]$$

Para zerar $a_{31} = 3$, usaremos o pivô desta linha $a_{11} = 1$, para determinar $m_1^{(0)}$.

$$\underline{\underline{B}}_1 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 3 & 2 & -1 & 4 \end{array}\right]$$

$m_2^{(0)} = \frac{-a_{31}^{(0)}}{a_{11}^{(0)}} = \frac{-3}{1} = -3$

após determinar $m_1^{(0)}$, faremos a seguinte operação

$L_3^{(1)} \rightarrow m_2^{(0)} L_1^{(0)} + L_3^{(0)}$

isto é, cada elemento da linha $L_3^{(1)}$ é obtido da combinação linear das linhas $L_1^{(0)}$ e $L_3^{(0)}$ uma matriz $\underline{B}_0$, da seguinte forma:

$a_{31}^{(1)} = m_1^{(0)}.a_{11}^{(0)} + a_{31}^{(0)} = -3 * 1 + 3 = 0$

$a_{32}^{(1)} = m_1^{(0)}.a_{12}^{(0)} + a_{32}^{(0)} = -3 * 2 + 2 = -4$

$a_{33}^{(1)} = m_1^{(0)}.a_{13}^{(0)} + a_{33}^{(0)} = -3 * (-1) + (-1) = 2$

$a_{34}^{(1)} = m_1^{(0)}.a_{14}^{(0)} + a_{34}^{(0)} = -3 * 2 + 4 = -2$

$$\underline{\underline{B}}_1 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & -4 & 2 & -2 \end{array}\right]$$

Vamos agora zerar o elemento $a_{32} = -4$, para isto, usaremos o pivô da segunda linha $a_{22} = -5$, para determinar $m_1^{(1)}$.

$$\underline{B}_1 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & -4 & 2 & -2 \end{array}\right]$$

$$m_1^{(1)} = \frac{-a_{32}^{(1)}}{a_{22}^{(1)}} = \frac{-(-4)}{-5} = \frac{-4}{5}$$

após determinar $m_1^{(1)}$, faremos a seguinte operação

$$L_3^{(2)} \rightarrow m_1^{(1)} L_2^{(1)} + L_3^{(1)}$$

isto é, cada elemento da linha $L_3^{(2)}$ é obtido da combinação linear das linhas $L_2^{(1)}$ e $L_3^{(1)}$ uma matriz $\underline{B_1}$, da seguinte forma:

$$a_{31}^{(2)} = m_1^{(1)}.a_{21}^{(1)} + a_{31}^{(1)} = \frac{-4}{5} * 0 + 0 = 0$$

$$a_{32}^{(2)} = m_1^{(1)}.a_{22}^{(1)} + a_{32}^{(1)} = \frac{-4}{5} * (-5) + (-4) = 0$$

$$a_{33}^{(2)} = m_1^{(1)}.a_{23}^{(1)} + a_{33}^{(1)} = \frac{-4}{5} * 4 + 2 = \frac{-6}{5}$$

$$a_{34}^{(2)} = m_1^{(1)}.a_{24}^{(1)} + a_{34}^{(1)} = \frac{-4}{5} * 2 + (-2) = \frac{-18}{5}$$

$$\underline{B}_2 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Observe que as operações realizadas resultaram em um sistema cujos elementos abaixo da diagonal principal (triangulo inferior) são iguais a zero.

$$\underline{B}_2 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Agora o nosso objetivo é zerar o triangulo superior deste sistema

$$\underline{B}_2 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Para isto devemos primeiramente zerar o elemento $a_{23} = 4$, para isto utilizaremos o pivô $a_{33}^{(2)} = -6/5$ para calcular $m_1^{(2)}$

$$\underline{B}_2 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 4 & 2 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

$$m_1^{(2)} = \frac{-a_{23}^{(2)}}{a_{33}^{(2)}} = \frac{-4}{-6/5} = \frac{10}{3}$$

após determinar $m_1^{(2)}$, faremos a seguinte operação

$$L_2^{(3)} \rightarrow m_1^{(2)} L_3^{(2)} + L_2^{(2)}$$

isto é, cada elemento da linha $L_2^{(3)}$ é obtido da combinação linear das linhas $L_2^{(2)}$ e $L_3^{(2)}$ uma matriz $\underline{B}_2$, da seguinte forma:

$$a_{21}^{(3)} = m_1^{(2)}.a_{31}^{(2)} + a_{21}^{(2)} = \frac{10}{3} * 0 + 0 = 0$$

$$a_{22}^{(3)} = m_1^{(2)}.a_{32}^{(2)} + a_{22}^{(2)} = \frac{10}{3} * 0 + (-5) = -5$$

$$a_{23}^{(3)} = m_1^{(2)}.a_{33}^{(2)} + a_{23}^{(2)} = \frac{10}{3} * (-6/5) + 4 = 0$$

$$a_{24}^{(3)} = m_1^{(2)}.a_{34}^{(2)} + a_{24}^{(2)} = \frac{10}{3} * (-18/5) + 2 = -10$$

$$\underline{B}_3 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Vamos, agora, zerar o elemento $a_{13} = -1$, para isto utilizaremos o pivô $a_{33}^{(2)} = -6/5$ para calcular $m_2^{(2)}$

$$\underline{B}_3 = \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

$$m_2^{(2)} = \frac{-a_{13}^{(2)}}{a_{33}^{(2)}} = \frac{-(-1)}{-6/5} = \frac{-5}{6}$$

após determinar $m_2^{(2)}$, faremos a seguinte operação

$$L_1^{(3)} \rightarrow m_2^{(2)} L_3^{(2)} + L_1^{(2)}$$

isto é, cada elemento da linha $L_1^{(3)}$ é obtido da combinação linear das linhas $L_1^{(2)}$ e $L_3^{(2)}$ uma matriz $\underline{B}_2$, da seguinte forma:

$$a_{11}^{(3)} = m_2^{(2)}.a_{31}^{(2)} + a_{11}^{(2)} = \frac{-5}{6} * 0 + 1 = 1$$

$$a_{12}^{(3)} = m_2^{(2)}.a_{32}^{(2)} + a_{12}^{(2)} = \frac{-5}{6} * 0 + 2 = 2$$

$$a_{13}^{(3)} = m_2^{(2)}.a_{33}^{(2)} + a_{13}^{(2)} = \frac{-5}{6} * (-1) + \left(\frac{-5}{6}\right) = 0$$

$$a_{14}^{(3)} = m_2^{(2)}.a_{34}^{(2)} + a_{14}^{(2)} = \frac{-5}{6} * 2 + \left(\frac{-18}{5}\right) = 5$$

$$\underline{B}_3 = \left[\begin{array}{ccc|c} 1 & 2 & 0 & 5 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Vamos agora zerar o elemento $a_{12} = 2$, para isto, usaremos o pivô da segunda linha $a_{22} = -5$, para determinar $m_1^{(3)}$.

$$\underline{B}_3 = \left[\begin{array}{ccc|c} 1 & 2 & 0 & 5 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

$$m_1^{(3)} = \frac{-a_{13}^{(3)}}{a_{33}^{(3)}} = \frac{-2}{-5} = \frac{2}{5}$$

após determinar $m_1^{(3)}$, faremos a seguinte operação

$$L_1^{(4)} \rightarrow m_1^{(3)} L_2^{(3)} + L_1^{(3)}$$

isto é, cada elemento da linha $L_1^{(4)}$ é obtido da combinação linear das linhas $L_1^{(3)}$ e $L_2^{(3)}$ uma matriz $\underline{B_3}$, da seguinte forma:

$$a_{11}^{(4)} = m_1^{(3)}.a_{21}^{(3)} + a_{11}^{(3)} = \frac{2}{5} * 0 + 1 = 1$$

$$a_{12}^{(4)} = m_1^{(3)}.a_{22}^{(3)} + a_{12}^{(3)} = \frac{2}{5} * (-5) + 2 = 0$$

$$a_{13}^{(4)} = m_1^{(3)}.a_{23}^{(3)} + a_{13}^{(3)} = \frac{2}{5} * 0 + 0 = 0$$

$$a_{14}^{(4)} = m_1^{(3)}.a_{24}^{(3)} + a_{14}^{(3)} = \frac{2}{5} * (-10) + 5 = 1$$

$$\underline{B}_4 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Observe que as operações realizadas resultaram em um sistema cujos elementos acima da diagonal principal (triangulo superior) são iguais a zero.

$$\underline{\underline{B}}_4 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & -5 & 0 & -10 \\ 0 & 0 & -6/5 & -18/5 \end{array}\right]$$

Para obtermos a solução do sistema divida cada linha pelo seu respectivo pivô, com isto temos:

$$L_1^{(5)} \to \frac{L_1^{(4)}}{a_{11}^{(4)}} = \frac{L_1^{(4)}}{1}; \quad L_2^{(5)} \to \frac{L_2^{(4)}}{a_{22}^{(4)}} = \frac{L_2^{(4)}}{-5}; \quad L_3^{(5)} \to \frac{L_3^{(4)}}{a_{33}^{(4)}} = \frac{L_3^{(4)}}{-6/5}.$$

Com esta operação obtemos

$$\underline{B}_5 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right] \quad \Rightarrow \quad \begin{cases} x = 1 \\ y = 2 \\ z = 3 \end{cases}$$

PROGRAMA EM PYTHON

```
# Gauss - Jordan - Sistema

import numpy as np

# Entrada (sistema)

M = np.array(
    [[2.0   ,  -1.0   ,  1.0   ,  3.0],
     [1.0   ,   2.0   ,  1.0   ,  8.0],
     [2.0   ,   1.0   ,  2.0   , 10.0]]
    )

print("Gauss - Jordan - Sistema")
print("Matriz Ampliada")
#print(D)

linha = np.size(M[:,1])
coluna = np.size(M[1,:])

for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

print('Linha: {:d}'.format(linha))
print('Coluna: {:d}'.format(coluna))

# Zera Triangulo Infrerior
t = 1
fm = 0
```

```
for j in range(0 , linha , 1):
  for i in range(t , linha , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
      M[i , k] = fm * M[j , k] + M[i , k]
  t = t + 1

print("Zera Triangulo Inferior")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

t = 1
fm = 0

for j in range(linha - 1 , 0 , -1):
  for i in range(0 , linha - t , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
      M[i,k] = fm * M[j,k] + M[i,k]
  t = t + 1

print("Zera Triangulo Superior")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")


fm = 0;

for i in range(0 , linha , 1):
  fm = M[i,i]
  for j in range(0 , coluna , 1):
    M[i,j] = M[i,j]/fm

print("Matriz Normalizada")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

print("Solução do Sistema")
for i in range(0 , linha , 1):
  print("%8.4f"%M[i , coluna-1])
```

SAÍDA DO PROGRAMA

```
Gauss - Jordan - Sistema
Matriz Ampliada
  2.0000  -1.0000   1.0000   3.0000
  1.0000   2.0000   1.0000   8.0000
  2.0000   1.0000   2.0000  10.0000
Linha: 3
Coluna: 4
Zera Triangulo Inferior
  2.0000  -1.0000   1.0000   3.0000
  0.0000   2.5000   0.5000   6.5000
  0.0000   0.0000   0.6000   1.8000
Zera Triangulo Superior
  2.0000   0.0000   0.0000   2.0000
  0.0000   2.5000   0.0000   5.0000
  0.0000   0.0000   0.6000   1.8000
Matriz Normalizada
  1.0000   0.0000   0.0000   1.0000
  0.0000   1.0000   0.0000   2.0000
  0.0000   0.0000   1.0000   3.0000
Solução do Sistema
  1.0000
  2.0000
  3.0000
```

ATIVIDADE

(01) Resolva o sistemas

(a) $\begin{cases} x+y+z=6 \\ x-y-z=-4 \\ x-y+z=2 \end{cases}$ (b) $\begin{cases} x+2y-z=0 \\ x+y+z=7 \\ -x+2y+3z=12 \end{cases}$

(02) Resolva o sistemas

(a) $\begin{cases} x+2y+3z=-1 \\ -x+5y+2z=5 \\ -2x+2y+z=0 \end{cases}$ (b) $\begin{cases} x+2y+3z=8 \\ x+y+2z=5 \\ -2x+y+z=1 \end{cases}$

CÁLCULO DA INVERSA DE UMA MATRIZ

Usaremos, agora, o método de Gauss-Jordan para calcular a inversa de uma matriz. Para que você entender facilmente explicaremos este método, de determinação da inversa de uma matriz, utilizando um exemplo.

Exemplo 01 – Calcule a inversa da matriz $\underline{M} = \begin{bmatrix} 1 & 1 & 2 \\ 0 & -1 & 4 \\ 1 & 1 & 1 \end{bmatrix}$

Solução: No cálculo da inversa de uma matriz ($\underline{M}^{-1}$), a matriz ampliada $\underline{B}$ é montada utilizando a matriz $\underline{M}$ e uma matriz identidade $\underline{I}$ da dimensão da matriz $\underline{M}$. Isto é, a matriz identidade $\underline{I}$ substitui a matriz dos termos independentes $\underline{b}$, utilizada na resolução de sistemas lineares. Deste modo, a matriz $\underline{B}$ fica da forma: $\underline{B} = [\underline{M}: \underline{I}]$

$$\underline{\underline{B}}_0 = \left[\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -1 & 4 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \Longrightarrow \quad m_1^{(0)} = \frac{-a_{31}^{(0)}}{a_{11}^{(0)}} = -1$$

$$\underline{\underline{B}}_1 = \left[\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -1 & 4 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{array}\right] \Longleftarrow \quad L_3^{(1)} \rightarrow m_1^{(0)} L_1^{(0)} + L_3^{(0)}$$

$$\Longrightarrow \quad m_1^{(1)} = \frac{-a_{23}^{(1)}}{a_{33}^{(1)}} = 2$$

$$\underline{\underline{B}}_2 = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & -1 & 0 & 2 \\ 0 & -1 & 4 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{array}\right] \Longleftarrow \quad L_2^{(2)} \rightarrow m_1^{(1)} L_3^{(1)} + L_2^{(1)}$$

$$\Longrightarrow \quad m_2^{(1)} = \frac{-a_{23}^{(1)}}{a_{33}^{(1)}} = 4$$

$$\underline{\underline{B}}_2 = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & -1 & 0 & 2 \\ 0 & -1 & 0 & -4 & 1 & 4 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{array}\right] \Longleftarrow \quad L_2^{(2)} \rightarrow m_2^{(1)} L_3^{(1)} + L_2^{(1)}$$

$$\Longrightarrow \quad m_1^{(2)} = \frac{-a_{12}^{(2)}}{a_{22}^{(2)}} = 1$$

$$\underline{\underline{B}}_3 = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -5 & 1 & 6 \\ 0 & -1 & 0 & -4 & 1 & 4 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{array}\right] \Longleftarrow \quad L_1^{(3)} \rightarrow m_1^{(2)} L_2^{(2)} + L_1^{(2)}$$

$$\underline{\underline{B}}_3 = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -5 & 1 & 6 \\ 0 & 1 & 0 & 4 & -1 & -4 \\ 0 & 0 & 1 & 1 & 0 & -1 \end{array}\right] \Longrightarrow \quad \begin{cases} L_1^{(5)} \rightarrow \dfrac{L_1^{(4)}}{a_{11}^{(4)}} = \dfrac{L_1^{(4)}}{1} \\ L_2^{(5)} \rightarrow \dfrac{L_2^{(4)}}{a_{22}^{(4)}} \dfrac{L_2^{(4)}}{-1} \\ L_3^{(5)} \rightarrow \dfrac{L_3^{(4)}}{a_{33}^{(4)}} = \dfrac{L_3^{(4)}}{-1} \end{cases}$$

$$\underline{\underline{M}} = \begin{bmatrix} 1 & 1 & 2 \\ 0 & -1 & 4 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{e} \quad \underline{\underline{M}}^{-1} = \begin{bmatrix} -5 & 1 & 6 \\ 4 & -1 & -4 \\ 1 & 0 & -1 \end{bmatrix}$$

PROGRAMA EM PYTHON

```
# Gauss - Jordan - Matriz Inversa
```

```
import numpy as np

# Entrada (matriz)

MM = np.array(
    [[-1.0   ,   2.0   ,  1.0],
     [-1.0   ,   1.0   ,  1.0],
     [-1.0   ,   1.0   , -1.0]]
    )

print("Gauss - Jordan - Matriz Inversa")

linha = np.size(MM[:,1])
coluna = np.size(MM[1,:])

print("Matriz a ser invertida")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%MM[i,j], end=' ')
  print(" ")

print('Linha: {:d}'.format(linha))
print('Coluna: {:d}'.format(coluna))

M = np.zeros((linha , 2*linha))

for j in range(0 , linha , 1):
  M[: , j] = MM[: , j]

M0 = np.eye(3)
for j in range(linha , 2*linha , 1):
  M[: , j] = M0[: , j - linha]

print("Matriz a ser escalonada")
for i in range(0 , linha , 1):
  for j in range(0 , 2*linha  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

# Zera Triangulo Infrerior
t = 1
fm = 0

for j in range(0 , linha , 1):
  for i in range(t , linha , 1):
```

```
    fm = - M[i,j]/M[j,j]
    for k in range(0 , 2*linha , 1):
      M[i , k] = fm * M[j , k] + M[i , k]
  t = t + 1

print("Zera Triangulo Inferior")
for i in range(0 , linha , 1):
  for j in range(0 , 2*linha  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

t = 1
fm = 0

for j in range(linha - 1 , 0 , -1):
  for i in range(0 , linha - t , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , 2*linha , 1):
      M[i,k] = fm * M[j,k] + M[i,k]
  t = t + 1

print("Zera Triangulo Superior")
for i in range(0 , linha , 1):
  for j in range(0 , 2*linha  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")


fm = 0;

for i in range(0 , linha , 1):
  fm = M[i,i]
  for j in range(0 , 2*linha , 1):
    M[i,j] = M[i,j]/fm

print("Matriz Normalizada")
for i in range(0 , linha , 1):
  for j in range(0 , 2*linha  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

print("Matriz Inversa")
for i in range(0 , linha , 1):
  for j in range(linha , 2*linha  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")
```

SAÍDA DO PROGRAMA

```
Gauss - Jordan - Matriz Inversa
Matriz a ser invertida
 -1.0000   2.0000   1.0000
 -1.0000   1.0000   1.0000
 -1.0000   1.0000  -1.0000
Linha: 3
Coluna: 3
Matriz a ser escalonada
 -1.0000   2.0000   1.0000   1.0000   0.0000   0.0000
 -1.0000   1.0000   1.0000   0.0000   1.0000   0.0000
 -1.0000   1.0000  -1.0000   0.0000   0.0000   1.0000
Zera Triangulo Inferior
 -1.0000   2.0000   1.0000   1.0000   0.0000   0.0000
  0.0000  -1.0000   0.0000  -1.0000   1.0000   0.0000
  0.0000   0.0000  -2.0000   0.0000  -1.0000   1.0000
Zera Triangulo Superior
 -1.0000   0.0000   0.0000  -1.0000   1.5000   0.5000
  0.0000  -1.0000   0.0000  -1.0000   1.0000   0.0000
  0.0000   0.0000  -2.0000   0.0000  -1.0000   1.0000
Matriz Normalizada
  1.0000  -0.0000  -0.0000   1.0000  -1.5000  -0.5000
 -0.0000   1.0000  -0.0000   1.0000  -1.0000  -0.0000
 -0.0000  -0.0000   1.0000  -0.0000   0.5000  -0.5000
Matriz Inversa
  1.0000  -1.5000  -0.5000
  1.0000  -1.0000  -0.0000
 -0.0000   0.5000  -0.5000
```

ATIVIDADE

(01) Determine a inversa das matriz abaixo

(a) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$ Resposta = $\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & -1/2 \\ 0 & -1/2 & 1/2 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & 1 \\ -1 & 2 & 3 \end{bmatrix}$ Resposta = $\begin{bmatrix} -1/10 & 4/5 & -3/10 \\ 2/5 & -1/5 & 1/5 \\ -3/10 & 2/5 & 1/10 \end{bmatrix}$

(02) Determine a inversa das matrizes abaixo

(a) $\begin{bmatrix} 1 & 2 & 3 \\ -1 & 5 & 2 \\ -2 & 2 & 1 \end{bmatrix}$ (b) $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ -2 & 1 & 1 \end{bmatrix}$

CÁLCULO DO DETERMINANTE DE UMA MATRIZ

O método de Gauss-Jordan, também pode ser utilizado para calcularmos o determinante de uma matriz. Para isto, devemos escalonar a matriz ampliada $\underline{B}$, como fizemos no cálculo da solução do sistema e na determinação da matriz inversa, porém não devemos fazer o último passo, que é a normalização da matriz pelos elementos da diagonal principal. Para que você entender melhor observe o exemplo a seguir, onde iremos calcular o determinante de uma matriz utilizando o método de Gauss-Jordan.

Exemplo 02 – Calcule o determinante da matriz $\underline{M} = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 1 \\ 1 & 2 & -1 \end{bmatrix}$

Solução: Observe que a matriz que iremos calcular o determinante é a mesma do exemplo anterior. Fizemos isto, para que você entendesse melhor que os passos utilizados no calculo do determinante são os mesmo utilizados na inversa da matriz. Devemos primeiramente montar a matriz ampliada $\underline{B} = [\underline{M} : \underline{I}]$

$$\underline{\underline{B}}_0 = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 1 \\ 1 & 2 & -1 \end{bmatrix} \Longrightarrow \quad m_1^{(0)} = \frac{-a_{31}^{(0)}}{a_{11}^{(0)}} = -1$$

$$L_3^{(1)} \rightarrow m_1^{(0)} L_1^{(0)} + L_3^{(0)}$$

$$\underline{\underline{B}}_1 = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 1 \\ 0 & -1 & -1 \end{bmatrix} \Longrightarrow \quad m_1^{(1)} = \frac{-a_{32}^{(1)}}{a_{22}^{(1)}} = 0.5$$

$$L_3^{(2)} \rightarrow m_1^{(1)} L_2^{(1)} + L_3^{(1)}$$

$$\underline{\underline{B}}_2 = \begin{bmatrix} 1.00 & 3.00 & 0 \\ 0 & 2.00 & 1.00 \\ 0 & 0 & -0.50 \end{bmatrix} \Longrightarrow \quad m_1^{(2)} = \frac{-a_{23}^{(2)}}{a_{33}^{(2)}} = 2$$

$$L_2^{(3)} \rightarrow m_2^{(2)} L_3^{(2)} + L_2^{(2)}$$

$$\underline{\underline{B}}_3 = \begin{bmatrix} 1.00 & 3.00 & 0 \\ 0 & 2.00 & 0 \\ 0 & 0 & -0.50 \end{bmatrix} \Longrightarrow \quad m_1^{(3)} = \frac{-a_{12}}{a_{22}^{(3)}} = -1.5$$

$$L_1^{(4)} \rightarrow m_1^{(3)} L_2^{(3)} + L_1^{(3)}$$

$$\underline{\underline{B}}_4 = \begin{bmatrix} 1.00 & 0 & 0 \\ 0 & 2.00 & 0 \\ 0 & 0 & -0.50 \end{bmatrix}$$

$$\det(\underline{\underline{M}}) = 1.00 * 2.00 * (-0.50) = -1.00$$

PROGRAMA EM PYTHON

```
# Gauss - Jordan - Determinante

import numpy as np

# Entrada (sistema)

M = np.array(
    [[1.0   ,   2.0   ,  2.0],
     [2.0   ,  -2.0   ,  2.0],
     [2.0   ,  -1.0   ,  2.0]]
    )

print("Gauss - Jordan - Determinante")
print("Matriz")

linha = np.size(M[:,1])
coluna = np.size(M[1,:])

for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

#print('Linha: {:d}'.format(linha))
#print('Coluna: {:d}'.format(coluna))

# Zera Triangulo Infrerior
t = 1
fm = 0

for j in range(0 , linha , 1):
  for i in range(t , linha , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
      M[i , k] = fm * M[j , k] + M[i , k]
  t = t + 1

print("\nZera Triangulo Inferior")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

t = 1
fm = 0
```

```
for j in range(linha - 1 , 0 , -1):
  for i in range(0 , linha - t , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
      M[i,k] = fm * M[j,k] + M[i,k]
  t = t + 1

print("Zera Triangulo Superior")
for i in range(0 , linha , 1):
  for j in range(0 , coluna  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

det = 1.

for i in range(0 , linha , 1):
  det = det * M[i,i]

print("\nDeterminante: %8.4f" %det)
```

SAÍDA DO PROGRAMA

```
Gauss - Jordan - Determinante
Matriz
  1.0000   2.0000   2.0000
  2.0000  -2.0000   2.0000
  2.0000  -1.0000   2.0000

Zera Triangulo Inferior
  1.0000   2.0000   2.0000
  0.0000  -6.0000  -2.0000
  0.0000   0.0000  -0.3333
Zera Triangulo Superior
  1.0000   0.0000   0.0000
  0.0000  -6.0000   0.0000
  0.0000   0.0000  -0.3333

Determinante:   2.0000
```

ATIVIDADE

(01) Determine o determinante das matrizes abaixo

(a) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$ (b) $\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & 1 \\ -1 & 2 & 3 \end{bmatrix}$

(02) Calcule o determinante das matrizes

(a) $\begin{bmatrix} 1 & 2 & 3 \\ -1 & 5 & 2 \\ -2 & 2 & 1 \end{bmatrix}$ (b) $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ -2 & 1 & 1 \end{bmatrix}$

MÉTODOS ITERATIVOS

A outra forma de se determinar a solução de um sistema $\underline{Ax} = \underline{b}$, que é através dos métodos iterativos. Os métodos iterativos consistem em determinar uma sequência de aproximações $\underline{x}^{(1)}$, $\underline{x}^{(2)}$, ... , $\underline{x}^{(k)}$, para a solução do sistema $\underline{x}$, a partir de uma dada aproximação inicial $\underline{x}^{(0)}$.

Segundo este raciocínio, o sistema $\underline{Ax} = \underline{b}$, é transformado em um outro sistema equivalente com a seguinte forma

$$\underline{x}^{(k+1)} = \underline{Fx}^{(k)} + \underline{d}$$

onde $\underline{F}$ é uma matriz $n \times n$, $\underline{x}$ e $\underline{d}$ são matrizes $n \times 1$. $\underline{x}^{(k+1)}$ é uma aproximação obtida a partir da aproximação $\underline{x}^{(k)}$. Sendo a seqüência de aproximações obtida da seguinte forma

$$\underline{x}^{(1)} = \underline{Fx}^{(0)} + \underline{d}$$
$$\underline{x}^{(2)} = \underline{Fx}^{(1)} + \underline{d}$$
$$\underline{x}^{(3)} = \underline{Fx}^{(2)} + \underline{d}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\underline{x}^{(k+1)} = \underline{Fx}^{(k)} + \underline{d}$$

As aproximações são calculadas até que se tenha

$$\left\|\underline{x}^{(k)} - \underline{x}\right\| = \underset{1 \le i \le n}{max}\left\{\underline{x}_i^{(k)} - \underline{x}_i\right\}$$

O que garante que a sequência $\underline{x}^{(1)}$, $\underline{x}^{(2)}$, ... , $\underline{x}^{(k)}$ converge para a solução? Se $\lim_{k\to\infty}\left\|\underline{x}^{(k)} - \underline{x}\right\| = 0$, então a seqüência $\underline{x}^{(1)}$, $\underline{x}^{(2)}$, ... , $\underline{x}^{(k)}$ converge para a solução $\underline{x}$.

O que diferencia um método iterativo de outro são as de definirmos as matrizes $\underline{F}$ e $\underline{d}$. A seguir apresentaremos o metido de Jacobi que será o nosso primeiro método iterativo.

3.4. MÉTODO DE JACOBI

Para entendermos o método de Jacobi tomemos o sistema

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{b2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

em cada equação do sistema devemos isolar o valor de $\underline{x_i}$, isto é, na primeira equação devemos isolar $\underline{x_1}$, na segunda equação devemos isolar $\underline{x_2}$, e assim por diante, com isto teremos:

$$\begin{cases} x_1 = \dfrac{b_1 - (a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n)}{a_{11}} \\ x_2 = \dfrac{b_2 - (a_{21}x_1 + a_{13}x_3 + \ldots + a_{2n}x_n)}{a_{22}} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = \dfrac{b_n - (a_{n1}x_1 + a_{b2}x_2 + a_{13}x_3 + \ldots + a_{nn-1}x_{n-1})}{a_{nn}} \end{cases}$$

É importante você observar que os elementos a_{ii} devem ser diferentes de zeros $a_{ii} \neq 0, \forall i$, se não teremos divisão por zero. Caso isto não ocorra devemos reagrupar o sistema para que se consiga esta condição

Podemos colocar o sistema na seguinte forma $\underline{x}^{(k+1)} = \underline{F}\underline{x}^{(k)} + \underline{d}$, onde

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \underline{d} = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix}$$

$$\underline{F} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & \ldots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \ldots & -\frac{a_{2n}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 & \ldots & -\frac{a_{3n}}{a_{33}} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & -\frac{a_{n3}}{a_{nn}} & \ldots & 0 \end{bmatrix}$$

O método de Jacobi funciona da seguinte forma:

1º Passo: Devemos escolher uma aproximação inicial $\underline{x}^{(0)}$.

2º Passo: Devemos gerar as aproximações $\underline{x}^{(k)}$ a partir das iterações

$\underline{x}^{(k+1)} = \underline{F}\underline{x}^{(k)} + \underline{d}, \qquad k = 0,1,2,\ldots$

3º Passo: Paramos de calcular as aproximações quando um dos critérios de parada abaixo for satisfeito.

1º critério: $\underset{1 \leq i \leq n}{max} |\underline{x}_i^{(k+1)} - \underline{x}_i^{(k)}| \leq E$, onde E é a tolerância.

2º critério: $k > M$, onde M é o número máximo de iterações.

A tolerância E fixa o grau de precisão das soluções. Para você compreender melhor o método de Jacobi observe o exemplo a seguir.

Exemplo 01 – Resolva pelo método de Jacobi o sistema

$\begin{cases} 2x_1 - x_2 = 1 \\ x_1 + 2x_2 = 3 \end{cases}$ com $E \leq 10^{-2}$ ou $k > 10$.

Solução

Isolando o valor de x_1 na primeira equação e x_2 na segunda equação, temos as equações de iteração

$$\begin{cases} x_1^{k+1} = \frac{1}{2}(1 + x_2^k) \\ x_2^{k+1} = \frac{1}{2}(3 - x_1^k) \end{cases} \quad \text{onde} \quad k = 0,1,2,\ldots$$

Utilizaremos como aproximação inicial $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ para calcular $\underline{x}^{(1)}$, como mostraremos a seguir

Para $k = 0$

$$\begin{cases} x_1^1 = \frac{1}{2}(1 + x_2^0) \\ x_2^1 = \frac{1}{2}(3 - x_1^0) \end{cases} \Rightarrow \begin{cases} x_1^1 = \frac{1}{2}(1 + 0) = 0.5 \\ x_2^1 = \frac{1}{2}(3 - 0) = 1.5 \end{cases} \Rightarrow x^{(1)} = \begin{bmatrix} 0.5 \\ 1.5 \end{bmatrix}$$

Para $k = 1$

$$\begin{cases} x_1^2 = \frac{1}{2}(1 + x_2^1) \\ x_2^2 = \frac{1}{2}(3 - x_1^1) \end{cases} \Rightarrow \begin{cases} x_1^1 = \frac{1}{2}(1 + 0.5) = 1.25 \\ x_2^1 = \frac{1}{2}(3 - 1.5) = 1.25 \end{cases} \Rightarrow x^{(2)} = \begin{bmatrix} 1.25 \\ 1.25 \end{bmatrix}$$

repetiremos estes cálculos para $k = 2, 3, \ldots$. e colocamos os valores obtidos na tabela abaixo:

k	x_1^k	x_2^k	E
0	0.0000	0.0000	0.0000
1	0.5000	1.5000	1.5000
2	1.2500	1.2500	0.7500
3	1.1250	0.8750	0.3750
4	0.9375	0.9375	0.1875
5	0.9688	1.0313	0.0938
6	1.0156	1.0156	0.0469
7	1.0078	0.9922	0.0234
8	0.9961	0.9961	0.0117
9	0.9980	1.0020	0.0059
10	**1.0010**	**1.0010**	0.0029

como

$$\left.\begin{array}{l} 0.0029 \leq 10^{-2} \\ ou \\ k > 10? \end{array}\right\} \Rightarrow \begin{cases} x_1 = 1.0010 \\ x_2 = 1.0010 \end{cases} \Rightarrow \underline{x} = \begin{bmatrix} 1.0010 \\ 1.0010 \end{bmatrix}$$

Exemplo 02 – Resolva pelo método de Jacobi o sistema

$$\begin{cases} x_1 - 0.25x_2 - 0.25x_3 + 0 = 0 \\ -0.25x_1 + x_2 - 0 - 0.25x_4 = 0 \\ -0.25x_1 + 0 + x_3 - 0.25x_4 = -0.25 \\ 0 - 0.25x_2 + 0 + x_4 = -0.25 \end{cases}$$

com $E \leq 10^{-2}$ ou $k > 10$ e $\underline{x} = [0000]$.

Solução

Isolando o valor de x_1 na primeira equação, x_2 na segunda equação, x_3 na terceira equação e x_4 na quarta equação, obtemos as equações de iteração

$$\begin{cases} x_1^{k+1} = 0.25x_2^k + 0.25x_3^k \\ x_2^{k+1} = 0.25x_1^k + 0.25x_4^k \\ x_3^{k+1} = 0.25x_1^k + 0.25x_4^k - 0.25 \\ x_4^{k+1} = 0.25x_2^k - 0.25 \end{cases} \quad \text{onde} \quad k = 0,1,2,\ldots$$

Utilizaremos como aproximação inicial $x^{(0)} = [0000]$, com os valores das aproximações montaremos a próxima tabela.

k	x_1^k	x_2^k	x_3^k	x_4^k	E
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1.0000	0.0000	0.0000	-0.2500	-0.2500	0.2500
2.0000	-0.0625	-0.0063	-0.2562	-0.2500	0.0625
3.0000	-0.0656	-0.0219	-0.2719	-0.2516	0.0156
4.0000	-0.0734	-0.0227	-0.2727	-0.2555	0.0078
5.0000	-0.0738	-0.0247	-0.2747	-0.2557	0.0021
6.0000	-0.0749	-0.0249	-0.2749	-0.2562	0.0010
7.0000	**-0.0749**	**-0.0251**	**-0.2751**	**-0.2562**	0.0003

como

$$\left.\begin{array}{l} 0.0003 \leq 10^{-2} \\ ou \\ k > 10? \end{array}\right\} \Rightarrow \begin{cases} x_1 = -0.0749 \\ x_2 = -0.0251 \\ x_3 = -0.2751 \\ x_4 = -0.2562 \end{cases} \Rightarrow \underline{x} = \begin{bmatrix} -0.0749 \\ -0.0251 \\ -0.2751 \\ -0.2562 \end{bmatrix}$$

PROGRAMA EM PYTHON

```
#Jacobi - Sistema

import numpy as np

# Entrada (sistema)
nloop = 50   # numero máximo de loop
erro = 0.001  # tolerância
```

```
M = np.array(
    [[3.0   ,   1.0   ,  1.0   ,  8.0],
     [1.0   ,  -2.0   ,  2.0   ,  3.0],
     [1.0   ,  -1.0   ,  3.0   ,  8.0]]
    )

print("Jacobi - Sistema")

tole = 10
pare = 0
v = 0
i1 = 0
i2 = 1
m = np.size(M[:,1])
n = np.size(M[1,:])

print("Matriz Ampliada")
for i in range(0 , m , 1):
  for j in range(0 , n  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

#print('m: {:d}'.format(m))
#print('n: {:d}'.format(n))

R = np.zeros((nloop , m))
v = np.zeros((m))
va = 0;
k = 0

print('\n k    x1       x2       x3       Erro')

while(pare == 0):
  for i in range(0 , m , 1):
    va = 0
    for j in range(0 , m , 1):
      if(i == j):
        va = va + 0.0
      if(i != j):
        va = va + ( M[i,j] * R[i1,j] )
      R[i2,i] = (1/M[i,i])*(M[i,n-1]- va)
  print("%2d"%k , "%8.4f"%R[k , 0] , "%8.4f"%R[k , 1] , "%8.4f"%R[k ,
2] , "%8.4f"%tole)
  tole = 10
  if (k >= 0):
    for i in range(0 , m , 1):
      v[i] = abs(R[i2,i] - R[i1,i]);
      tole = max(v[:]);
  if(tole < erro):
```

```
    print("\nSolução do sistema")
    for t in range(0 , m  , 1):
      print("%8.2f"%R[i2 , t], end=' ')
    pare = 1

  if(i1 == nloop):
    pare = 1
  i1 = i1 + 1
  i2 = i2 + 1
  k = k + 1
```

SAÍDA DO PROGRAMA

```
Jacobi - Sistema
Matriz Ampliada
  3.0000   1.0000   1.0000   8.0000
  1.0000  -2.0000   2.0000   3.0000
  1.0000  -1.0000   3.0000   8.0000

 k    x1       x2       x3       Erro
 0   0.0000   0.0000   0.0000  10.0000
 1   2.6667  -1.5000   2.6667   2.6667
 2   2.2778   2.5000   1.2778   4.0000
 3   1.4074   0.9167   2.7407   1.5833
 4   1.4475   1.9444   2.5031   1.0278
 5   1.1842   1.7269   2.8323   0.3292
 6   1.1469   1.9244   2.8476   0.1975
 7   1.0760   1.9210   2.9258   0.0782
 8   1.0510   1.9638   2.9483   0.0428
 9   1.0293   1.9739   2.9709   0.0226
10   1.0184   1.9856   2.9815   0.0117
11   1.0110   1.9907   2.9891   0.0075
12   1.0067   1.9945   2.9933   0.0042
13   1.0041   1.9966   2.9959   0.0027
14   1.0025   1.9980   2.9975   0.0016

Solução do sistema
    1.00     2.00     3.00
```

ATIVIDADE

(01) Resolva os sistemas, com $x_0 = [000]$, $E \leq 10^{-2}$ ou $k < 10$, onde k iterações.

(a) $\begin{cases} 2x - y + z = 2 \\ x + 2y + z = 4 \\ 2x + y + 2z = 5 \end{cases}$ (b) $\begin{cases} 4x - y + z = 5 \\ x + 2y + z = 5 \\ x - 3y + 3z = 4 \end{cases}$

(02) Resolva os sistemas

(a) $\begin{cases} 3x + y - z = 2 \\ x + 5y + z = 14 \\ x - y - 3z = -10 \end{cases}$ (b) $\begin{cases} 3x - y + z = 4 \\ -x + 4y + z = 10 \\ -x - y + 3z = 6 \end{cases}$

4.5. MÉTODO DE GAUSS-SEIDEL

O método Gauss-Seidel é um outro método iterativo para calcular a solução de sistemas lineares. Sua conversão é mais rápida do que o método de Jacobi .

O método iterativo de Gauss-Seidel consiste em:

1º Passo: Definirmos uma aproximação inicial $\underline{x}^{(0)}$.

2º Passo: Calcula-se a sequência de aproximações $\underline{x}^{(1)}$, $\underline{x}^{(2)}$, ... , $\underline{x}^{(k)}$ utilizando-se as seguintes fórmulas:

$$x_1^{(k+1)} = \frac{1}{a_{11}}\left[b_1 - a_{12}\,x_2^{(k)} - a_{13}\,x_3^{(k)} - a_{13}\,x_3^{(k)} - \cdots - a_{1n}\,x_n^{(k)}\right]$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}\left[b_2 - a_{21}\,x_1^{(k+1)} - a_{23}\,x_3^{(k)} - a_{23}\,x_3^{(k)} - \cdots - a_{2n}\,x_n^{(k)}\right]$$

$$x_3^{(k+1)} = \frac{1}{a_{33}}\left[b_3 - a_{31}\,x_1^{(k+1)} - a_{32}\,x_2^{(k+1)} - a_{34}\,x_4^{(k)} - \cdots - a_{3n}\,x_n^{(k)}\right]$$

$$\vdots$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}\left[b_n - a_{n1}\,x_1^{(k+1)} - a_{n2}\,x_2^{(k+1)} - a_{n4}\,x_4^{(k+1)} - \cdots - a_{n,n-1}\,x_{n-1}^{(k+1)}\right]$$

Observe que no cálculo da aproximação $x_n^{(k+1)}$, utilizamos as aproximações $x_1^{(k+1)}$, $x_2^{(k+1)}$, ... , $x_{n-1}^{(k+1)}$. Isto faz com que este método tenha convergência mais rápida. Explicaremos o método iterativo de Gauss-Seidel com o auxílio do exemplo a seguir.

Exemplo 01 - Exemplo 01 – Resolva pelo método de Jacobi o sistema

$$\begin{cases} 2x_1 - x_2 = 1 \\ x_1 + 2x_2 = 3 \end{cases} \quad \text{com } \underline{x}^{(0)} = [00], \;\; E \leq 10^{-2} \quad \text{ou} \quad k > 10.$$

Solução

Isolando o valor de x_1 na primeira equação e x_2 na segunda equação, temos as equações de iteração

$$\begin{cases} x_1^{k+1} = \frac{1}{2}(1 + x_2^k) \\ x_2^{k+1} = \frac{1}{2}(3 - x_1^{k+1}) \end{cases} \quad \text{onde} \quad k = 0,1,2,\ldots$$

O calculo das aproximações é feito da seguinte forma

Para $k = 0$ (1ª iteração)

$$\begin{cases} x_1^{(1)} = \frac{1}{2}(1 + x_2^{(0)}) \\ x_2^{(1)} = \frac{1}{2}(3 - x_1^{(1)}) \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{2}(1 + 0) = 0.5 \\ x_2^{(1)} = \frac{1}{2}(3 - 0.5) = 1.25 \end{cases} \Rightarrow \quad x^{(1)} = \begin{bmatrix} 0.5 \\ 1.25 \end{bmatrix}$$

Para $k = 1$ (2ª iteração)

$$\begin{cases} x_1^{(2)} = \frac{1}{2}(1 + x_2^{(1)}) \\ x_2^{(2)} = \frac{1}{2}(3 - x_1^{(2)}) \end{cases} \Rightarrow \begin{cases} x_1^{(2)} = \frac{1}{2}(1 + 1.25) = 1.125 \\ x_2^{(2)} = \frac{1}{2}(3 - 1.125) = 0.9375 \end{cases} \Rightarrow \quad x^{(2)} = \begin{bmatrix} 1.125 \\ 0.9375 \end{bmatrix}$$

repetiremos estes cálculos para $k = 2,3,\ldots.$ e colocamos os valores obtidos na tabela a seguir.

K	x_1^k	x_2^k	E
0	0.0000	0.0000	0.0000
1	0.5000	1.2500	1.2500
2	1.1250	0.9375	0.6250
3	0.9688	1.0156	0.1563
4	1.0078	0.9961	0.0391
5	0.9980	1.0010	0.0098
6	1.0005	0.9998	0.0024
7	**0.9999**	**1.0001**	0.0006

como

$$\left.\begin{array}{l} 0.0006 \leq 10^{-2} \\ ou \\ k > 10? \end{array}\right\} \Rightarrow \begin{cases} x_1 = 0.9999 \\ x_2 = 1.0001 \end{cases} \Rightarrow \quad \underline{x} = \begin{bmatrix} 0.9999 \\ 1.0001 \end{bmatrix}$$

Perceba que este método converge mais rápido, comparando este exemplo com o primeiro exemplo do método Jacobi. Para facilitar a nossa comparação entre os métodos de Jacobi e Gauss-Seidel, resolveremos a seguir, pelo método de Gauss-Seidel, o exemplo resolvido pelo método de Jacobi.

Exemplo 02 – Resolva pelo método de Gauss-Seidel o sistema

$$\begin{cases} x_1 - 0.25x_2 - 0.25x_3 + 0 = 0 \\ -0.25x_1 + x_2 + 0 - 0.25x_4 = 0 \\ -0.25x_1 + 0 + x_3 - 0.25x_4 = -0.25 \\ 0 - 0.25x_2 + 0 + x_4 = -0.25 \end{cases}$$

com $E \leq 10^{-2}$ ou $k > 10$ e $\underline{x}^{(0)} = [0000]$.

Solução

Isolando o valor de x_1 na primeira equação, x_2 na segunda equação, x_3 na terceira equação e x_4 na quarta equação, obtemos as equações de iteração

$$\begin{cases} x_1^{(k+1)} = 0.25x_2^{(k)} + 0.25x_3^{(k)} \\ x_2^{(k+1)} = 0.25x_1^{(k+1)} + 0.25x_4^{(k)} \\ x_3^{(k+1)} = 0.25x_1^{(k+1)} + 0.25x_4^{(k)} - 0.25 \\ x_4^{(k+1)} = 0.25x_2^{(k+1)} - 0.25 \end{cases} \quad \text{onde} \quad k = 0,1,2,\ldots$$

Utilizaremos como aproximação inicial $\underline{x}^{(0)} = [0000]$, com os valores das aproximações montaremos a próxima tabela.

k	x_1^k	x_2^k	x_3^k	x_4^k	E
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1.0000	0.0000	0.0000	-0.2500	-0.2500	0.2500
2.0000	-0.0625	-0.0219	-0.2719	-0.2555	0.0625
3.0000	-0.0734	-0.0247	-0.2747	-0.2562	0.0109
4.0000	-0.0749	-0.0251	-0.2751	-0.2563	0.0014
5.0000	**-0.0751**	**-0.0252**	**-0.2752**	**-0.2563**	0.0002

como

$$\left.\begin{array}{l} 0.0002 \leq 10^{-2} \\ ou \\ k > 10? \end{array}\right\} \Rightarrow \begin{cases} x_1 = -0.0751 \\ x_2 = -0.0252 \\ x_3 = -0.2752 \\ x_4 = -0.2563 \end{cases} \Rightarrow \underline{x} = \begin{bmatrix} -0.0751 \\ -0.0252 \\ -0.2752 \\ -0.2563 \end{bmatrix}$$

PROGRAMA EM PYTHON

```
#Gauss - Seidel - Sistema

import numpy as np

# Entrada (sistema)
nloop = 50   # numero máximo de loop
erro = 0.001  # tolerância

M = np.array(
```

```
    [[2.0   ,  -1.0   ,  2.0   ,  6.0],
     [1.0   ,   2.0   ,  2.0   , 11.0],
     [1.0   ,  -1.0   ,  3.0   ,  8.0]]
    )

print("Gauss - Seidel - Sistema")

tole = 10
pare = 0
v = 0
i1 = 0
i2 = 1
m = np.size(M[:,1])
n = np.size(M[1,:])

print("Matriz Ampliada")
for i in range(0 , m , 1):
  for j in range(0 , n  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

#print('m: {:d}'.format(m))
#print('n: {:d}'.format(n))

R = np.zeros((nloop , m))
v = np.zeros((m))
va = 0;
k = 0

print('\n k    x1       x2       x3       Erro')

while(pare == 0):
  for i in range(0 , m , 1):
    va = 0
    for j in range(0 , m , 1):
      if(i == j):
        va = va + 0.0
      if(i != j):
        if(i < j):
          va = va + ( M[i,j] * R[i1,j] )
        if(i > j):
          va = va + ( M[i,j] * R[i2,j] )
      R[i2,i] = (1/M[i,i])*(M[i,n-1]- va)
  print("%2d"%k , "%8.4f"%R[k , 0] , "%8.4f"%R[k , 1] , "%8.4f"%R[k ,
2] , "%8.4f"%tole)
  tole = 10
  if (k >= 0):
    for i in range(0 , m , 1):
      v[i] = abs(R[i2,i] - R[i1,i]);
```

```
      tole = max(v[:]);
  if(tole < erro):
    print("\nSolução do sistema")
    for t in range(0 , m  , 1):
      print("%8.2f"%R[i2 , t], end=' ')
    pare = 1

  if(i1 == nloop):
    pare = 1
  i1 = i1 + 1
  i2 = i2 + 1
  k = k + 1
```

SAÍDA DO PROGRAMA

```
Gauss - Seidel - Sistema
Matriz Ampliada
  2.0000  -1.0000   2.0000   6.0000
  1.0000   2.0000   2.0000  11.0000
  1.0000  -1.0000   3.0000   8.0000

 k    x1       x2       x3       Erro
 0   0.0000   0.0000   0.0000  10.0000
 1   3.0000   4.0000   3.0000   4.0000
 2   2.0000   1.5000   2.5000   2.5000
 3   1.2500   2.3750   3.0417   0.8750
 4   1.1458   1.8854   2.9132   0.4896
 5   1.0295   2.0720   3.0142   0.1866
 6   1.0218   1.9749   2.9844   0.0971
 7   1.0031   2.0141   3.0037   0.0392
 8   1.0034   1.9946   2.9971   0.0195
 9   1.0002   2.0028   3.0009   0.0082
10   1.0005   1.9989   2.9994   0.0039
11   1.0000   2.0006   3.0002   0.0017

Solução do sistema
    1.00     2.00     3.00
```

ATIVIDADE

(01) Resolva o sistemas, com $x_0 = [0\ ,\ 0\ ,0]$, $E \leq 10^{-2}$ ou $k \leq 10$, onde k é o número de iterações. Utilize o método de Gauss Seidel.

(a) $\begin{cases} 2x - y + z = 2 \\ x + 2y + z = 4 \\ 2x + y + 2z = 5 \end{cases}$

(b) $\begin{cases} 4x - y + z = 5 \\ x + 2y + z = 5 \\ x - 3y + 3z = 4 \end{cases}$

(c) $\begin{cases} 3x - y - z = -2 \\ 2x + 5y + z = 15 \\ -x - y - 3z = -12 \end{cases}$

(d) $\begin{cases} 3x - y - z + t = 2 \\ 2x + 5y + z + t = 19 \\ -x - y - 3z - t = -16 \\ x + 2y + z + 5t = 28 \end{cases}$

(02) Resolva os sistemas, com $x_0 = [0\ ,0\ ,0]$, $E \leq 10^{-2}$ ou $k \leq 10$, onde k é o número de iterações. Utilize o método de Gauss Seidel.

(a) $\begin{cases} 3x + y - z = 2 \\ x + 5y + z = 14 \\ x - y - 3z = -10 \end{cases}$

(b) $\begin{cases} 3x - y + z = 4 \\ -x + 4y + z = 10 \\ -x - y + 3z = 6 \end{cases}$

(c) $\begin{cases} 3x + y - z = 2 \\ -x + 4y + 2z = 13 \\ -x + y + 2z = 7 \end{cases}$

(d) $\begin{cases} 3x + y - z + t = 6 \\ x + 5y + z + t = 18 \\ x - y - 3z + t = -6 \\ x + 2y + z + 5t = 28 \end{cases}$

5. AJUSTE DE CURVAS

Em muitas situações, principalmente as que estão relacionadas com levantamento de dados, conhecemos alguns valores da função, só nos pontos amostrados, e na maioria das vezes precisamos estimar o valor da função para pontos não amostrados. O exemplo a seguir apresenta um problema desta natureza.

Exemplo – Em uma cidade A foi feito um foi feito um censo cujos resultado está mostrado na tabela a seguir.

Ano	Número de habitantes
1940	19600
1960	19800
1980	20000
1990	20100
2000	20200

Tabela 1 – Resultado do censo feito na cidade hipotética A.

Quantos habitantes havia na cidade A em 1970? Para resolver este problema necessitamos estimar uma função que ajuste estes dados, e só então poderemos estimar o número de habitantes no ano em que se deseja.

5.1. AJUSTE LINEAR

Para calcularmos o número de habitantes no ano de 1970, devemos observar que os dados possuem um comportamento linear, como mostra a Figura 1, logo estes dados podem ser aproximados por uma reta da forma

$y = \alpha_0 + \alpha_1 x,$

onde α_0 e α_1 são denominados parâmetros do modelo.

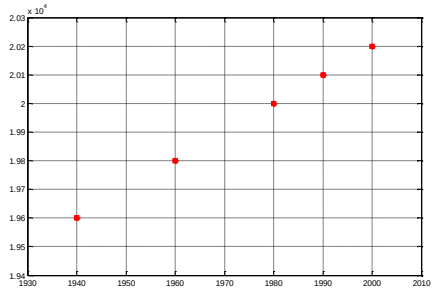


Figura 1 – Representação gráfica dos dados da Tabela 1.

Os valores de α_0 e α_1 que queremos estimar, para isto devemos fazer a seguinte consideração que é ilustrada com o gráfico da Figura 2.

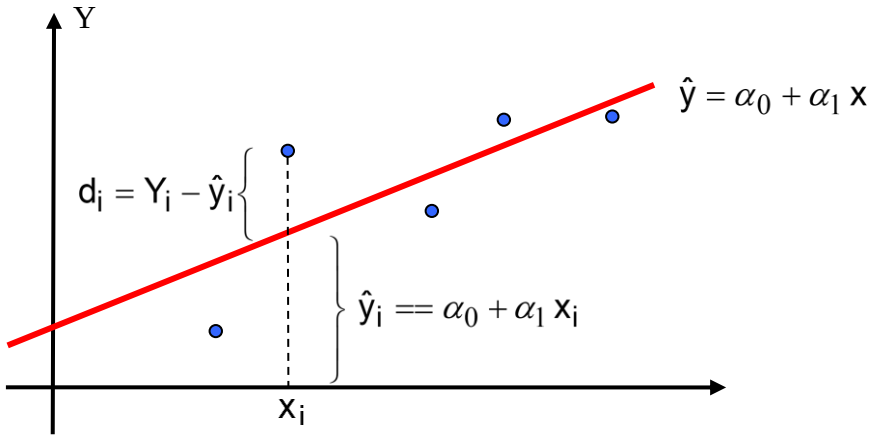


Figura 2 – As bolinhas representam os valores amostrados no campo e a reta representa a função ajustada nos pontos amostrados. No ponto x_i o valor y_i representa o valor amostrado, e $\hat{y}_i$ o seu valor estimado pela função ajustada e $d_i = y_i - \hat{y}_i$ é a diferença entre o valor amostrado (valor real do campo) e o valor estimado.

Como estimar a função $\hat{y} = \alpha_0 + \alpha_1 x$? Para estimarmos a função $\hat{y} = \alpha_0 + \alpha_1 x$, o erro entre o valor amostrado y_i e o valor estimado $\hat{y}_i$ deve ser mínimo, para isto a soma dos quadrados do erro de todos os pontos deve ser a menor possível.

Para você entender melhor, primeiro definiremos a função que representa a soma do quadrado dos erros:

$$D = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

onde temos n é o número de pontos amostrados. A magnitude de D depende da reta ajustada, ou seja, depende de α_0 e α_1. Assim como $\hat{y} = \alpha_0 + \alpha_1 x$, podemos escrever:

$$D(\alpha_0, \alpha_1) = \sum_{i=1}^{n} [y_i - (\alpha_0 + \alpha_1 x)]^2.$$

O mínimo de uma função de duas variáveis $D(\alpha_0, \alpha_1)$ ocorre quando as suas derivadas parciais $\frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_0}$ e $\frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_1}$ são simultaneamente iguais a zero.

Então para determinarmos α_0 e α_1 da função $\hat{y} = \alpha_0 + \alpha_1 x$, devemos fazer

$$\frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_0} = 0 \quad \text{e} \quad \frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_1} = 0,$$

O que resulta nas expressões:

$$\alpha_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \quad \text{e} \quad \alpha_0 = \frac{\sum_{i=1}^{n} y_i - \left(\sum_{i=1}^{n} x_i\right) \alpha_1}{n}.$$

Explicaremos o uso destas fórmulas através de um exemplo e você perceberá que sua aplicação é simples.

Exemplo: Encontre o número de habitantes de uma cidade no ano de 1970 considerando os dados do censo mostrado na Tabela 2.

i	Ano (x_i)	Número de habitantes (y_i)
1	1940	19600
2	1960	19800
3	1980	20000
4	1990	20100
5	2000	20200

Tabela 2 – Censo feito na cidade hipotética A. É o mesmo dado da Tabela 1.

Para calcularmos α_1 e α_0 devemos primeiro completar a tabela com as colunas contendo informação de x_i^2 e $x_i y_i$ (ver Tabela 3)

i	Ano (x_i)	Número de habitantes (y_i)	x_i^2	$x_i y_i$
1	1940	19600	3763600	38024000
2	1960	19800	3841600	38808000
3	1980	20000	3920400	39600000
4	1990	20100	3960100	39999000
5	2000	20200	4000000	40400000

Tabela 3 – Contém informações da Tabela 2 mais as colunas para x_i^2 e $x_i y_i$.

Agora calcularemos $\sum_{i=1}^{n} x_i$, $\sum_{i=1}^{n} y_i$, $\sum_{i=1}^{n} x_i^2$ e $\sum_{i=1}^{n} x_i y_i$ que são obtidos simplesmente pela soma dos elementos de cada coluna, como mostra a Tabela 4.

i	Ano (x_i)	Número de habitantes (y_i)	x_i^2	$x_i y_i$
1	1940	19600	3763600	38024000
2	1960	19800	3841600	38808000
3	1980	20000	3920400	39600000
4	1990	20100	3960100	39999000
5	2000	20200	4000000	40400000
	$\sum_{i=1}^{n} x_i = 9870$	$\sum_{i=1}^{n} y_i = 99700$	$\sum_{i=1}^{n} x_i^2 = 19485700$	$\sum_{i=1}^{n} x_i y_i = 196831000$

Tabela 4 – Estão os valores de x_i, y_i, x_i^2, $x_i y_i$, $\sum_{i=1}^{n} x_i$, $\sum_{i=1}^{n} y_i$, $\sum_{i=1}^{n} x_i^2$ e $\sum_{i=1}^{n} x_i y_i$.

Com os valores da Tabela 4 podemos calcular os coeficientes α_1 e α_0, da seguinte forma:

$$\alpha_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} = \frac{5 * 196831000 - 9870 * 99700}{5 * 19485700 - 196831000} = 10$$

$$\alpha_0 = \frac{\sum_{i=1}^{n} y_i - \left(\sum_{i=1}^{n} x_i\right)\alpha_1}{n} = \frac{99700 - (9870)10}{5} = 200$$

Com isto a função de ajuste é

$$\hat{y} = 200 + 10x$$

cujo gráfico está mostrado na Figura 3 juntamente com os pontos amostrados.

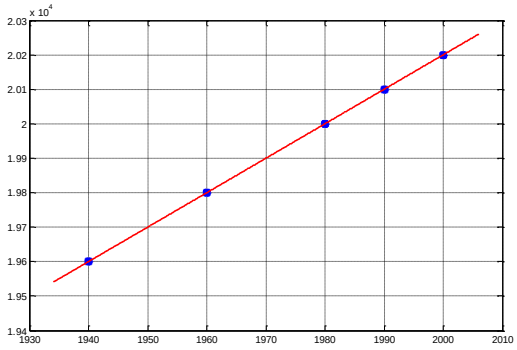


Figura 3 – A reta representa o gráfico da função ajustada $\hat{y} = 200 + 10x$ e os pontos os valores amostrados. Podemos perceber o bom ajuste da curva.

O número de habitantes em 1970 é obtido pela fórmula $\hat{y} = 200 + 10x$, da seguinte forma:

$\hat{y} = 200 + 10 * 1970 = 19900$, logo tivemos 19900 habitantes em 1970.

Exemplo:

Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 3$, segundo uma aproximação linear.

i	x_i	y_i
1	0.5000	2.5500
2	1.0000	4.5000
3	1.5000	5.6000
4	2.0000	7.0000
5	2.5000	9.2000
6	3.5000	11.0000
7	4.0000	13.0000

Solução

Devemos completar a tabela com os valores x_i^2 e $x_i y_i$

i	x_i	y_i	x_i^2	$x_i y_i$
1	0.5000	2.5500	0.2500	1.2750
2	1.0000	4.5000	1.0000	4.5000
3	1.5000	5.6000	2.2500	8.4000
4	2.0000	7.0000	4.0000	14.0000
5	2.5000	9.2000	6.2500	23.0000
6	3.5000	11.0000	12.2500	38.5000
7	4.0000	13.0000	16.0000	52.0000
	$\sum_{i=1}^{n} x_i = 15$	$\sum_{i=1}^{n} y_i = 52.8500$	$\sum_{i=1}^{n} x_i^2 = 42$	$\sum_{i=1}^{n} x_i y_i = 141.6750$

De onde temos

$$\alpha_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} = \frac{7 * 141.6750 - 15 * 52.8500}{7 * 42 - 141.6750} = 2.8837$$

$$\alpha_0 = \frac{\sum_{i=1}^{n} y_i - \left(\sum_{i=1}^{n} x_i\right)\alpha_1}{n} = \frac{52.8500 - (15)2.8837}{7} = 1.3707$$

Com isto a função de ajuste é

$\hat{y} = 1.3707 + 2.8837x$;

Logo quando $x = 3 \quad \Rightarrow \quad \hat{y} = 10.0217$

ATIVIDADE

(01) Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 0.5$, segundo uma aproximação linear.

i	x_i	y_i
1	0.0000	-0.2000
2	0.2000	0.8000
3	0.4000	1.8000
4	0.6000	2.8000
5	0.8000	3.8000
6	1.0000	4.8000
7	1.2000	5.8000

5.2. AJUSTE POLINOMIAL

O ajuste linear é um caso particular do ajuste polinomial, onde ajustaremos aos pontos amostrados um polinômio, $\hat{y}$, de grau n.

$\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \ldots + \alpha_n x^n$.

Os são coeficientes $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$ são obtidos através de um sistema da forma:

$$\underline{XA} = \underline{B}$$

Para que você entenda claramente a construção deste sistema iniciaremos abordando o ajuste linear segundo esta perspectiva.

Para ajustarmos uma reta (polinômio do 1º grau) $\hat{y} = \alpha_0 + \alpha_1 x$, devemos minimizar a função $D(\alpha_0, \alpha_1) = \sum_{i=1}^{n} [y_i - (\alpha_0 + \alpha_1 x)]^2$, para isto devemos fazer

$\frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_0} = 0 \quad \Rightarrow \quad n\alpha_0 + (\sum_{i=1}^{n} x_i)\alpha_1 = \sum_{i=1}^{n} y_i$

$\frac{\partial D(\alpha_0, \alpha_1)}{\partial \alpha_1} = 0 \quad \Rightarrow \quad \alpha_0 \sum_{i=1}^{n} x_i + \alpha_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i$

Podemos escrever este sistema na forma matricial

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$$

Comparando com o sistema $\underline{\underline{X}}\,\underline{\underline{A}} = \underline{\underline{B}}$, temos que:

$$\underline{X} = \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix}, \qquad \underline{A} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \quad \text{e} \quad \underline{B} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$$

Com a resolução do sistema, encontraremos $\underline{A}$ que possibilitará a determinação do polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x$.

Para entendermos como interpolar um polinômio de grau n, observe a tabela:

Polinômio a interpolador	Sistema a ser determinado
$\hat{y} = \alpha_0 + \alpha_1 x$	$\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$
$\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$	$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \end{bmatrix}$
$\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$	$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \sum_{i=1}^{n} x_i^6 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \\ \sum_{i=1}^{n} x_i^3 y_i \end{bmatrix}$

Seguindo o raciocínio da tabela, podemos afirmar que para ajustarmos o polinômio:

$$\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \ldots + \alpha_n x^n$$

Devemos resolver o sistema:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \cdots & \sum_{i=1}^{n} x_i^n \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \cdots & \sum_{i=1}^{n} x_i^{n+1} \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \cdots & \sum_{i=1}^{n} x_i^{n+2} \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \sum_{i=1}^{n} x_i^6 & \cdots & \sum_{i=1}^{n} x_i^{n+3} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \sum_{i=1}^{n} x_i^n & \sum_{i=1}^{n} x_i^{n+1} & \sum_{i=1}^{n} x_i^{n+2} & \sum_{i=1}^{n} x_i^{n+3} & \cdots & \sum_{i=1}^{n} x_i^{2n} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \\ \sum_{i=1}^{n} x_i^3 y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^n y_i \end{bmatrix}$$

Para você entender como montar este sistema, observe o próximo exemplo.

Exemplo: Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 3$, segundo o polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$.

i	x_i	y_i
1	0.5000	1.2500
2	1.0000	3.0000
3	1.5000	5.2500
4	2.0000	8.0000
5	2.5000	11.2500
6	3.5000	19.2500
7	4.0000	24.0000

Solução

Para montarmos o sistema devemos completar a tabela com as informações:

i	x_i	y_i	x_i^2	x_i^3	x_i^4	$x_i y_i$	$x_i^2 y_i$
1	0.5000	1.2500	0.2500	0.1250	0.0625	0.6250	0.3125
2	1.0000	3.0000	1.0000	1.0000	1.0000	3.0000	3.0000
3	1.5000	5.2500	2.2500	3.3750	5.0625	7.8750	11.8125
4	2.0000	8.0000	4.0000	8.0000	16.0000	16.0000	32.0000
5	2.5000	11.2500	6.2500	15.6250	39.0625	28.1250	70.3125
6	3.5000	19.2500	12.2500	42.8750	150.0625	67.3750	235.8125
7	4.0000	24.0000	16.0000	64.0000	256.0000	96.0000	384.0000
$\sum_{i=1}^{n}$	15	72	42	135	467.2500	219	737.2500

Desta forma o sistema para o ajuste do polinômio $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$, adquire a forma:

$$\begin{bmatrix} 7 & 15 & 42 \\ 15 & 42 & 135 \\ 42 & 135 & 467.2500 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 72 \\ 219 \\ 737.2500 \end{bmatrix}$$

De onde obtemos o seguinte polinômio $\hat{y} = 0 + 2x + x^2$, cujo gráfico está mostrado na Figura 4 juntamente com os pontos amostrado. Logo quando $x = 3 \Rightarrow \ \hat{y} = 15$.

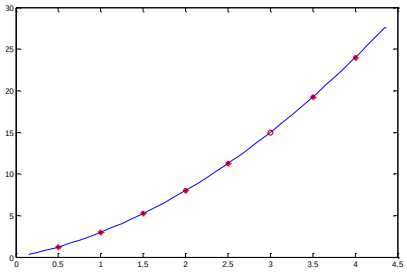


Figura 4 – Polinômio interpolador $\hat{y} = 0 + 2x + x^2$ e pontos amostrados.

PROGRAMA EM PYTHON

```
# Ajuste Linear

import numpy as np

# Entrada
x0 = 3

p = 2        # Grau do polinómio

print("Ajuste Linear")

D = np.array(
    [[ 0.5000   ,   1.2500],
     [ 1.0000   ,   3.0000],
     [ 1.5000   ,   5.2500],
     [ 2.0000   ,   8.0000],
     [ 2.5000   ,  11.2500],
     [ 3.5000   ,  19.2500],
     [ 4.0000   ,  24.0000]]
    )

md = np.size(D[:,1])
nd = np.size(D[1,:])
pp = 2*p

xi = np.zeros((md , pp+1))
ssxi = np.zeros((1 , pp+1))
sxi = np.zeros((1 , pp+2))
yi = np.zeros((md , pp+1))
syi = np.zeros((1 , pp+1))
M = np.zeros((p+1 , p+2))

xi[:,0] = D[:,0]
```

```
yi[:,0] = D[:,1]

print("    x         y")
for i in range(0 , md , 1):
  for j in range(0 , nd  , 1):
    print("%8.4f"%D[i,j], end=' ')
  print(" ")

print("\nf(%8.4f"%x0 , ") = ???")
print("Grau do polinômio: %d"%p + " grau")

for j in range(1 , pp , 1):
  for i in range(0 , md  , 1):
    xi[i , j]  = xi[i , 0]**(j+1)

s = 0
for j in range(0 , pp , 1):
  s = 0
  for i in range(0 , md  , 1):
    s = s + xi[i,j]
  ssxi[0,j] = s

sxi[0,0] = md
for j in range(0 , pp , 1):
  sxi[0,j+1] = ssxi[0,j]

#print("    x           x^2         x^3         x^4")
#for i in range(0 , md , 1):
  #for j in range(0 , pp  , 1):
    #print("%10.4f"%xi[i,j], end=' ')
  #print(" ")

#for j in range(0 , pp  , 1):
  #print("%10.4f"%ssxi[0,j], end=' ')
#print(" ")

#for j in range(0 , pp+1  , 1):
  #print("%10.4f"%sxi[0,j], end=' ')
#print(" ")


for j in range(1 , p+1 , 1):
  for i in range(0 , md  , 1):
    yi[i , j]  = (xi[i , 0]**(j))*yi[i , 0]

s = 0
for j in range(0 , p+1 , 1):
  s = 0
  for i in range(0 , md  , 1):
```

```
    s = s + yi[i,j]
  syi[0,j] = s

#print("    (x^0)y     (x^1)y     (x^2)y")
#for i in range(0 , md , 1):
  #for j in range(0 , p+1  , 1):
    #print("%10.4f"%yi[i,j], end=' ')
  #print(" ")

#for j in range(0 , p+1  , 1):
  #print("%10.4f"%syi[0,j], end=' ')
#print(" ")


k = 0
for i in range(0 , p+1 , 1):
  for j in range(0 , p+3 , 1):
    if(j+k <= p+2):
      a = sxi[0,j+k]
    if(j+k > p+2):
      a = 0
    if(j < p+1):
      M[i,j] = a
  k = k + 1

for i in range(0 , p+1 , 1):
  M[i,p+1] = syi[0,i]


print("sistema")
for i in range(0 , p+1 , 1):
  for j in range(0 , p+2  , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")

linha = np.size(M[:,1])
coluna = np.size(M[1,:])

#print('Linha: {:d}'.format(linha))
#print('Coluna: {:d}'.format(coluna))

# Zera Triangulo Infrerior
t = 1
fm = 0

for j in range(0 , linha , 1):
  for i in range(t , linha , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
```

```
      M[i , k] = fm * M[j , k] + M[i , k]
  t = t + 1

#print("Zera Triangulo Infrerior")
#for i in range(0 , linha , 1):
  #for j in range(0 , coluna  , 1):
    #print("%8.4f"%M[i,j], end=' ')
  #print(" ")

t = 1
fm = 0

for j in range(linha - 1 , 0 , -1):
  for i in range(0 , linha - t , 1):
    fm = - M[i,j]/M[j,j]
    for k in range(0 , coluna , 1):
      M[i,k] = fm * M[j,k] + M[i,k]
  t = t + 1

#print("Zera Triangulo Superior")
#for i in range(0 , linha , 1):
  #for j in range(0 , coluna  , 1):
    #print("%8.4f"%M[i,j], end=' ')
  #print(" ")


fm = 0

for i in range(0 , linha , 1):
  fm = M[i,i]
  for j in range(0 , coluna , 1):
    M[i,j] = M[i,j]/fm

#print("Matriz Normalizada")
#for i in range(0 , linha , 1):
  #for j in range(0 , coluna  , 1):
    #print("%8.4f"%M[i,j], end=' ')
  #print(" ")

print("\nSolução")
print("f(x)= a0 + a1 x + a2 x^2 + a3 x^3 + a4 x^4")
for i in range(0 , linha , 1):
  print("a%d"%i , "= %8.4f"%M[i , coluna-1])

fx0 = 0
for i in range(0 , linha , 1):
  fx0 = fx0 + M[i , coluna-1] * x0**i

print(" ")
```

```
print("f(%8.4f"%x0 , ") = %8.4f"%fx0)
print(" ")
print(" ")

import matplotlib.pyplot as plt
import numpy as np
xii = np.linspace(-10, 10, 100)
nd2 = np.size(xii)
fxii = np.zeros((1 , nd2))


for j in range(0 , nd2 , 1):
  fxii[0,j] = 0
  for i in range(0 , linha  , 1):
    fxii[0,j] = fxii[0,j] + M[i , coluna-1] * xii[j]**i

fig = plt.figure()
plt.plot(D[:,0], D[:,1], '*')
plt.plot(xii[:], fxii[0,:])
plt.grid()
```

SAÍDA DO PROGRAMA

```
Ajuste Linear
    x        y
  0.5000   1.2500
  1.0000   3.0000
  1.5000   5.2500
  2.0000   8.0000
  2.5000  11.2500
  3.5000  19.2500
  4.0000  24.0000

f(  3.0000 ) = ???
Grau do polinômio: 2 grau
sistema
  7.0000  15.0000  42.0000  72.0000
 15.0000  42.0000 135.0000 219.0000
 42.0000 135.0000 467.2500 737.2500

Solução
f(x)= a0 + a1 x + a2 x^2 + a3 x^3 + a4 x^4
a0 =   0.0000
a1 =   2.0000
a2 =   1.0000

f(  3.0000 ) =  15.0000
```

ATIVIDADE

(01) Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 3$, segundo o polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$.

i	x_i	y_i
1	0.5000	0.7500
2	1.0000	2.0000
3	1.5000	3.7500
4	2.0000	6.0000
5	2.5000	8.7500
6	3.5000	15.7500
7	4.0000	20.0000

(02) Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 0.3$, segundo o polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$.

i	x_i	y_i
1	0.0000	0.0000
2	0.2000	-0.1600
3	0.4000	-0.2400
4	0.6000	-0.2400
5	0.8000	-0.1600
6	1.0000	0.0000
7	1.2000	0.2400

(03) Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 0.7$, segundo o polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2$.

i	x_i	y_i
1	0.0000	0.0000
2	0.2000	0.1200
3	0.4000	0.0800
4	0.6000	-0.1200
5	0.8000	-0.4800
6	1.0000	-1.0000
7	1.2000	-1.6800

(04) Com base dos dados amostrados dispostos na tabela a seguir encontre o valor quando $x = 0.5$, segundo o polinômio interpolador $\hat{y} = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$.

i	x_i	y_i
1	0.0000	0.0000
2	0.2000	0.2320
3	0.4000	0.4960
4	0.6000	0.7440
5	0.8000	0.9280
6	1.0000	1.0000
7	1.2000	0.9120

6. INTERPOLAÇÃO

Em um dado experimento foram feitas 4 amostras, cujos valores estão dispostos na tabela abaixo e cuja representação gráfica está na figura a seguir.

Tabela 1

Número da amostra (i)	x_i	y_i
1	0.5	0.25
2	1.2	1.44
3	3.0	9.00
4	4.5	20.25

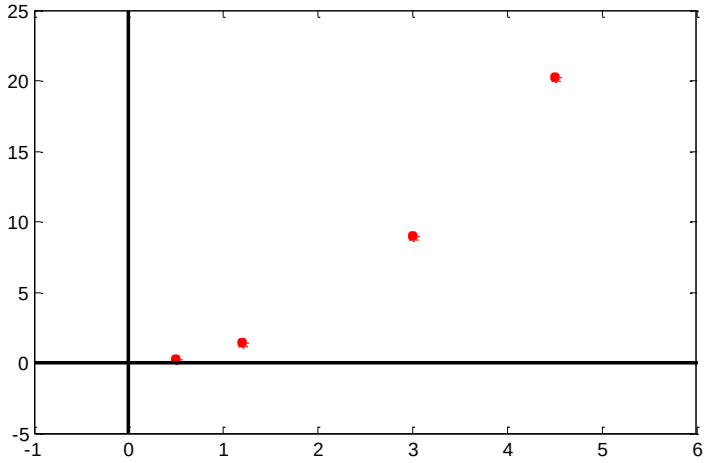


Figura 1. Representação gráfica dos dados da Tabela 1.

Quanto vale y_i quando $x_i = 2$?

Semelhante a este problema, onde $y = f(x)$, existem muitos outros, onde muitas funções são conhecidas apenas em um conjunto finito e discreto de pontos de um intervalo $[a, b]$.

Nestes casos, onde não se tem a forma analítica da função $y = f(x)$, devemos substituí-la por outra função $g(x)$, que é uma aproximação da função $y = f(x)$ e que é deduzida a partir de dados da tabelados.

Para determinarmos o valor de y_i quando $x_i = 2$, iremos determinar a função $g(x) = ax^2 + bx + c$, que é denominada de polinômio interpolador. Para os dados da Tabela 1, obteremos a função $g(x) = 1x^2 + 0x + 0$, cujo gráfico está mostrado na figura a seguir juntamente com os dados da tabela 1.

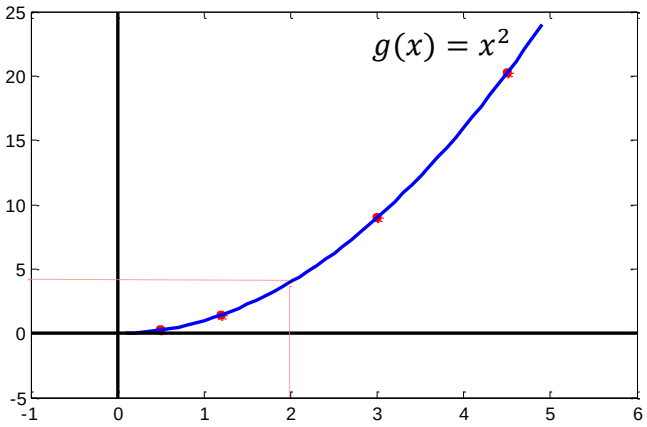


Figura 2. Gráfico da função $g(x) = 1x^2 + 0x + 0$ juntamente os pontos da tabela 1.

Substituindo $x_i = 2$ no polinômio interpolador obtemos o valor de y_i quando $x_i = 2$.

$$g(2) = 1 \cdot 2^2 + 0 \cdot 2 + 0 = 4$$

Este tipo de solução, também é utilizado quando se têm funções cuja forma analítica é complicada e de difícil manuseio, nestes casos, devemos substituir a expressão analítica por outra mais simples.

Como foi obtido o polinômio interpolador? Iremos explicar detalhadamente como calcular o polinômio interpolador, mas primeiro devemos definir o que é interpolação.

CONCEITO DE INTERPOLAÇÃO

Seja a função $y = f(x)$, cujos valores estão em uma tabela. Se desejarmos determinar $f(\bar{x})$ sendo:

(a) $\bar{x} \in (x_0, x_n)$ e $\bar{x} \neq x_i$ onde $i = 0, 1, 2, \ldots, n$

(b) $\bar{x} \notin (x_0, x_n)$

O item (a) representa um problema de interpolação, isto é, $\bar{x}$ está dentro do intervalo amostrado, logo devemos calcular um polinômio interpolador, que é uma aproximação da função tabelada.

O item (b) representa um problema de extrapolação, isto é, $\bar{x}$ está fora do intervalo amostrado. Nos trataremos apenas de problemas de interpolação neste capítulo.

6.1. INTERPOLAÇÃO LINEAR

Para que você entenda claramente o que é interpolação, explicaremos interpolação linear através de um exemplo prático ilustrado a seguir.

(01) Na tabela está a produção seguir está assinalado o número de habitantes de uma cidade A em quatro censos.

Tabela 1

ANO	1950	1960
Nº de Habitantes	352.724	683.908

Determinar o número aproximado de habitantes na cidade A em 1955. O grau do polinômio interpolador é uma unidade menor que o número de pontos conhecidos

Solução

Neste caso, o polinômio interpolador terá grau 1, isto é, será da forma

$$P_1(x) = a_1x + a_0$$

Para se determinar os coeficientes, a_0 e a_1 devemos fazer

$$\begin{cases} P_1(x_0) = a_1x_0 + a_0 = y_0 \\ P_1(x_1) = a_1x_1 + a_0 = y_1 \end{cases} \Rightarrow \quad \begin{cases} a_1x_0 + a_0 = y_0 \\ a_1x_1 + a_0 = y_1 \end{cases}$$

Para $x_0 = 1950$ e $y_0 = 352.724$ temos que

$$a_1 1950 + a_0 = 352.724$$

Para $x_1 = 1960$ e $y_1 = 683.908$ temos que

$$a_1 1960 + a_0 = 683.908$$

Com isto temos o seguinte sistema

$$\begin{cases} a_1 1950 + a_0 = 352.724 \\ a_1 1960 + a_0 = 683.908 \end{cases}$$

onde $a_1 = 33118{,}40$ e $a_0 = -64228156$ logo teremos

$$P_1(x) = 33118{,}40x - 64228156$$

como queremos saber o número aproximado de habitantes na cidade A em $x = 1955$, temos

$P_1(x) = 33118{,}40 \cdot 1955 - 64228156 = 518.316$habitantes

ATIVIDADE

Obs. Utilize o programa de ajuste linear e e faça a interpolação com o grau máximo que os dados permitem.

(01) Na tabela a seguir está a produção de uma certa indústria.

ANO	1990	2000
Nº de peças	340	680

Determinar o número de peças no ano 1995.

(02) Com base na tabela a seguir encontre o valor de y para $x = 7$.

X	2	10
Y	9	25

(03) Com base na tabela a seguir encontre o valor de y para $x = 5$.

X	2	8
Y	2	20

(04) Com base na tabela a seguir encontre o valor de y para $x = 7$.

X	2	5	9
Y	-2	7	47

ERRO DE TRUNCAMENTO

Para que você entenda o erro de truncamento, observe o gráfico mostrado a figura a seguir.

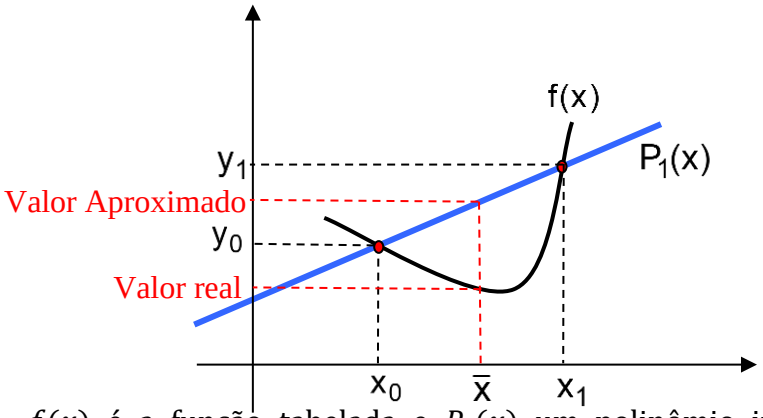


Figura - $f(x)$ é a função tabelada e $P_1(x)$ um polinômio interpolador de 1º grau. Podemos observar que, neste caso, $P_1(x)$ não aproxima bem a solução.

Teoricamente o erro de truncamento cometido no ponto $\bar{x}$ é dado pela fórmula

$E_T(x) = (x - x_0)(x - x_1) \cdot A,$

onde A é uma constante a determinar, como a função erro de truncamento.

No cálculo de A, utilizaremos a função auxiliar $G(t)$ definida por:

$G(t) = f(t) - P_1(t) - E_T(t)$.

Para que você tenha melhor entendimento, calcularemos o erro de polinômio interpolador do primeiro grau, onde

$P_1(t) = a_1 t + a_0$ e $E_T(t) = (t - x_0)(t - x_1) \cdot A$,

substituindo, obteremos:

$G(t) = f(t) - P_1(t) - (t - x_0)(t - x_1) \cdot A$,

onde a função $G(t)$ se anula em pelo menos em três pontos $t = x_0$, $t = x_1$ e $t = \bar{x}$.

TEOREMA DE ROLLE

Se a função $f(x)$ é contínua no intervalo $[a\,, b]$ e diferenciável no intervalo $(a\,, b)$ e $f(a) = f(b)$, então, existe um $\xi \in (a, b)$, tal que $f'(\xi) = 0$

Considerações:

Se $f(t)$ é contínua em $[x_0, x_1]$ e diferenciável em $(x_0\,, x_1)$, pode-se concluir que $G(t)$ também o é, tendo em vista que $P_1(t)$ e $E_T(t)$ são funções polinomiais de 1º e 2º graus, respectivamente, logo

Existe $\xi_1 \in (x_0, \bar{x})$ tal que $G(\xi_1) = 0$ e

Existe $\xi_2 \in (\bar{x}, x_1)$ tal que $G(\xi_2) = 0$

Aplicando o teorema de Rolle na função $G'(t)$, teremos:

Existe $\xi \in (\xi_1, \xi_2)$ e portanto $\xi \in (x_0, x_1)$, tal que $G''(t) = 0$, logo teremos

$G''(\xi) = f''(\xi) - 2A = 0 \;\Rightarrow\; A = \frac{f''(\xi)}{2}$

de onde obteres a expressão para o cálculo do erro de truncamento

$$E_T(t) = (x - x_0)(x - x_1) \cdot \frac{f''(\xi)}{2}$$

para algum $\xi \in (x_0, x_1)$.

Exemplo 1. Seja a função $f(x) = senx$. Determine:

(a) O valor aproximado para $f\left(\frac{\pi}{2}\right)$ a partir dos pontos (1,0 ; 0,84) e (2,0 ; 0,91).

(b) O erro de truncamento cometido no cálculo do item anterior.

Solução

(a) Para determinarmos $f\left(\frac{\pi}{2}\right)$ devemos primeiro calcular o polinômio interpolador $P_1(x) = a_1 x + a_0$

Para $x_0 = 1{,}0$ e $y_0 = 0{,}84$ temos que

$$a_1 1{,}0 + a_0 = 0{,}84$$

Para $x_1 = 2{,}0$ e $y_1 = 0{,}91$ temos que

$$a_1 2{,}0 + a_0 = 0{,}91$$

que resulta no sistema:

$\begin{cases} a_1 + a_0 = 0{,}84 \\ 2a_1 + a_0 = 0{,}91 \end{cases}$ cuja solução é $\begin{cases} a_1 = 0{,}07 \\ a_0 = 0{,}77 \end{cases}$

então teremos

$$P_1(x) = 0{,}07x + 0{,}77 \quad \Rightarrow \quad P_1\left(\frac{\pi}{2}\right) = 0{,}07\left(\frac{\pi}{2}\right) + 0{,}77 = 0{,}88$$

(b) Para determinarmos o erro de truncamento devemos calcular a 1º e a 2º derivada da função $f(x)$

$$f(x) = senx \quad \Rightarrow \quad f'(x) = \cos x \quad \Rightarrow \quad f''(x) = -senx$$

$$|E_T(\xi)| \leq \left|(\xi - x_0)(\xi - x_1) \cdot \frac{f''(\xi)}{2}\right|$$

$$\left|E_T\left(\frac{\pi}{2}\right)\right| \leq \left|\left(\frac{\pi}{2} - 1\right)\left(\frac{\pi}{2} - 2\right) \cdot \frac{(-1)}{2}\right|$$

ou seja

$$\left|E_T\left(\frac{\pi}{2}\right)\right| \leq 0{,}12 \quad \text{ou} \quad -0{,}12 \leq E_T\left(\frac{\pi}{2}\right) \leq 0{,}12$$

6.2. INTERPOLAÇÃO DE LAGRANGE

As interpolações apresentadas anteriormente (interpolação linear) são casos particulares da interpolação de Lagrange. Agora vamos determinar outra forma de se obter o polinômio interpolador $P(x)$ de grau menor ou igual a n, sendo dado para isto, $n+1$ pontos distintos.

Para podemos ter uma boa compreensão da interpolação de Lagrange, temos que primeiro entender o teorema apresentado a seguir.

Teorema

Sejam (x_i, y_i), $i = 0,1,2,\ldots,n,n+1$ pontos distintos, isto é, $x_i \neq x_j$ para $i \neq j$. Existe um único polinômio $P(x)$ de grau não maior que n, tal que $p(x_i) = y_i$, para todo i. O polinômio $P(x)$ pode ser escrito na forma:

$$P_n(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_nx^n$$

ou da seguinte forma

$$P_n(x) = \sum_{i=0}^{n} a_i x^i$$

Observe que $P(x)$ é, no máximo, de grau n, se $a_n \neq 0$. Para determinar o polinômio $P(x)$ devemos conhecer os valores $a_0, a_1, a_2, \ldots, a_n$. Como $P(x)$ contém os pontos (x_i, y_i) podemos escrever $p(x_i) = y_i$, da seguinte forma

$$\text{S: } \begin{cases} a_0 + a_1x_0 + a_2x_0^2 + a_3x_0^3 + \ldots + a_nx_0^n = y_0 \\ a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3 + \ldots + a_nx_1^n = y_1 \\ a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3 + \ldots + a_nx_2^n = y_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_0 + a_1x_n + a_2x_n^2 + a_3x_n^3 + \ldots + a_nx_n^n = y_n \end{cases}$$

A solução do sistema S são os valores $a_0, a_1, a_2, \ldots, a_n$, com os quais determinamos o polinômio $P_n(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_nx^n$.

O polinômio $P(x)$ é único? Para verificarmos que tal polinômio é único, basta calcularmos o determinante da matriz $\underline{A}$ (matriz dos coeficientes) e verificar que ele é diferente de zero.

$$\underline{A} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^2 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^2 \end{bmatrix}$$

Observe que a matriz $\underline{A}$, tem a forma da matriz de Vandermonte, também conhecida como matriz das potências. Seu determinante, segundo a Álgebra Linear, é dado pela expressão:

$det(\underline{A}) = \prod_{i>j}(x_i - x_j), \quad \text{com } x_i \neq x_j$

Sabemos que $det(\underline{A}) \neq 0$, logo isto prova que $P(x)$ é único.

Obtenção da Fórmula

Para que você entenda a interpolação de Lagrange é necessário que compreender como é obtida a fórmula de recorrência deste método.

O teorema fundamental da Álgebra garante que podemos escrever o polinômio $P(x)$ da seguinte forma

$$P(x) = (x - x_0)(x - x_1)(x - x_2)(x - x_3)\dots(x - x_n)$$

onde $x_0, x_1, x_2, x_3, \dots, x_n$ são as raízes do polinômio $P(x)$. Montaremos agora, uma sequência de polinômios auxiliares da seguinte forma

1º polinômio: se retirarmos $(x - x_0)$ obteremos o polinômio

$p_0(x) = (x - x_1)(x - x_2)(x - x_3)\dots(x - x_n)$

2º polinômio: se retirarmos $(x - x_1)$ obteremos o polinômio

$p_1(x) = (x - x_0)(x - x_2)(x - x_3)\dots(x - x_n)$

3º polinômio: se retirarmos $(x - x_2)$ obteremos o polinômio

$p_2(x) = (x - x_0)(x - x_1)(x - x_3)\dots(x - x_n)$

Seguindo este raciocínio obteremos os polinômios $p_0(x), p_1(x), p_2(x), \dots, p_n(x)$. Estes polinômios podem ser escritos na forma sintética:

$p_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n}(x - x_j), \quad (i = 0,1,2,3,\dots,n)$

Tais polinômios possuem as seguintes propriedades

(a) $p_i(x_i) \neq 0,$ para todo *i.*

(b) $p_i(x_j) = 0,$ para todo $j \neq i$.

e são conhecidos como polinômios de Lagrange. O polinômio $P(x)$ pode ser escrito como uma combinação linear dos polinômios $p_0(x), p_1(x), p_2(x), \ldots, p_n(x)$, da seguinte forma:

$$P(x) = b_0 p_0(x) + b_1 p_1(x) + b_2 p_2(x) + \ldots + b_n p_n(x)$$

ou

$$P(x) = \sum_{i=0}^{n} b_i p_i(x)$$

Mas, como $p_i(x_j) = 0,$ para todo $j \neq i$ e $p_i(x_i) \neq 0,$ para todo i, temos que

$$P_n(x_n) = b_n p_n(x_n)$$

logo $b_n = \frac{P_n(x_n)}{p_n(x_n)}$

e como $P_n(x_i) = y_i$, teremos $b_i = \frac{y_i}{p_i(x_i)}$

substituindo este valor no somatório será

$$P(x) = \sum_{i=0}^{n} \frac{y_i}{p_i(x_i)} p_i(x)$$

de onde teremos $P(x) = \sum_{i=0}^{n} y_i \frac{p_i(x)}{p_i(x_i)}$

como $p_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} (x - x_j)$ então

$$P(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

denominada de fórmula de interpolação de Lagrange.

Calma! Parece complicado, mas garantimos que não é! Acompanhe cuidadosamente o próximo exemplo e você certamente entenderá como interpolar com o método de Lagrange.

Exemplo 1. A partir das informações existentes na tabela, determine:

i	x_i	y_i
0	0.0	0.000
1	0.2	2.008
2	0.4	4.064
3	0.5	5.125

(a) O polinômio interpolador de Lagrange

(b) $P(0.3)$

Solução

(a) Como temos 4 pontos, o polinômio interpolador será de grau 3, logo

$P_3(x) = \sum_{i=0}^{3} y_i \prod_{\substack{j=0 \\ j\neq i}}^{3} \frac{(x-x_j)}{(x_i-x_j)}$, ou seja

$$P_3(x) = y_0 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} +$$
$$+y_1 \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} +$$
$$+y_2 \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} +$$
$$+y_3 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$$

substituindo os valores da tabela, teremos

$$P_3(x) = 0.000 \frac{(x-0.2)(x-0.4)(x-0.5)}{(0.0-0.2)(0.0-0.4)(0.0-0.5)} +$$
$$+2.008 \frac{(x-0.0)(x-0.4)(x-0.5)}{(0.2-0.0)(0.2-0.4)(0.2-0.5)} +$$
$$+4.064 \frac{(x-0.0)(x-0.2)(x-0.5)}{(0.4-0.0)(0.4-0.2)(0.4-0.5)} +$$
$$+5.125 \frac{(x-0.0)(x-0.2)(x-0.4)}{(0.5-0.0)(0.5-0.2)(0.5-0.4)}$$

simplificando a expressão, temos o seguinte polinômio interpolador

$$P_3(x) = x^3 + 10x$$

(b) $P_3(0.3) = 0.3^3 + 10 \cdot 0.3 = 3.027$

Exemplo 2. A partir das informações existentes na tabela, determine:

I	x_i	y_i
0	1	1
1	2	4
2	4	16

(a) O polinômio interpolador de Lagrange

(b) $P(3)$

Solução

(a) Como temos 3 pontos, o polinômio interpolador será de grau 2, logo

$P_2(x) = \sum_{i=0}^{2} y_i \prod_{\substack{j=0 \\ j\neq i}}^{2} \frac{(x-x_j)}{(x_i-x_j)}$, ou seja

$$P_2(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

substituindo os valores da tabela, teremos

$$P_2(x) = 1\frac{(x-2)(x-4)}{(1-2)(1-4)} + 4\frac{(x-1)(x-4)}{(2-1)(2-4)} + 16\frac{(x-1)(x-2)}{(4-1)(4-2)}$$

que é o polinômio interpolador

(b) $P_2(3) = 1\frac{(3-2)(3-4)}{(1-2)(1-4)} + 4\frac{(3-1)(3-4)}{(2-1)(2-4)} + 16\frac{(3-1)(3-2)}{(4-1)(4-2)}$

$P_2(3) = 9$

PROGRAMA EM PYTHON

```
# Interpolação de Lagrange

import numpy as np

# Entrada
x0 = 0.3

D = np.array(
    [[0.0   ,  0.000],
     [0.2   ,  2.008],
     [0.4   ,  4.064],
     [0.5   ,  5.125]]
    )

print("Interpolação de Lagrange")
print(D)
#print(D[0,:])

# Variáveis auxiliares
s = 0
p = 1

# matriz linha X coluna
linha = np.size(D[:,1])
coluna = np.size(D[1,:])

#print("[linha, coluna] = " + format([linha, coluna]))

s = 0

for i in range(linha):
  p = 1;
  for j  in range(linha):
    if(i == j):
      p = p
    if(i != j):
      p = p * ((x0 - D[j,0])/(D[i,0] - D[j,0]));
  s = s + D[i, 1] * p
```

```
print("\n interpolacao em x: {:.3f}" .format(x0))
print(" y interpolado: {:.3f}" .format(s))
```

SAÍDA DO PROGRAMA

```
Interpolação de Lagrange
[[0.    0.   ]
 [0.2   2.008]
 [0.4   4.064]
 [0.5   5.125]]

 interpolacao em x: 0.300
 y interpolado: 3.027
```

ATIVIDADE

(01) A partir das informações existentes na tabela, determine:

i	x_i	y_i
0	0.0	0.0000
1	0.2	1.0400
2	0.4	2.1600
3	0.6	3.3600

(a) O polinômio interpolador de Lagrange (b) $P(0.3)$

(02) A partir das informações existentes na tabela, determine:

i	x_i	y_i
0	0.1	0.1010
1	0.3	0.3270
2	0.5	0.6250
3	0.7	1.0430

(a) O polinômio interpolador de Lagrange

(b) $P(0.4)$

(03) A partir das informações existentes na tabela, determine:

i	x_i	y_i
0	0.0	0.0000
1	0.2	0.4080
2	0.4	0.8640
3	0.6	1.4160

(a) O polinômio interpolador de Lagrange

(b) $P(0.5)$

(04) A partir das informações existentes na tabela, determine:

I	x_i	y_i
0	0.1	0.0110
1	0.3	0.1170
2	0.5	0.3750
3	0.7	0.8330

(a) O polinômio interpolador de Lagrange (b) $P(0.6)$

6.3. INTERPOLAÇÃO DE NEWTON

Para que você tenha uma boa compreensão do método de interpolação de Newton com diferenças divididas, iniciaremos este tópico, discorrendo sobre o conceito de diferenças divididas.

CONCEITO DE DIFERENÇAS DIVIDIDAS

Seja $y = f(x)$ uma função que contém n pontos distintos (x_i, y_i), onde $i = 0,1,2,\ldots,n$. Representaremos diferença divididas, por $f[]$. Definiremos diferença dividida de ordem zero a própria função, isto é,

$f^0[x_1] = f(x_1) = y_1$.

A diferença dividida de 1ª ordem para os argumentos x_0 e x_1 é uma aproximação da 1ª derivada, isto é,

$f^1[x_0, x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0}$,

onde temos a seguinte propriedade $f[x_1, x_0] = f[x_0, x_1]$. Considerando $y_i = f(x_i)$, podemos escrever as diferenças divididas de 1º ordem, de forma geral, por:

$f^1[x_i, x_{i+1}] = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$.

A diferença dividida de 2ª ordem para os argumentos x_0, x_1 e x_2 é dada por:

$f^2[x_0, x_1, x_2] = \frac{f^1[x_1,x_2]-f^1[x_0,x_1]}{x_2-x_0}$.

A diferença dividida de 3ª ordem para os argumentos x_0, x_1, x_2 e x_3 é dada por:

$f^3[x_0, x_1, x_2, x_3] = \frac{f^2[x_1,x_2,x_3]-f^2[x_0,x_1,x_2]}{x_3-x_0}$.

Genericamente, a diferença dividida de ordem n é dada por:

$f^n[x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+n}] = \frac{f^{n-1}[x_{i+1},x_{i+2},\ldots,x_{i+n}]-f^{n-1}[x_i,x_{i+1},x_{i+2},\ldots,x_{i+n-1}]}{x_{i+n}-x_i}$.

Para que você compreenda melhor como fazer estas diferenças dividida observe o próximo exemplo numérico.

Exemplo 1. Dada a função tabelada calcule a diferença dividida de segunda ordem.

i	x_i	y_i
0	0.3	3.09
1	1.5	17.25
2	2.1	25.41

Solução

Devemos calcular as diferenças divididas de primeira ordem

$f^1[x_0, x_1] = \frac{y_1-y_0}{x_1-x_0} = \frac{17.25-3.09}{1.5-0.3} = 11.80$

$f^1[x_1, x_2] = \frac{y_2 - y_1}{x_2 - x_1} = \frac{25.41 - 17.25}{2.1 - 1.5} = 13.60$

com todas as diferenças divididas de primeira ordem calculadas, vamos então calcular a de segunda ordem

$f^2[x_0, x_1, x_2] = \frac{f^1[x_1, x_2] - f^1[x_0, x_1]}{x_2 - x_0} = \frac{13.60 - 11.80}{2.1 - 0.3} = 1.0$

Para facilitar os procedimentos numéricos e organizar os nossos cálculos colocaremos na própria tabela o desenvolvimento do cálculo da seguinte forma:

i	x_i	y_i	$f^1[x_i, x_{i+1}]$	$f^2[x_0, x_1, x_2]$
0	0.3	3.09	$f^1[x_0, x_1]$	$f^2[x_0, x_1, x_2]$
1	1.5	17.25	$f^1[x_1, x_2]$	
2	2.1	25.41		

Fazendo a substituição numérica temos:

i	x_i	y_i	$f^1[x_i, x_{i+1}]$	$f^2[x_0, x_1, x_2]$
0	0.3	3.09	11.80	1.00
1	1.5	17.25	13.60	
2	2.1	25.41		

Agora que já sabemos como calcular as diferenças divididas, iremos nos concentrar na fórmula de recorrência para interpolação de Newton.

A fórmula de recorrência de interpola, de Newton com diferenças dividida, depende do número de pontos existente na tabela.

1º Caso: Existem só dois pontos na tabela

A fórmula, de interpolação, é obtida a partir da expressão de diferença divididas de primeira ordem,

$$f^1[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_0) - f(x_1)}{x_0 - x_1}$$

onde isolando $f(x)$, para obter a fórmula de interpolação:

$$f(x_0) = f(x_1) + (x_0 - x_1) f^1[x_0, x_1]$$

assumiremos $x = x_0$, onde x é qualquer valor dentro do intervalo $[x_0, x_1]$.

2º Caso: Existem só três pontos na tabela

A fórmula de interpolação, neste caso, é obtida a partir da expressão de diferença divididas de segunda ordem,

$f^2[x_0, x_1, x_2] = \frac{f^1[x_1, x_2] - f^1[x_0, x_1]}{x_2 - x_0} = \frac{f^1[x_0, x_1] - f^1[x_1, x_2]}{x_0 - x_2}$

onde isolando $f^1[x_1, x_2]$, obtemos:

$f^1[x_0, x_1] = f^1[x_1, x_2] + (x_0 - x_2)f^2[x_0, x_1, x_2]$

Substituindo na primeira fórmula de interpolação, temos

$f(x_0) = f(x_1) + (x_0 - x_1)\{f^1[x_1, x_2] + (x_0 - x_2)f^2[x_0, x_1, x_2]\}$

que pode ser escrita por

$$f(x_0) = f(x_1) + (x_0 - x_1)f^1[x_1, x_2] + (x_0 - x_1)(x_0 - x_2)f^2[x_0, x_1, x_2]$$

que é a fórmula de interpolação para este caso, onde assumiremos $x = x_0$, onde x é qualquer valor dentro do intervalo $[x_0, x_2]$.

3º Caso: Existem só quatro pontos na tabela

A fórmula de interpolação, neste caso, é obtida a partir da expressão de diferença divididas de terceira ordem,

$$f^3[x_0, x_1, x_2, x_3] = \frac{f^2[x_1, x_2, x_3] - f^2[x_0, x_1, x_2]}{x_3 - x_0} = \frac{f^2[x_0, x_1, x_2] - f^2[x_1, x_2, x_3]}{x_0 - x_3}$$

onde isolamos $f^2[x_0, x_1, x_2]$, para obter:

$$f^2[x_0, x_1, x_2] = f^2[x_1, x_2, x_3] + (x_0 - x_3)f^3[x_0, x_1, x_2, x_3]$$

Substituindo na segunda fórmula de interpolação, temos

$$f(x_0) = f(x_1) + (x_0 - x_1)f^1[x_1, x_2] + (x_0 - x_1)(x_0 - x_2)\{f^2[x_1, x_2, x_3] + (x_0 - x_3)f^3[x_0, x_1, x_2, x_3]\}$$

que pode ser expresso por:

$$f(x_0) = f(x_1) + (x_0 - x_1)f^1[x_1, x_2] + (x_0 - x_1)(x_0 - x_2)f^2[x_1, x_2, x_3] + (x_0 - x_1)(x_0 - x_2)(x_0 - x_3)f^3[x_0, x_1, x_2, x_3]$$

que é a fórmula de interpolação para este caso, onde assumiremos $x = x_0$, onde x é qualquer valor dentro do intervalo $[x_0, x_3]$.

4º Caso: Generalização para n pontos na tabela

Para uma tabela de n pontos, a fórmula de interpolação pode ser expressa, segundo o mesmo raciocínio, por:

$$f(x_0) = f(x_1) + \sum_{i=0}^{n} f^i[x_0, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j)$$

onde assumiremos $x = x_0$, onde x é qualquer valor dentro do intervalo $[x_0, x_n]$.

Exemplo 1. Determinar o valor aproximado de $f(0.4)$, usando todos os pontos tabelados

I	x_i	y_i
0	0.0	1.008
1	0.2	1.064
2	0.3	1.125
3	0.5	1.343
4	0.6	1.512

Solução

i	x_i	$y_i = f[]$	$f^1[]$	$f^2[]$	$f^3[]$	$f^4[]$
0	0.0000	**1.0080**	**0.2800**	**1.1000**	**1.0000**	**-0.0000**
1	0.2000	1.0640	0.6100	1.6000	1.0000	0.0000
2	0.3000	1.1250	1.0900	2.0000	0.0000	0.0000
3	0.5000	1.3430	1.6900	0.0000	0.0000	0.0000
4	0.6000	1.5120	0.0000	0.0000	0.0000	0.0000

Utilizamos os valores em azul no momento a substituição

$$f(0.4) = f[] + (0.4 - x_0)f^1[] + (0.4 - x_0)(0.4 - x_1)f^2[] +$$
$$+(0.4 - x_0)(0.4 - x_1)(0.4 - x_2)f^3[] + (0.4 - x_0)(0.4 - x_1)(0.4 - x_2)(0.4 - x_3)f^4[]$$

$$f(0.4) = 1.2160$$

Exemplo 2. Determinar o valor aproximado de $f(0.2)$, usando todos os pontos tabelados

i	x_i	y_i
0	0.0	1.000
1	0.1	2.001
2	0.3	4.081
3	0.6	8.296
4	1.0	21.000

i	x_i	$y_i = f[]$	$f^1[]$	$f^2[]$	$f^3[]$	$f^4[]$
0	0.000	**1.0000**	**10.0100**	**1.3000**	**10.0000**	**10.0000**
1	0.1000	2.0010	10.4000	7.3000	20.0000	0.0000
2	0.3000	4.0810	14.0500	25.3000	0.0000	0.0000
3	0.6000	8.2960	31.7600	0.0000	0.0000	0.0000
4	1.0000	21.0000	0.0000	0.0000	0.0000	0.0000

Utilizamos os valores em azul no momento as substituição

$$f(0.2) = f[] + (0.2 - x_0)f^1[] + (0.2 - x_0)(0.2 - x_1)f^2[] +$$
$$+(0.2 - x_0)(0.2 - x_1)(0.2 - x_2)f^3[] + (0.2 - x_0)(0.2 - x_1)(0.2 - x_2)(0.2 - x_3)f^4[]$$

$$f(0.2) = 3.0160$$

PROGRAMA EM PYTHON

```
# Interpolação de Newton

import numpy as np

# Entrada
x0 = 0.4    #valor a ser interpolado

D = np.array(
    [[0.0   ,  1.008],
     [0.2   ,  1.064],
     [0.3   ,  1.125],
     [0.5   ,  1.343],
     [0.6   ,  1.512]]
    )

print("Interpolação de Newton")
print("Dado")
print(D)
#print(D[0,:])

# Variáveis auxiliares
s = 0
p = 1

# matriz linha X coluna
linha = np.size(D[:,1])
coluna = np.size(D[1,:])

#print("[linha, coluna] = " + format([linha, coluna]))

s = 0

shape = (linha, linha)
M = np.zeros(shape)

M[:,0] = D[:,1]

t = 0

for j in range(0 , linha , 1):
  t = t + 1
  for i in range(1 , linha - j , 1):
    #print([i , j , M[i,j-1] , M[i-1,j-1], D[i+j , 0] , D[i+j-t , 0]])
    M[i-1,j+1] = (M[i , j] - M[i-1 , j]) / (D[i+j , 0] - D[i+j-t , 0])
    #print("%2d"%(i-1), "%2d"%(j+1), "%8.4f"%M[i-1,j+1], "%8.4f"%M[i ,
j] , "%8.4f"%M[i-1 , j], "%8.4f"%D[i+j , 0], "%8.4f"%D[i+j-t , 0])

print("Tabela")
print('    f[]     f1[]     f2[]     f3[]     f4[]')
```

```
for i in range(0 , linha , 1):
  for j in range(0 , linha , 1):
    print("%8.4f"%M[i,j], end=' ')
  print(" ")


s = M[0,0]
for j in range(1 , linha , 1):
    p = 1;
    for i in range(0 , j , 1):
        p = p * ( x0 - D[i , 0] )
    s =  s + M[0 , j] * p

print('\nValor de x é: {:8.4f}'.format(x0))
print('Valor interpolado é: {:8.4f}'.format(s))
```

SAÍDA DO PROGRAMA

```
Interpolação de Newton
Dado
[[0.    1.008]
 [0.2   1.064]
 [0.3   1.125]
 [0.5   1.343]
 [0.6   1.512]]
Tabela
    f[]     f1[]     f2[]     f3[]     f4[]
  1.0080   0.2800   1.1000   1.0000  -0.0000
  1.0640   0.6100   1.6000   1.0000   0.0000
  1.1250   1.0900   2.0000   0.0000   0.0000
  1.3430   1.6900   0.0000   0.0000   0.0000
  1.5120   0.0000   0.0000   0.0000   0.0000

Valor de x é:   0.4000
Valor interpolado é:   1.2160
```

ATIVIDADE

(01) Determinar o valor aproximado de $f(0.3)$, usando todos os pontos tabelados

i	x_i	y_i
0	0.0	0.0000
1	0.2	0.0480
2	0.4	0.2240
3	0.6	0.5760
4	0.8	1.1520

(02) Determinar o valor aproximado de $f(0.4)$, usando todos os pontos tabelados

i	x_i	y_i
0	0.1	0.1010
1	0.3	0.3270
2	0.5	0.6250
3	0.7	1.0430
4	0.9	1.6290

(03) Determinar o valor aproximado de $f(0.3)$, usando todos os pontos tabelados

i	x_i	y_i
0	0.0	0.1000
1	0.2	0.1080
2	0.4	0.1640
3	0.6	0.3160
4	0.8	0.6120

7. INTEGRAÇÃO NUMÉRICA

Ao se resolver certos problemas, são comuns soluções que recaiam no cálculo de área de figuras plana onde se conhece as equações que contornam a figura. O problema a seguir, é um bom exemplo desta situação.

Exemplo 1. Um móvel se desloca ao longo de uma trajetória retilínea segunda a equação horária $v = 4t - t^2$, onde o tempo é medido em segundos e a distância em metros. O gráfico da função horária está mostrado a figura a seguir.

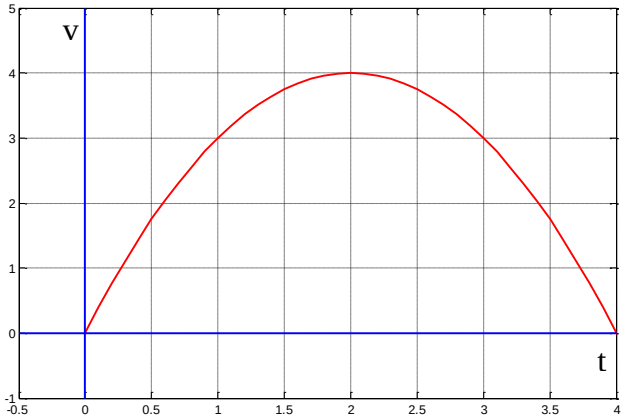


Figura 1 – Gráfico da função $v = 4t - t^2$, onde o tempo está em segundos e a velocidade em m/s.

O deslocamento deste móvel nos primeiros 4 segundos é determinado calculando a área plana compreendida entre a equação $v = 4t - t^2$ e o eixo dos tempos, isto é, determinar a área rachurada mostrada na figura 2.

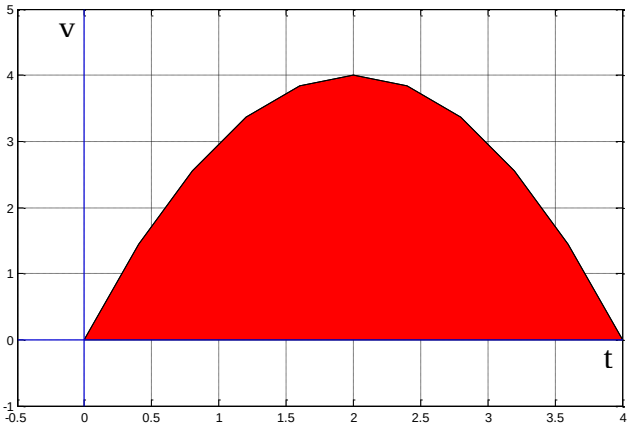


Figura 2 – Gráfico da função $v = 4t - t^2$, onde o t é tempo (seg) e v é a velocidade (m/s). A parte rachurada corresponde ao deslocamento do móvel.

Como calcular esta área? Se a função $f(x)$ é contínua em um intervalo $[a, b]$ e sua primitiva $F(x)$ é conhecida, então a área é calculada pela integral definida desta função no intervalo definido e é dada por:

$$\int_a^b f(x)dx = F(b) - F(a)$$

onde $F'(x) = f(x)$.

Como é feito em situações práticas? Em muitas situações práticas, onde não se tem uma fórmula analítica para a função $f(x)$, e sim uma tabela de pontos que expressão seu comportamento, para calculamos a área através do valor da integra definida de $f(x)$ é necessário a utilização de métodos numéricos.

7.1. REGRA DOS TRAPÉZIOS

Neste método, substituímos a rachurada que se deseja calcular pela área de um trapézio como ilustra a figura a seguir.

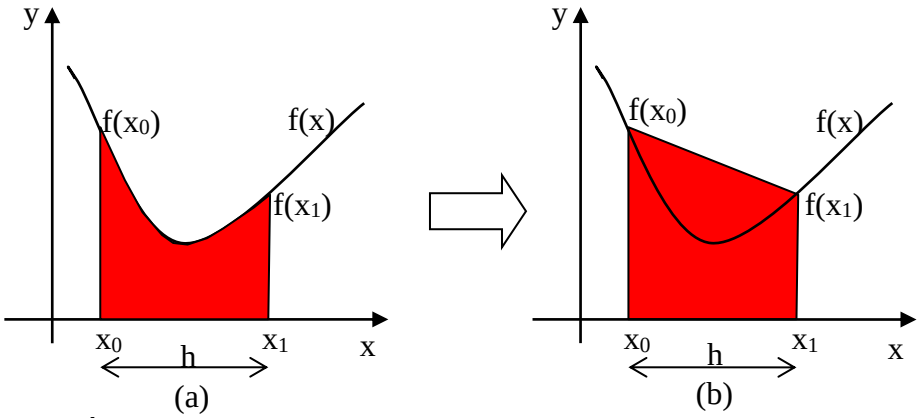


Figura 3 – (a) Área rachurada compreendida pela função $f(x)$ e o eixo do x no intervalo $[x_0 x_1]$. (b) Trapézio utilizado para aproximar a área rachurada do item (a).

O trapézio utilizado para aproximar a área rachurada é determinado, utilizando os dois pontos do intervalo, onde passamos uma reta. Da geometria sabemos que a área deste trapézio é dada por:

$$A = \frac{h}{2}[f(x_0) + f(x_1)]$$

A diferença entre a integral exata de $f(x)$ (área sob a curva $f(x)$) e a integral aproximada (área do trapézio) é denominada de erro de integração. A diferença dos resultados não é muito grande?

Uma forma de se melhorar o resultado estimado, isto é, diminuir a diferença entre o resultado estimado e o exato na regra do trapézio é subdividir o intervalo $[x_0 , x_1]$ em n intervalos de amplitude h e em cada intervalo aplica-se a regra dos trapézios, como ilustra a figura 4.

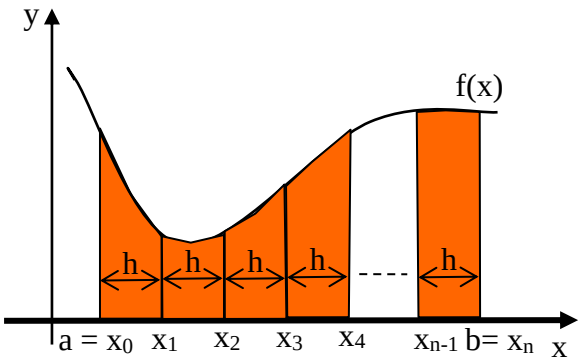


Figura 4 – Área compreendida pela função $f(x)$ e o eixo do x no intervalo $[x_0 x_1]$ é aproximada pela soma de n áreas dos trapézios de mesma base compreendidos no intervalo $[x_0 x_1]$.

Desta forma, a área aproximada é calculada pela expressão:

$$A = \frac{h}{2}(y_0 + y_1) + \frac{h}{2}(y_1 + y_2) + \ldots + \frac{h}{2}(y_{n-1} + y_n)$$

Que pode ser simplificado para

$$A = \frac{h}{2}(y_0 + 2y_1 + 2y_3 + \ldots + 2y_{n-1} + y_n)$$

Onde E_i é o erro cometido na aplicação da regra dos trapézios no intervalo cujos extremos são x_i e x_{i+1}, ou seja,

$$E_i = \frac{-h^3}{12} f''(\varepsilon)$$

Com isto o erro total cometido é a soma dos erros cometidos em cada intervalo, logo

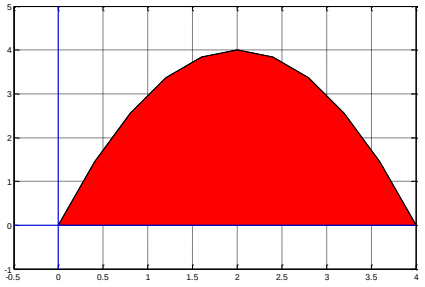
$$E = \frac{-h^3}{12} \sum_{i=1}^{n-1} f''(\varepsilon_i)$$

e pela continuidade de $f''(\varepsilon)$, existe n em $a \leq \varepsilon \leq b$, tal que

$E_i = -\frac{(b-a)^3}{12n^2} f''(\varepsilon)$, onde $a \leq \varepsilon \leq b$.

Exemplo 1 – Calcule a área entre o gráfico $v = 4t - t^2$ e o eixo do x, dentro do intervalo [0 , 4].

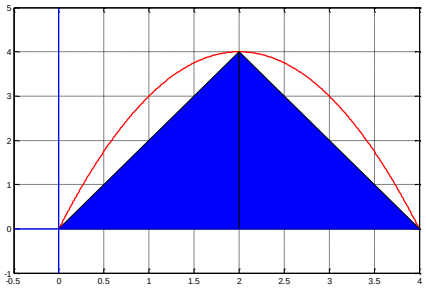
A precisão do valor aproximado depende do número n de trapézios, observe



Resolução analítica:

$$A = \int_0^4 (4t - t^2)dt = (2t^2 - \frac{t^3}{3})_0^4$$

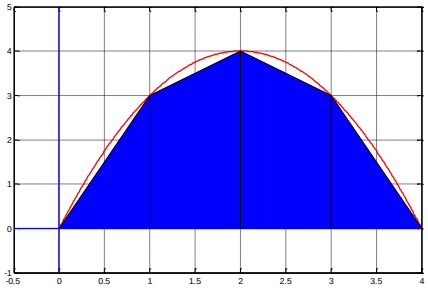
$$A = (2*4^2 - \frac{4^3}{3}) - (2*0^2 - \frac{0^3}{3}) \Rightarrow A = \frac{32}{3} = 10.6667$$



Aproximação para n = 2

$$A = \frac{h}{2}(y_1 + 2y_2 + y_3) \Rightarrow A = 8$$

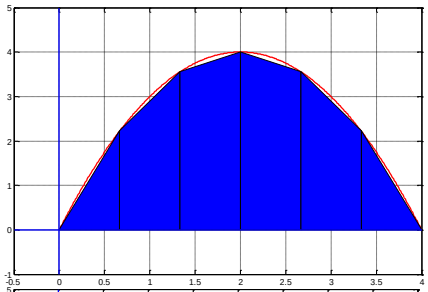
$$E = -\frac{(b-a)^3}{12n^2} f''(\varepsilon) \Rightarrow E = 2.6667$$



Aproximação para n = 4

$$A = \frac{h}{2}(y_1 + 2y_2 + 2y_3 + 2y_4 + y_5) \Rightarrow A = 10$$

$$E = -\frac{(b-a)^3}{12n^2} f''(\varepsilon) \Rightarrow E = 0.6667$$

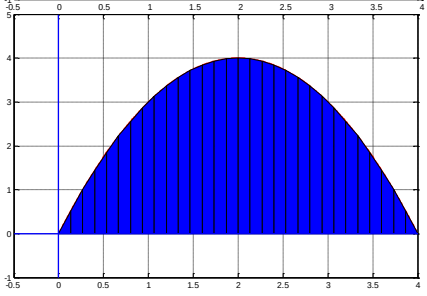


Aproximação para n = 6

$$A = \frac{h}{2}(y_1 + 2y_2 + 2y_3 + 2y_4 + 2y_5 + 2y_6 + y_7)$$

$$A = 10.3704$$

$$E = -\frac{(b-a)^3}{12n^2} f''(\varepsilon) \Rightarrow E = 0.2963$$



Aproximação para n = 30

$$A = 10.6548$$

$$E = -\frac{(b-a)^3}{12n^2} f''(\varepsilon) \Rightarrow E = 0.0119$$

Figura 5 – Mostrando a aproximação pela regra dos trapézios para diferentes valores de n.

Com $v'(t) = 4 - 2t$, e como $v''(t) = -2$, logo $f''(0) = -2$ em todas as expressões, onde $0 \leq \varepsilon \leq 4$.

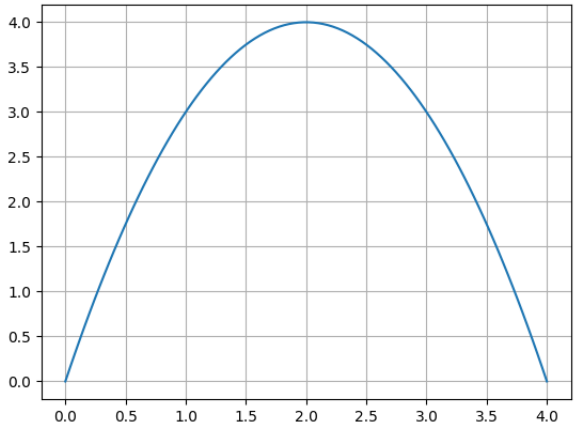
PROGRAMA EM PYTHON

```
# Regra dos Trapézios
# Entrada
xi = 0.         # intervalo [xi , xf]
xf = 4.
n = 30          # número de trapézios

def f(x):
  return 4*x - x**2

print("Integração Numérica - Método dos Trapézios")
print('f(x) = 4*x - x**2')

import math
import numpy as np

# variáveis auxiliar
h = 0
s = 0

h = (xf - xi)/n
vx = np.zeros((n+1))
vy = np.zeros((n+1))

vx[0] = xi
for i in range(1 , n+1 , 1):
  vx[i] = vx[i-1] + h

vy[0] = f(vx[0])
vy[n] = f(vx[n])
for i in range(1 , n , 1):
  vy[i] = 2 * f(vx[i])

#print(vx[:])
#print(vy[:])

s = 0
for i in range(0 , n+1 , 1):
  s = s + vy[i]

s = (h/2) * s

print("Número de trapézios: " , "%d"%n)
print("Intervalo: " ,"[" , "%8.4f"%xi , "," ,"%8.4f"%xf, "]")
print('Valor da Integral: {:8.4f}'.format(s))

import matplotlib.pyplot as plt
import numpy as np
```

```
xi = np.linspace(xi, xf, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Integração Numérica - Método dos Trapézios
f(x) = 4*x - x**2
Número de trapézios:  30
Intervalo:  [   0.0000 ,   4.0000 ]
Valor da Integral:  10.6548
```



ATIVIDADE

(01) Dada a função $f(x) = x^2$ calcular o valor da integral $I = \int_0^3 f(x)dx$, usando a regra dos trapézios e dividindo o intervalo em 6 partes.

(02) Dada a função $f(x) = ln\, x$ calcular o valor da integral $I = \int_2^4 f(x)dx$, usando a regra dos trapézios e dividindo o intervalo em 6 partes.

(03) Dada a função $f(x) = x^3$ calcular o valor da integral $I = \int_0^3 f(x)dx$, usando a regra dos trapézios e dividindo o intervalo em 6 partes.

(04) Dada a função $f(x) = e^x$ calcular o valor da integral $I = \int_2^4 f(x)dx$, usando a regra dos trapézios e dividindo o intervalo em 6 partes.

Você saiba que na regra dos trapézios, utilizamos uma aproximação de primeira ordem do polinômio interpolador de Gregory-Newton $P_n(x)$ para representar a função $f(x)$.

$$P_n(x) = y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0 + \frac{z(z-1)(z-2)}{3!} * \Delta^3 y_0 + \ldots +$$
$$+ \frac{z(z-1)(z-2) * \ldots * (z-n+1)}{(n+1)!} * \Delta^2 y_0$$

Isto é, utilizamos na regra do trapézio, utilizamos $P_2(x) = y_0 + z\Delta y_0$ (n = 1), para aproximar $f(x)$, com isto a integral passou a ser determinada por

$$I = \int_a^b f(x)dx = \int_a^b [y_0 + z\Delta y_0]dx$$

Como $z = \frac{x-x_0}{h} \Rightarrow \quad dx = hdz$,

e considerando $a = x_0$ e $b = x_1$, temos que

para $x = a \Rightarrow \quad z = \frac{x_0-x_0}{h} = 0$,

para $x = b \Rightarrow \quad z = \frac{x_1-x_0}{h} = 1$

substituindo os limes na integral temos

$$I = \int_a^b [y_0 + z\Delta y_0]dx = \int_0^1 [y_0 + z\Delta y_0]hdz = h\left[zy_0 + \frac{z^2}{2}\Delta y_0\right]_0^1$$

$$I = h\left[1 * y_0 + \frac{1^2}{2}\Delta y_0\right] - h\left[0 * y_0 + \frac{0^2}{2}\Delta y_0\right]$$

$I = h\left[y_0 + \frac{1}{2}\Delta y_0\right] \quad \Rightarrow \quad I = h\left[y_0 + \frac{1}{2}(y - y_0)\right]$

$I = h\left[\frac{y+y_0}{2}\right]$, foi esta a expressão utilizada no método dos trapézios.

7.2. PRIMEIRA REGRA DE SIMPSON

A vantagem, de revermos o método dos trapézios usando o polinômio interpolador de Gregory-Newton ($P_n(x)$) e que na primeira regra de Simpson, utilizamos uma aproximação de 2ª ordem deste polinômio, isto é, faremos:

$f(x) = y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0$, onde $z = \frac{x-x_0}{h}$

Com isto o valor da integral ser:

$$I = \int_a^b f(x)dx = \int_a^b \left[y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0\right] dx$$

Como $z = \frac{x-x_0}{h} \Rightarrow dx = hdz$,

Para se aproximar a função $f(x)$ por um polinômio do 2º grau, serão necessários 3 pontos: x_0, x_1 e x_2 (Figura 6).

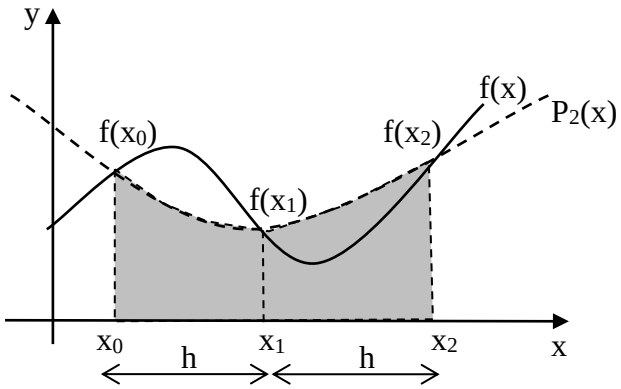


Figura 6 – Gráfico de $f(x)$ juntamente com a aproximação de segunda ordem $P_2(x)$.

Considerando $a = x_0$ e $b = x_2$, temos que :

$x = a \quad \Rightarrow \quad z = \frac{a-a}{h} = 0$,

$x = b \quad \Rightarrow \quad z = \frac{b-a}{h} = 2$

Com isto, a integral será resolvida da seguinte forma

$$I = \int_a^b f(x)dx = \int_0^2 \left[y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0\right] hdz$$

Cujo resultado é:

$$I = h\left[2y_0 + 2\Delta y_0 + \frac{1}{3}\Delta^2 y_0\right]$$

Como babemos que $\begin{cases}\Delta y_0 = y_1 - y_0 \\ \Delta^2 y_0 = y_2 - 2y_1 + y_0\end{cases}$, então com a substituição teremos

$I = \frac{h}{3}[y_0 + 4y_1 + y_2]$ que é denominado de 1ª regra de Simpson.

$I = h\left[\frac{y+y_0}{2}\right]$, foi esta a expressão utilizada no método dos trapézios.

Para diminuir o erro, isto é, a diferença do valor estimado e do valor real, devemos subdividir o intervalo de integração, da mesma forma que fizemos no método dos trapézios, com isto, a integral $I = \int_a^b f(x)dx$, será aplicada em cada dupla de intervalos da seguinte forma:

$$I = \frac{h}{3}\underbrace{[y_0 + 4y_1 + y_2]}_{1º sub\ int ervalo} + \frac{h}{3}\underbrace{[y_2 + 4y_3 + y_4]}_{2º sub\ int ervalo} + \ldots + \frac{h}{3}\underbrace{[y_{n-2} + 4y_{n-1} + y_n]}_{últimosub\ int ervalo}$$

O erro total cometido será a soma dos erros cometidos em cada aplicação da 1ª regra de Simpson nas duplas de subintervalos e são determinados por:

$E = \frac{-(b-a)^5}{180n^4} f^{(IV)}(\varepsilon)$, onde $a \leq \varepsilon \leq b$.

Exemplo 1. Calcule o valor da integral $\int_0^1 \frac{dx}{1+x^2}$, com $\varepsilon \leq 10^{-4}$.

Solução

Calcular esta integral significa calcular a área compreendida entre o gráfico e o eixo x, como mostra a figura a seguir.

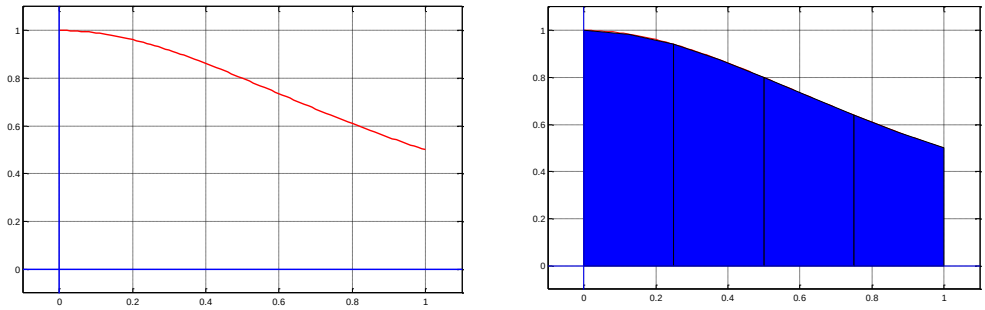


Figura 7 – Gráfico da função $f(x) = \frac{1}{1+x^2}$, onde a área rachurada representa o resultado da integral $\int_0^1 \frac{dx}{1+x^2}$.

Devemos definir qual dever ser o número n de subintervalos devemos usar, para isto utilizaremos a nossa fórmula do erro total

$E = \frac{-(b-a)^5}{180n^4} f^{(IV)}(\varepsilon)$, onde $a \leq \varepsilon \leq b$.

Como $f(x) = \frac{1}{1+x^2}$, então temos que

$f^{IV}(x) = \frac{24}{(1+x^2)^3} - \frac{288x^2}{(1+x^2)^4} + \frac{384x^4}{(1+x^2)^5}$, onde $0 \leq \varepsilon \leq 1$

Sabemos que o maior erro total será obtido quando $x = 0$, logo $|f^{IV}(x)|_{max}$, e considerando $\varepsilon \leq 10^{-4}$, então temos:

$\frac{-(1-0)^5}{180n^4} * 24 \leq 10^{-4} \Rightarrow \quad n^4 \geq \frac{24}{180} 10^4 \Rightarrow n \geq 6.042$

Isto é, devemos escolher um número de subintervalos maior que 7, e escolheremos para este caso $n = 8$. O valor da aproximação foi obtido, para $n = 8$, a partir da tabela a seguir.

i	x_i	y_i	c_i
0	0.0000	1.0000	1
1	0.1250	0.9846	4
2	0.2500	0.9412	2
3	0.3750	0.8767	4
4	0.5000	0.8000	2
5	0.6250	0.7191	4
6	0.7500	0.6400	2
7	0.8750	0.5664	4
8	1.0000	0.5000	1

Tabela 1- c_i são os coeficientes que devem ser aplicados y_i para determinar a aproximação do valor da integral.

Para calcularmos o valor da integral pela seguinte expressão

$$\int_0^1 \frac{dx}{1+x^2} = \frac{1}{h}\{y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + 4y_5 + 2y_6 + 4y_7 + y_8\}$$

Substituindo os valores da tabela teremos $\int_0^1 \frac{dx}{1+x^2} = 0.7854$

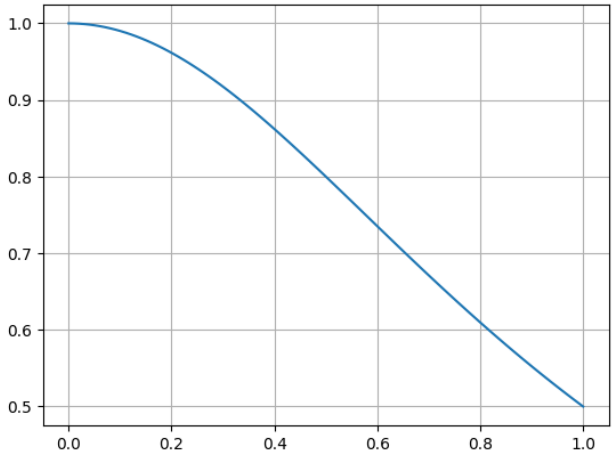
PROGRAMA EM PYTHON

```
# Primeira Regra de Simpson
# Entrada
xi = 0.         # intervalo [xi , xf]
xf = 1.
n = 8           # número de intervalos (deve ser um número maior que 7)

def f(x):
  return 1/(1 + x**2)

print("Integração Numérica - 1a regra de Simpson")
print('f(x) = 1/(1 + x**2)')

import math
import numpy as np
```

```
# variáveis auxiliar
h = 0
s = 0

h = (xf - xi)/n
vx = np.zeros((n+1))
vy = np.zeros((n+1))

vx[0] = xi
for i in range(1 , n+1 , 1):
  vx[i] = vx[i-1] + h

vy[0] = f(vx[0])
vy[n] = f(vx[n])
for i in range(1 , n , 1):
  vy[i] = f(vx[i])

#print(vx[:])
#print(vy[:])

s = 0
for i in range(0 , n , 2):
  s = s + (h/3)*(vy[i]+4*vy[i+1]+vy[i+2])

#s = (1/h) * s

print("Número de intervalos: " , "%d"%n)
print("Intervalo: " ,"[" , "%8.4f"%xi , "," ,"%8.4f"%xf, "]")
print('Valor da Integral: {:8.4f}'.format(s))
print(' ')

import matplotlib.pyplot as plt
import numpy as np
#xi = np.linspace(-10, 10, 100)
xi = np.linspace(xi, xf, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Integração Numérica - 1a regra de Simpson
f(x) = 1/(1 + x**2)
Número de intervalos:  8
Intervalo:  [   0.0000 ,   1.0000 ]
Valor da Integral:   0.7854
```



ATIVIDADE

(01) Calcule a integral $\int_0^1 \frac{dx}{1+2x^2}$, com $\varepsilon \leq 10^{-4}$, usando a 1ª regra de Simpson.

(02) Calcule a integral $\int_1^2 ln(1+x)dx$, com $\varepsilon \leq 10^{-4}$, usando a 1ª regra de Simpson.

(03) Calcule o valor da integral $\int_0^1 \frac{dx}{1+2x^3}$, com $\varepsilon \leq 10^{-4}$, usando a primeira regra de Simpson.

(04) Calcule o valor da integral $\int_1^2 ln(1+x^2)dx$, com $\varepsilon \leq 10^{-4}$, usando a primeira regra de Simpson.

7.3. SEGUNDA REGRA DE SIMPSON

Na segunda regra de Simpson utilizamos uma aproximação de terceira ordem no polinômio interpolador de Gregory-Newton ($P_n(x)$) o que resulta na expressão:

$P_n(x) = y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0 + \frac{z(z-1)(z-2)}{3!} * \Delta^3 y_0$, onde $z = \frac{x-x_0}{h}$.

Com isto o valor da integral ser:

$$I = \int_a^b f(x)dx = \int_a^b \left[y_0 + z\Delta y_0 + \frac{z(z-1)}{2!} * \Delta^2 y_0 + \frac{z(z-1)(z-2)}{3!} * \Delta^3 y_0\right] dx$$

como $z = \frac{x-x_0}{h} \Rightarrow dx = hdz$,

Desta forma a solução da integral é:

$$I = \frac{3h}{8}[y_0 + 3y_1 + 3y_2 + y_3]$$

O erro total neste método é dado pela expressão

$E = \frac{-3x^5}{80} f^{IV}(\varepsilon), \quad a \leq \varepsilon \leq b.$

Para diminuir o erro quando o intervalo não for muito pequeno, devemos subdividir o intervalo de integração da seguinte forma:

$$I = \frac{3h}{8}\underbrace{[y_0 + 3y_1 + 3y_2 + y_3]}_{1º sub\ int ervalo} + \frac{3h}{8}\underbrace{[y_3 + 3y_4 + 3y_5 + y_6]}_{2º sub\ int ervalo} + \ldots + \frac{3h}{8}\underbrace{[y_{n-3} + 3y_{n-2} + 3y_{n-1} + y_n]}_{últimosub\ int ervalo}$$

Exemplo 1 – Calcule o valor da integral $I = \int_1^4 ln(x^3 + e^x)dx$

Solução

Calcular esta integral significa determinar a área compreendida entre o gráfico e o eixo x, como mostra a Figura 8. O valor da integral é obtido pela seguinte expressão:

$$\int_1^4 ln(x^3 + e^x)dx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + 2y_6 + 3y_7 + 3y_8 + y_9\}$$

Os valores de $y_0, y_1, y_2, \ldots, y_n$ são obtidos na tabela a seguir,

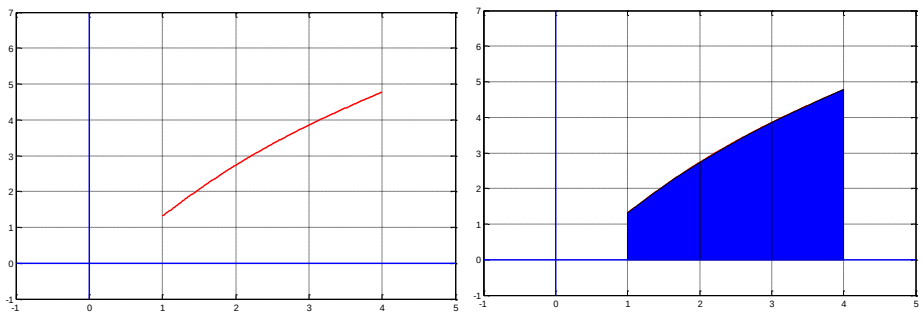


Figura 8 – Gráfico da função $f(x) = ln(x^3 + e^x)$, onde a área rachurada representa o resultado da integral $\int_1^4 ln(x^3 + e^x)dx$.

O valor da aproximação foi obtido, para $n = 9$, a partir da tabela a seguir.

i	x_i	y_i	c_i
0	1.0000	1.3133	1
1	1.3333	1.8187	3
2	1.6667	2.2950	3
3	2.0000	2.7337	2
4	2.3333	3.1362	3
5	2.6667	3.5072	3
6	3.0000	3.8520	2
7	3.3333	4.1754	3
8	3.6667	4.4821	3
9	4.0000	4.7757	1

Tabela 2 - c_i são os coeficientes que devem ser aplicados yi para determinar a aproximação do valor da integral.

Substituindo os valores da tabela teremos $\int_1^4 ln(x^3 + e^x)dx = 9.6880$

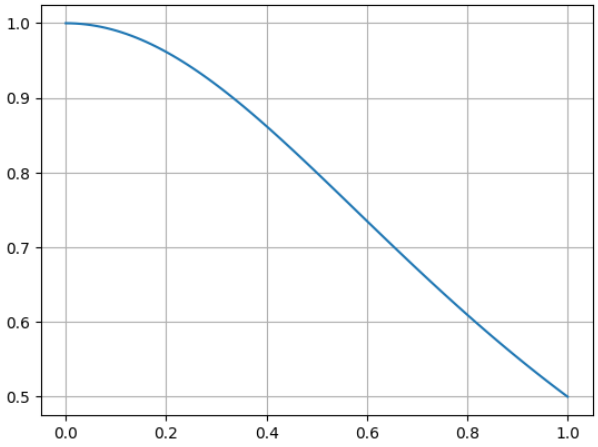
PROGRAMA EM PYTHON

```
# Segunda Regra de Simpson
# Entrada
xi = 0.         # intervalo [xi , xf]
xf = 1.
n = 9           # número de intervalos (deve ser um numeto maior que 8)

def f(x):
  return 1/(1 + x**2)

print("Integração Numérica - 2a regra de Simpson")
print('f(x) = 1/(1 + x**2)')

import math
import numpy as np


# variáveis auxiliar
h = 0
s = 0

h = (xf - xi)/n
vx = np.zeros((n+1))
vy = np.zeros((n+1))

vx[0] = xi
for i in range(1 , n+1 , 1):
```

```
  vx[i] = vx[i-1] + h

vy[0] = f(vx[0])
vy[n] = f(vx[n])
for i in range(1 , n , 1):
  vy[i] = f(vx[i])

#print(vx[:])
#print(vy[:])

s = 0
for i in range(0 , n-2 , 3):
  s = s + (3*h/8)*(vy[i]+3*vy[i+1]+3*vy[i+2]+vy[i+3])

print("Número de intervalos: " , "%d"%n)
print("Intervalo: " ,"[" , "%8.4f"%xi , "," ,"%8.4f"%xf, "]")
print('Valor da Integral: {:8.4f}'.format(s))
print(' ')

import matplotlib.pyplot as plt
import numpy as np
#xi = np.linspace(-10, 10, 100)
xi = np.linspace(xi, xf, 100)

fig = plt.figure()
plt.plot(xi, f(xi), '-')
plt.grid()
```

SAÍDA DO PROGRAMA

```
Integração Numérica - 2a regra de Simpson
f(x) = 1/(1 + x**2)
Número de intervalos:  9
Intervalo:  [   0.0000 ,   1.0000 ]
Valor da Integral:   0.7854
```



ATIVIDADE

(01) Calcule o valor da integral $\int_0^1 \frac{dx}{1+2x^2}$, com $\varepsilon \leq 10^{-4}$, usando a segunda regra de Simpson.

(02) Calcule o valor da integral $\int_1^2 ln(1+x)dx$, com $\varepsilon \leq 10^{-4}$, usando a segunda regra de Simpson.

(03) Calcule a integral $\int_0^1 \frac{dx}{1+2x^3}$, com $\varepsilon \leq 10^{-4}$, usando a 2ª regra de Simpson.

(04) Calcule a integral $\int_1^2 ln(1+x^2)dx$, com $\varepsilon \leq 10^{-4}$, usando a 2ª regra de Simpson.

FÁBIO JOSÉ DA COSTA ALVES - Licenciatura em Matemática pela União das Escolas Superiores do Pará, Licenciatura em Ciências de 1º Grau pela União das Escolas Superiores do Pará, Graduação em Engenharia Civil pela Universidade Federal do Pará. Possui Mestrado e Doutorado em Geofísica pela Universidade Federal do Pará e Pós-Doutorado pelo Programa de Pós-Graduação em Ensino de Ciências e Matemática da Universidade Federal do Rio Grande do Norte. Professor da Universidade do Estado do Pará. Docente do Programa de Pós-Graduação em Educação/UEPA e Docente do Programa de Pós-Graduação em Ensino de Matemática/UEPA. Líder do Grupo de Pesquisa em Ensino de Matemática e Tecnologias. Experiência em desenvolvimento de software educativo para o ensino de matemática.

CINTHIA CUNHA MARADEI PEREIRA - Possui Graduação em Licenciatura em Matemática e em Tecnologia em Processamento de Dados, Especialização em Informática Médica, Mestrado em Ciências da Computação e Doutorado em Genética e Biologia Molecular (Bioinformática). Professora da Universidade do Estado do Pará. Docente do Programa de Pós-Graduação em Ensino de Matemática/UEPA. Vice-líder do Grupo de Pesquisa em Ensino de Matemática e Tecnologias.