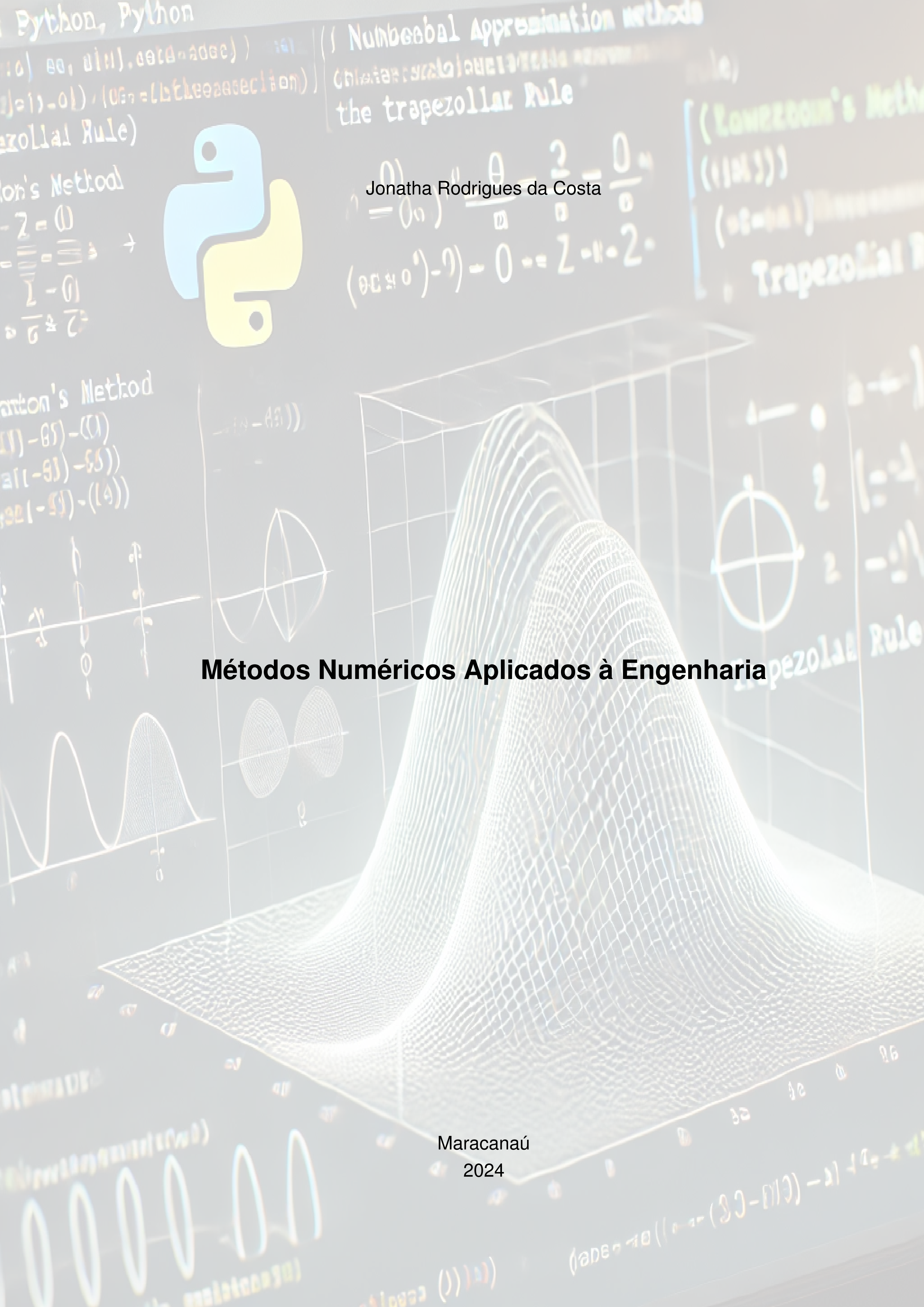


Jonatha Rodrigues da Costa

Métodos Numéricos Aplicados à Engenharia

Maracanaú
2024

Jonatha Rodrigues da Costa

Métodos Numéricos Aplicados à Engenharia

Este livro trata sobre métodos numéricos aplicados à engenharia incluindo códigos em Python® .

Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Maracanaú
2024

Dedico este trabalho a Deus, a minha família, aos colegas acadêmicos e aos discentes, especialmente os discentes dos cursos de engenharia.

Agradecimentos

"Estudar é uma dádiva divina!"

Resumo

O livro "Métodos Numéricos Aplicados à Engenharia"se destaca como uma valiosa contribuição para a comunidade acadêmica, abordando a aplicação prática de métodos numéricos no contexto da engenharia. Com um foco especial em Python®, o livro explora diversas bibliotecas populares, como NumPy, SciPy e Matplotlib, para facilitar a implementação e visualização de algoritmos numéricos.
Uma das principais inovações deste trabalho é a comparação sistemática dos resultados obtidos por diferentes métodos numéricos, proporcionando aos leitores uma compreensão crítica das técnicas disponíveis. Cada capítulo é estruturado para apresentar um problema específico de engenharia, seguido pela aplicação dos métodos numéricos, permitindo que os leitores vejam a transição da teoria à prática.
Além disso, todos os códigos utilizados nas demonstrações e exemplos são disponibilizados em um repositório no GitHub®, promovendo a transparência e a colaboração na comunidade acadêmica. Isso permite que estudantes e profissionais acessem, experimentem e aprimorem os algoritmos discutidos no livro, incentivando um aprendizado ativo e uma melhor assimilação dos conteúdos abordados.
Com uma abordagem didática e prática, “Métodos Numéricos Aplicados à Engenharia”não apenas enriquece o conhecimento técnico dos leitores, mas também serve como um recurso valioso para a formação de futuros engenheiros e pesquisadores na utilização eficaz de ferramentas computacionais em suas atividades.

Sumário

Lista de ilustrações

1 Introdução

Métodos numéricos (MN) formam um conjunto poderoso de técnicas e algoritmos destinados à resolução de problemas complexos, com base em aproximações sucessivas. Essencialmente, esses métodos transformam equações matemáticas, muitas vezes difíceis de serem resolvidas analiticamente, em proposições mais simples, tratadas com operações aritméticas de baixa complexidade, mas alta precisão.

Esses métodos são amplamente aplicados em áreas onde soluções exatas não podem ser obtidas de maneira prática ou eficiente, como no caso de equações diferenciais, integrais e sistemas de equações lineares e não lineares. Sua relevância vem se consolidando cada vez mais em disciplinas como engenharia, física, ciência da computação, entre outras, sendo ferramentas fundamentais para a solução de problemas matemáticos. O rápido avanço das capacidades computacionais ao longo das últimas décadas tornou os métodos numéricos indispensáveis, permitindo a resolução de problemas que, há não muito tempo, seriam considerados intratáveis. Com o suporte de recursos computacionais acessíveis, a implementação de algoritmos de MN tornou-se mais eficiente e prática.

Conhecidos também como abordagens indiretas, os métodos numéricos baseiam-se na repetição de cálculos detalhados e progressivos, focados na obtenção de soluções aproximadas que convergem para resultados dentro de um intervalo de precisão aceitável. Embora as operações envolvidas sejam relativamente simples, a execução repetitiva pode gerar processos altamente intensivos do ponto de vista computacional. Nesse contexto, o conceito de **convergência** assume papel central: o objetivo dos MN é garantir que, com o aumento das iterações ou do refinamento do modelo, a solução se aproxime cada vez mais da resposta exata.

A análise numérica, ramo da matemática que estuda o comportamento desses métodos, é essencial para garantir a validade e a eficácia das soluções obtidas. Por meio dela, é possível avaliar a precisão, estabilidade e velocidade de convergência dos algoritmos aplicados. Além disso, a análise numérica oferece ferramentas para estudar e controlar os erros introduzidos pelas aproximações, como o **erro de truncamento** e o **erro de arredondamento**, que são inevitáveis em qualquer cálculo que envolva números com precisão finita. Assim, o desenvolvimento de métodos eficazes não se limita à busca de soluções; envolve também a compreensão profunda de como minimizar e gerenciar esses erros.

Os métodos numéricos, portanto, não fornecem soluções exatas no sentido clássico, mas sim **soluções aproximadas**, que são suficientemente precisas para a maioria das aplicações práticas. Esse caráter aproximativo é particularmente relevante no campo da engenharia, onde a modelagem de fenômenos físicos complexos resulta frequentemente em equações que não podem ser resolvidas de forma analítica. Como destacado por Mathews et al. (2000) e Gilat e Subramaniam (2008), a aplicação de métodos numéricos oferece uma solução computacionalmente viável para tais desafios, fornecendo respostas práticas e aplicáveis.

A importância dos métodos numéricos se amplia ainda mais quando consideramos as condições reais, onde frequentemente lidamos com dados incertos, medições com imprecisões e variabilidade nas condições experimentais. Ao empregar um método numérico, engenheiros e cientistas estão, na verdade, buscando soluções que, embora aproximadas, sejam suficientemente confiáveis para a tomada de decisões. Por isso, a **validação e verificação** das soluções numéricas, por meio de técnicas adequadas, são tão cruciais quanto o próprio processo de cálculo.

Além disso, o surgimento de ferramentas computacionais modernas, como MATLAB®, Python® e outras linguagens de programação, trouxe um nível de acessibilidade sem precedentes para a implementação dos métodos numéricos. Esses *softwares* facilitam a aplicação dos algoritmos, permitindo a resolução de problemas que seriam extremamente trabalhosos ou até inviáveis se feitos manualmente. Dessa forma, a combinação de uma base teórica sólida com o uso de ferramentas computacionais avançadas proporciona aos engenheiros e cientistas meios para abordar problemas de forma mais eficiente e segura.

Em síntese, os métodos numéricos são ferramentas indispensáveis para a resolução de problemas matemáticos em engenharia e em diversas outras disciplinas. Eles oferecem uma solução prática para questões complexas, transformando equações de difícil resolução em respostas viáveis por meio de aproximações precisas. O rigor na implementação e o profundo entendimento da análise de erros e da convergência são fundamentais para garantir que as soluções obtidas sejam confiáveis e aplicáveis na prática.

1.1 ABORDAGENS DE MÉTODOS NUMÉRICOS

Como discutido em (CHAPRA; CANALE, 2016) e (PINTO, 2001), antes da utilização de MNs, a resolução de problemas e formulações matemáticas envolvia principalmente três abordagens, a saber:

a) **Métodos puramente analíticos**: que, embora fossem frequentemente úteis, tinham limitações, oferecendo uma visão completa apenas para sistemas de geometria simples, baixa dimensão e baixa complexidade. Estes métodos eram práticos apenas para sistemas lineares, enquanto problemas do mundo real frequentemente envolvem sistemas não-lineares;

b) **Métodos de soluções gráficas**: que caracterizavam o comportamento dos sistemas por meio de representações gráficas. Essas abordagens buscavam resolver problemas mais complexos, porém, muitas vezes, não produziam resultados precisos. Além disso, a implementação desses métodos podia ser desafiadora;

c) **Métodos manuais com calculadoras**: utilizados para aplicar técnicas numéricas manualmente. No entanto, esses métodos frequentemente resultavam em inconsistências de cálculo devido a erros simples que ocorriam durante a execução de inúmeras tarefas manuais.

Observa-se, com facilidadem que esse processo tornava o trabalho de cálculo por MN demasiadamente exaustivo e passivel do erros humanos. Contudo, ampliação do acesso aos computadores digitais mundou essa abordagem.

1.2 COMPUTADORES E MÉTODOS NUMÉRICOS

O avanço no desenvolvimento de computadores digitais rápidos e eficientes trouxe, por sua vez, um benefício já há muito desejado para a realização de cálculos repetitivos e reprocessamento de dados e sinais, conforme supracitado. Desse modo, além de fornecer uma capacidade de aumenta de processamento e armazenamento de dados, a disponibilidade muito difundida dos computadores (especialmente dos computadores pessoais) representou uma influência significativa na resolução moderna de problemas de engenharia utilizando MN para tanto.

Convergente com isso, ambientes computacionais algébricos como MAPLE®[1] e ambientes de desenvolvimento como o MATLAB® tem destaque desde o contexto acadêmico às aplicações industriais, sendo apontados dentre os preferidos por matemáticos e engenheiros quando necessitam resolver problemas que são viáveis apenas computacionalmente. Ambos os ambientes são fortemente utilizados na computação científica, como também na área de modelagem de sistemas e em atividades que necessitam de carga exaustiva de cálculo matemático, especialmente quando se utilizam aproximações por iterações.

Embora apresentem inúmeras vantagens em termos de facilidade e robustez, os referidos ambientes também apresentam características restritivas como: custo financeiro das licenças, código-fonte fechado[2] e alto custo computacional (consumo de tempo de processamento, memória e de energia no execução de um algoritmo ou *script*) na execução, conforme (SILVA et al., 2014). Essas características podem inviabilizar a utilização desses ambientes quando de aplicações com limitações de recursos.

Em face disto, *softwares* alternativos tem destacado-se com ampla utilização nas áreas supracitadas, conforme Costa (2017). Dentre esses:

1. MATLAB
2. FreeMat
3. Maxima
4. Sage
5. Scilab
6. LabVIEW
7. Octave

O MATLAB® (ícone na Figura 1a) e o Octave® (ícone na Figura 1b) tem destaque especial por sua recursividade. No caso do primeiro, a quantidade de bibliotecas de estruturas de códigos já disponíveis em sua *toolbox* torna-o de elevada atratividade. O **Octave**®[3], por sua vez, sendo gratuito, multi plataforma e de interação próxima ao MATLAB®, tem apresentado-se como uma alternativa à altura quando se trata de modelagem e simulação de sistemas no contexto de MN.

Figura 1 – Ícones de ambientes computacionais matemáticos

(a) MATLAB

(b) Octave

Fonte: Autor, 2024

O MATLAB® & Octave® integram análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar onde problemas e soluções são expressos somente como eles são escritos matematicamente, ao contrário da programação tradicional. Sendo que a ferramenta Octave, está disponível *on-line* e *off-line* sem custo, nos Sistemas Operacionais Microsoft Windows, GNU Linux, *Android* e afins.

[1] MAPLE: um sistema computacional comercial de uso genérico baseado em expressões algébricas, simbólicas, permitindo o desenho de gráficos em duas e três dimensões.

[2] CÓDIGO-FONTE FECHADO: *software* cujo o acesso, utilização, modificação ou redistribuição do código fonte, em quaisquer casos, são proibidos por quem tem os direitos sobre o código.

[3] OCTAVE: Disponível também em simulação on-line https://octave-online.net/

1.3 AMBIENTES INTEGRADOS A NAVEGADORES

Ambientes computacionais via navegadores de *internet* tem ganhado significativo espaço entre profissionais e acadêmicos. Nesse sentido, destaca-se a linguagem de programação Python® - ícone na Figura 2.

Figura 2 – Logo da linguagem Python®

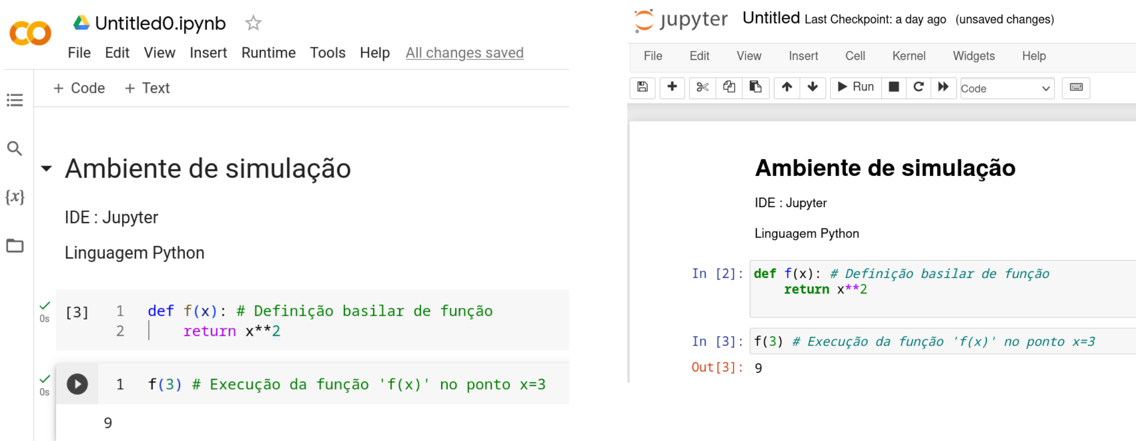
Fonte: Autor, 2024

Python® é uma linguagem de alto nível, interpretada e de utilização geral. Criada por Guido van Rossum, a linguagem tem sua primeira versão lançada em 1991. Desde então, a linguagem ganhou popularidade significativa devido à sua sintaxe simples e legibilidade, o que torna-a uma escolha popular tanto para iniciantes quanto para desenvolvedores experientes, conforme Hunt (2019).

Essa linguagem apresenta-se incorporada a um *(Integrated Development Environment* (ambiente de desenvolvimento integrado) (IDE) de empresas de serviços de *web* como a Google®, pelo recurso *google colaboratory*, Figura 3a. Nesse mesmo contexto o Jupyter Notebook®, Figura 3b, apresenta-se também como uma aplicação *web* interativa que permite criar e compartilhar documentos que contêm código, equações, visualizações e texto narrativo.

Figura 3 – IDEs via navegador de *web*

(a) Colab (b) Jupyter



Fonte: Autor (2024)

Ambas são amplamente utilizados por cientistas de dados, pesquisadores acadêmicos, engenheiros e educadores para explorar dados, criar modelos, realizar análises estatísticas e compartilhar resultados de maneira interativa e fácil de entender. Além destas IDEs que integram esta linguagem, também são conhecidas outras símiles como: Spyder(<https://www.spyder-ide.org>),Visual Studio Code(<https:

//code.visualstudio.com>), GDBonline (<https://www.onlinegdb.com>), Replit (<https://replit.com/>), PyCharm (<https://www.jetbrains.com/pycharm/>) e Atom (<https://atom.io/>).

Em síntese, a utilização de IDEs *on-line* é útil para aprender Python®, por exemple, porque oferece um ambiente de desenvolvimento prático, interativo e sem barreiras, permitindo que os acadêmicos se concentrem na lógica da programação e nas técnicas da linguagem, em vez de se preocuparem com a configuração de ambientes de desenvolvimento complicados. Além disso, também promovem a colaboração e a criação de conteúdo educacional interativo.

1.4 APRENDIZAGEM DA LINGUAGEM PYTHON

Um curso prático de Python® aplicado aos MNs está disponibilizado pelo autor deste trabalho através do Github® em <https://github.com/jonathacosta/py2eng/tree/main/PyBasicCodes>.

2 Cálculo: Fundamentos Matemáticos

Neste capítulo, revisaremos os principais conceitos matemáticos que servem como base para os estudos de MN. Estes incluem cálculo diferencial e integral, além de algumas de suas propriedades essenciais. Estes tópicos são cruciais para a aplicação eficaz de métodos numéricos em problemas práticos de engenharia.

2.1 CONCEITOS DE CÁLCULO: LIMITE, DERIVADA E INTEGRAL

O Cálculo de Limite, Derivada e Integral é um ramo fundamental da matemática, amplamente utilizado para modelar e analisar fenômenos que envolvem variações contínuas. Em problemas de engenharia e outras ciências aplicadas, ele permite descrever tanto mudanças instantâneas quanto acumulações ao longo do tempo ou de espaço, sendo crucial para a resolução de problemas envolvendo movimento, fluxo, taxas de variação, e otimização.

2.2 LIMITES

Os **limites** desempenham um papel central no cálculo e na análise matemática, sendo a base para a definição de conceitos como derivadas e integrais. Um limite é o valor que uma função ou uma sequência "tende a"quando a variável independente se aproxima de um certo ponto.

Formalmente, o limite de uma função $f(x)$ conforme x se aproxima de um valor a é denotado na Equação 1:

$$\lim_{x \to a} f(x) = L \tag{1}$$

Isso significa que, à medida que x se aproxima de a, os valores de $f(x)$ se aproximam de L. Os limites são essenciais para entender o comportamento de funções em pontos específicos, especialmente em casos onde a função pode não estar definida ou pode ter um comportamento indefinido, como divisões por zero.

2.2.1 LIMITES LATERAIS

O conceito de **limite lateral** surge quando estamos interessados em entender o comportamento de uma função ao se aproximar de um ponto por um lado específico (esquerda ou direita). Denotamos o limite lateral à direita de a como $\lim_{x \to a^+} f(x)$ e à esquerda como $\lim_{x \to a^-} f(x)$.

Para que o limite $\lim_{x \to a} f(x)$ exista, é necessário que os limites laterais sejam iguais, conforme definido na Equação 2:

$$\lim_{x \to a^+} f(x) = \lim_{x \to a^-} f(x) \tag{2}$$

2.2.2 PROPRIEDADES DOS LIMITES

Os limites possuem diversas propriedades importantes que facilitam seu cálculo em diferentes contextos:

- **Propriedade da soma**, conforme Equação 3:

$$\lim_{x \to a}[f(x) + g(x)] = \lim_{x \to a} f(x) + \lim_{x \to a} g(x) \tag{3}$$

Exemplo: Considere as funções $f(x) = x^2$ e $g(x) = 3x$. Queremos calcular o limite de $f(x) + g(x)$ conforme x tende a 2:

$$\lim_{x \to 2}[x^2 + 3x] = \lim_{x \to 2} x^2 + \lim_{x \to 2} 3x = 4 + 6 = 10$$

- **Propriedade do produto**, expressa pela Equação 4:

$$\lim_{x \to a}[f(x) \cdot g(x)] = \lim_{x \to a} f(x) \cdot \lim_{x \to a} g(x) \tag{4}$$

Exemplo: Suponha que $f(x) = x + 1$ e $g(x) = 2x$. Calculando o limite do produto $f(x) \cdot g(x)$ conforme $x \to 3$:

$$\lim_{x \to 3}[(x+1) \cdot 2x] = \lim_{x \to 3}(x+1) \cdot \lim_{x \to 3} 2x = 4 \cdot 6 = 24$$

- **Propriedade do quociente** (desde que o denominador não tenda a zero), conforme Equação 5:

$$\lim_{x \to a}\left[\frac{f(x)}{g(x)}\right] = \frac{\lim_{x \to a} f(x)}{\lim_{x \to a} g(x)} \tag{5}$$

Exemplo: Considere as funções $f(x) = x^2 - 1$ e $g(x) = x - 1$. Queremos calcular o limite do quociente conforme $x \to 2$:

$$\lim_{x \to 2}\left[\frac{x^2 - 1}{x - 1}\right] = \frac{\lim_{x \to 2}(x^2 - 1)}{\lim_{x \to 2}(x - 1)} = \frac{3}{1} = 3$$

Essas propriedades dos limites simplificam o cálculo de funções compostas e são fundamentais para a análise de funções em diferentes áreas, como física, economia e engenharia.

2.2.3 SCRIPT PYTHON DE OPERAÇÕES BÁSICAS COM LIMITES

```python
# Propriedade: Soma
from sympy import symbols, limit
x = symbols('x')
f = x**2
g = 3*x
limite_soma = limit(f + g, x, 2)
print(limite_soma) # Saida: 10

# Propriedade: Produto
f = x + 1
g = 2*x
limite_produto = limit(f * g, x, 3)
print(limite_produto) # Saida: 24

# Propriedade: Quociente
f = x**2 - 1
g = x - 1
limite_quociente = limit(f / g, x, 2)
print(limite_quociente) # Saida: 3
```

Note que o *script* acima apresenta a estruturação da solução analítica em modo computacional utilizando bibliotecas peculiares à linguagem em destaque.

2.3 DERIVADAS

A **derivada** de uma função $f(x)$ é uma medida quantitativa de como a função varia em relação à sua variável independente x. Intuitivamente, a derivada expressa a inclinação da reta tangente à curva da função em um ponto específico, indicando a taxa de variação instantânea da função nesse ponto. A definição formal de uma derivada é dada pelo limite do quociente de diferença, conforme expressa na Equação 6:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{6}$$

Este limite representa a inclinação da reta tangente à curva de $f(x)$ em um ponto. Se a derivada existir para todos os pontos de $f(x)$, dizemos que a *função é diferenciável nesse intervalo*. As derivadas são fundamentais para a modelagem de processos dinâmicos, como velocidade e aceleração em sistemas físicos.

2.3.1 PROPRIEDADES DAS DERIVADAS

As derivadas seguem várias propriedades úteis, que facilitam o cálculo de derivadas de funções mais complexas a partir de funções mais simples.

- **Linearidade**: A derivada de uma soma ponderada de funções é a soma ponderada das derivadas dessas funções. Se $f(x)$ e $g(x)$ são diferenciáveis e a e b são constantes, então temos a seguinte propriedade expressa na Equação 7:

$$\frac{d}{dx}[af(x) + bg(x)] = af'(x) + bg'(x) \tag{7}$$

  Exemplo: Suponha que $f(x) = x^2$ e $g(x) = 3x$, e que $a = 2$ e $b = 5$. Então:

$$\frac{d}{dx}[2x^2 + 5(3x)] = 2 \cdot 2x + 5 \cdot 3 = 4x + 15$$

- **Regra do Produto**: A derivada do produto de duas funções é dada pela regra do produto, expressa na Equação 8, que afirma:

$$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + f(x)g'(x) \tag{8}$$

  Exemplo: Considere $f(x) = x^2$ e $g(x) = \sin(x)$. A derivada do produto $x^2 \sin(x)$ será:

$$\frac{d}{dx}[x^2 \sin(x)] = 2x \sin(x) + x^2 \cos(x)$$

- **Regra da Cadeia**: Quando temos uma função composta $f(g(x))$, a derivada da função composta é dada pela regra da cadeia,expressa na Equação 9,:

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x) \tag{9}$$

Exemplo: Suponha que $f(u) = u^3$ e $g(x) = \cos(x)$, então $f(g(x)) = (\cos(x))^3$. A derivada de $(\cos(x))^3$ será:

$$\frac{d}{dx}[\cos^3(x)] = 3\cos^2(x) \cdot (-\sin(x)) = -3\cos^2(x)\sin(x)$$

Essas propriedades das derivadas simplificam o cálculo em uma ampla variedade de problemas práticos, permitindo a análise de fenômenos como a variação de funções complexas e sua aplicação em otimização e modelagem.

2.3.2 SCRIPT PYTHON DE OPERAÇÕES BÁSICAS COM DERIVADAS

```python
import sympy as sp
# Exemplo: Linearidade
x = sp.symbols('x')
f = 2*x**2 + 15*x
derivative = sp.diff(f, x)
print(derivative) # Saida: 4*x + 15

# Exemplo: Regra do produto
f = x**2
g = sp.sin(x)
product_derivative = sp.diff(f * g, x)
print(product_derivative) # Saida: 2*x*sin(x) + x**2*cos(x)

# Exemplo: Regra da cadeira
u = sp.cos(x)
f_u = u**3
chain_derivative = sp.diff(f_u, x)
print(chain_derivative) # Saida: -3*cos(x)**2*sin(x)
```

2.4 INTEGRAIS

A **integração**, ou cálculo integral, é o processo inverso da diferenciação. Ela é usada para acumular valores ao longo de um intervalo, sendo fundamental para problemas que envolvem áreas sob curvas, volumes de sólidos de revolução e até mesmo em equações diferenciais. A integral de uma função $f(x)$ no intervalo $[a,b]$ é definida como na Equação 10:

$$\int_a^b f(x)\,dx \tag{10}$$

A integral definida acumula a área sob a curva de $f(x)$ entre $x = a$ e $x = b$, enquanto a integral indefinida está associada à anti-derivada de $f(x)$. Integrais aparecem em muitos contextos, desde o cálculo de áreas e volumes até a resolução de equações diferenciais que descrevem sistemas dinâmicos.

2.4.1 PROPRIEDADES DAS INTEGRAIS

Assim como as derivadas, as integrais possuem propriedades que facilitam o cálculo, especialmente quando se trata de funções compostas ou definidas por pedaços.

- **Linearidade**: A integral de uma soma ponderada de funções é a soma ponderada das integrais dessas funções. Para funções $f(x)$ e $g(x)$, e constantes a e b, temos a seguinte relação expressa na Equação 11:

$$\int_a^b [af(x)+bg(x)]\,dx = a\int_a^b f(x)\,dx + b\int_a^b g(x)\,dx \tag{11}$$

Exemplo : Suponha $f(x)=x^2$ e $g(x)=x$, e que queremos calcular a integral de $3f(x)+2g(x)$ no intervalo de 0 a 1. Aplicando a linearidade:

$$\int_0^1 [3x^2+2x]\,dx = 3\int_0^1 x^2\,dx + 2\int_0^1 x\,dx$$

Calculando cada integral separadamente:

$$\int_0^1 x^2\,dx = \left[\frac{x^3}{3}\right]_0^1 = \frac{1}{3}$$

$$\int_0^1 x\,dx = \left[\frac{x^2}{2}\right]_0^1 = \frac{1}{2}$$

Agora, substituímos esses valores:

$$3\times\frac{1}{3}+2\times\frac{1}{2}=1+1=2$$

Portanto, a integral de $3x^2+2x$ de 0 a 1 é igual a 2.

- **Adição de Intervalos**: Se quisermos calcular a integral de uma função $f(x)$ em um intervalo maior, podemos dividi-la em subintervalos e somar as integrais nesses subintervalos, conforme expresso na Equação 12.

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx \tag{12}$$

Exemplo: Considere $f(x)=x^2$ e queremos calcular $\int_0^2 x^2\,dx$. Podemos dividir o intervalo de 0 a 2 em dois subintervalos: de 0 a 1 e de 1 a 2. Assim, temos:

$$\int_0^2 x^2\,dx = \int_0^1 x^2\,dx + \int_1^2 x^2\,dx$$

Já sabemos que $\int_0^1 x^2\,dx = \frac{1}{3}$ (do exemplo anterior). Agora, calculamos $\int_1^2 x^2\,dx$:

$$\int_1^2 x^2\,dx = \left[\frac{x^3}{3}\right]_1^2 = \frac{8}{3}-\frac{1}{3}=\frac{7}{3}$$

Somando os dois resultados:

$$\int_0^2 x^2\,dx = \frac{1}{3}+\frac{7}{3}=\frac{8}{3}$$

Portanto, a integral de x^2 de 0 a 2 é $\frac{8}{3}$.

As integrais desempenham um papel crucial na modelagem de fenômenos acumulativos, como o cálculo de trabalho em física, o volume de objetos tridimensionais, e a resolução de equações diferenciais que descrevem sistemas físicos.

2.4.2 SCRIPT PYTHON DE OPERAÇÕES BÁSICAS COM INTEGRAIS

```
import sympy as sp

# Definindo a variavel simbolica
x = sp.symbols('x')

# Definindo as funcoes
f = 3*x**2 + 2*x
g = x**2

# Calculando a integral de 0 a 1 para 3f(x) + 2g(x)
integral_f_g = sp.integrate(f, (x, 0, 1))

# Calculando as integrais separadamente
integral_f = sp.integrate(g, (x, 0, 1))
integral_g = sp.integrate(x, (x, 0, 1))

# Resultados
result_linearidade = 3 * integral_f + 2 * integral_g

# Exibindo os resultados
print("Resultado da integral de 3x^2 + 2x de 0 a 1:", integral_f_g)
print("Resultado da integral de 3x^2 + 2x (usando linearidade):",
    result_linearidade)

# Calculando a integral de x^2 de 0 a 2 usando adicao de intervalos
integral_0_1 = sp.integrate(g, (x, 0, 1))
integral_1_2 = sp.integrate(g, (x, 1, 2))

# Resultado final
result_adicao_intervalos = integral_0_1 + integral_1_2
print("Resultado da integral de x^2 de 0 a 2 usando adicao de intervalos:",
    result_adicao_intervalos)
```

2.5 CONCLUSÃO

Com a compreensão sólida desses conceitos matemáticos, estaremos prontos para explorar técnicas numéricas avançadas nos capítulos seguintes. Aplicaremos essas ferramentas para resolver problemas práticos de engenharia, como simulação de sistemas dinâmicos, otimização e análise estrutural. Códigos adicionais sobre cálculo explorando outras bibliotecas estão disponíveis no endereço <https://github.com/jonathacosta/NM/tree/main/NM_codes/Un01>

LISTA DE EXERCÍCIOS - CÁLCULO: LIMITES, DERIVADAS E INTEGRAIS

PARTE 1: LIMITES

1. **Cálculo Direto de Limites** Calcule os limites abaixo:

   a) $\lim_{x\to 3} \frac{x^2-9}{x-3}$
   b) $\lim_{x\to 0} \frac{\sin(5x)}{x}$
   c) $\lim_{x\to\infty} \left(1+\frac{1}{x}\right)^x$
   d) $\lim_{x\to -2} \frac{\sqrt{4+x}-2}{x+2}$

2. **Limites Laterais** Verifique se o limite existe e calcule-o:

   a) $\lim_{x\to 0^+} \frac{1}{x}$
   b) $\lim_{x\to 0^-} \frac{1}{x}$
   c) Explique por que o limite $\lim_{x\to 0} \frac{1}{x}$ não existe com base nos limites laterais calculados.

3. **Limites e Funções Trigonométricas** Encontre o limite:

   a) $\lim_{x\to 0} \frac{\tan(x)-\sin(x)}{x^3}$
   b) $\lim_{x\to 0} \frac{\sin^2(x)}{x^2}$

PARTE 2: DERIVADAS

4. **Derivação Direta** Calcule a derivada das funções abaixo:

   a) $f(x) = x^3 - 4x + 2$
   b) $g(x) = \ln(x^2+1)$
   c) $h(x) = e^{3x} \cdot \sin(x)$

5. **Derivadas de Ordem Superior** Determine a segunda e terceira derivada de $f(x) = x^4 - 3x^2 + x$.

6. **Derivada Implícita** Encontre $\frac{dy}{dx}$ se $x^2 + y^2 = 25$.

7. **Problemas de Otimização** Um fabricante deseja produzir uma lata cilíndrica de volume fixo $V = 500\,\text{cm}^3$. Determine as dimensões (altura h e raio r) que minimizam a quantidade de material utilizada para a construção da lata.

8. **Taxas Relacionadas** Uma escada de 10 metros de comprimento está encostada em uma parede. Se a base da escada escorrega horizontalmente para longe da parede a uma taxa de 1 m/s, com que velocidade a parte superior da escada desce a parede quando a base está a 6 metros da parede?

PARTE 3: INTEGRAIS

9. **Integração Direta** Calcule as integrais indefinidas e defina as constantes de integração:
   a) $\int (3x^2 - 4x + 1)\,dx$
   b) $\int \frac{1}{x \ln(x)}\,dx$
   c) $\int e^{2x} \sin(x)\,dx$
10. **Integrais Definidas** Calcule as integrais definidas abaixo:
    a) $\int_0^1 (x^3 - 3x^2 + 2)\,dx$
    b) $\int_1^e \frac{\ln(x)}{x}\,dx$
    c) $\int_0^\pi x \sin(x)\,dx$
11. **Aplicação da Integral**
    a) Encontre a área da região limitada pelas curvas $y = x^2$ e $y = 4$.
    b) Determine o volume do sólido gerado pela rotação da região entre $y = x^2$ e $y = 4$ em torno do eixo y.
12. **Integrais Improprias** Avalie, se convergente:
    a) $\int_1^\infty \frac{1}{x^2}\,dx$
    b) $\int_0^\infty e^{-x^2}\,dx$
    c) $\int_0^\infty \frac{\sin(x)}{x}\,dx$

PARTE 4: EXERCÍCIOS AVANÇADOS DE APLICAÇÃO

14. **Modelagem Matemática com Derivadas e Integrais** Considere uma população de peixes em um lago modelada pela função $P(t) = 5000e^{0.1t}$, onde t é o tempo em anos.
    a) Encontre a taxa de crescimento da população no instante $t = 5$ anos.
    b) Determine o tempo necessário para que a população atinja 10.000 peixes.
15. **Análise de Convergência e Divergência** Discuta a convergência dos limites e das integrais abaixo, apresentando uma justificativa teórica:
    a) $\lim_{x\to\infty} \frac{\ln(x)}{x}$
    b) $\int_1^\infty \frac{\ln(x)}{x^2}\,dx$
    c) $\sum_{n=1}^\infty \frac{1}{n^{1.5}}$

3 Álgebra: Fundamentos Matemáticos

Neste capítulo, revisaremos os principais conceitos matemáticos que servem como base para os estudos de MN. Estes incluem álgebra linear, além de algumas de suas propriedades essenciais. Estes tópicos são cruciais para a aplicação eficaz de métodos numéricos em problemas práticos de engenharia.

3.1 VETORES

Os vetores são elementos fundamentais na álgebra e na matemática aplicada, especialmente no estudo de métodos numéricos. Em geral, um vetor pode ser entendido como uma sequência de valores (ou elementos) dispostos em uma única linha (vetor linha) ou em uma única coluna (vetor coluna). Eles são amplamente utilizados em operações algébricas, análise numérica e computação científica.

3.1.1 DEFINIÇÃO DE VETOR

Um vetor é definido como uma lista ordenada de números, que podem representar quantidades físicas ou matemáticas. O vetor é representado por uma letra minúscula com uma seta em cima, como $\vec{v}$, ou, em notação matricial, por v.

Exemplo de vetor coluna:

$$\vec{v} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

Exemplo de vetor linha:

$$\vec{w} = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix}$$

3.1.2 OPERAÇÕES COM VETORES

3.1.2.1 Adição e Subtração de Vetores

A adição e subtração de vetores é realizada de forma componente a componente. Para que essas operações sejam possíveis, os vetores envolvidos devem ter a mesma dimensão.

Exemplo Numérico:

Sejam os vetores $\vec{a} = \begin{bmatrix} 3 \\ 5 \\ 7 \end{bmatrix}$ e $\vec{b} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$. Então:

$$\vec{a} + \vec{b} = \begin{bmatrix} 3+1 \\ 5+2 \\ 7+4 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 11 \end{bmatrix}$$

3.1.2.2 Multiplicação de Vetor por Escalar

A multiplicação de um vetor por um escalar α é realizada multiplicando cada componente do vetor pelo valor de α.

Exemplo Numérico:

Para o vetor $\vec{c} = \begin{bmatrix} 2 \\ -3 \\ 4 \end{bmatrix}$ e o escalar $\alpha = 3$, temos:

$$\alpha \cdot \vec{c} = 3 \cdot \begin{bmatrix} 2 \\ -3 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \\ 12 \end{bmatrix}$$

3.1.2.3 Produto Escalar (ou Produto Interno)

O produto escalar entre dois vetores $\vec{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}$ e $\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$ é dado pela soma do produto de seus elementos correspem quentes. O produto escalar é representado por $\vec{u} \cdot \vec{v}$.

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Exemplo Numérico:

Se $\vec{u} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ e $\vec{v} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$, então:

$$\vec{u} \cdot \vec{v} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32$$

3.1.2.4 Produto Vetorial (em 3 dimensões)

O produto vetorial é uma operação definida apenas em três dimensões, que resulta em um vetor perpendicular a ambos os vetores iniciais. Para dois vetores $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ e $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$, o produto vetorial $\vec{u} \times \vec{v}$ é calculado como:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} = \hat{\imath}(u_2 v_3 - u_3 v_2) - \hat{\jmath}(u_1 v_3 - u_3 v_1) + \hat{k}(u_1 v_2 - u_2 v_1)$$

Exemplo Numérico:

Para $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ e $\vec{v} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix} = \hat{\imath}(2 \cdot 6 - 3 \cdot 5) - \hat{\jmath}(1 \cdot 6 - 3 \cdot 4) + \hat{k}(1 \cdot 5 - 2 \cdot 4)$$

$$= \hat{\imath}(12 - 15) - \hat{\jmath}(6 - 12) + \hat{k}(5 - 8)$$

$$= -3\hat{\imath} + 6\hat{\jmath} - 3\hat{k} = \begin{bmatrix} -3 \\ 6 \\ -3 \end{bmatrix}$$

3.1.3 NORMA DE UM VETOR

A norma (ou módulo) de um vetor $\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$ é uma medida de seu comprimento, dada por:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

Exemplo Numérico:
Para o vetor $\vec{v} = \begin{bmatrix} 3 & 4 \end{bmatrix}$, temos:

$$\|\vec{v}\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

3.1.4 PROPRIEDADES DOS VETORES

Algumas propriedades úteis de vetores incluem:

- **Comutatividade da Adição**: $\vec{u} + \vec{v} = \vec{v} + \vec{u}$.
- **Associatividade da Adição**: $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$.
- **Distributividade**: $\alpha(\vec{u} + \vec{v}) = \alpha\vec{u} + \alpha\vec{v}$.

3.1.5 SCRIPT PYTHON DE OPERAÇÕES BÁSICAS COM VETORES

O código a seguir ilustra como resolver os exemplos dos vetores utilizando Python® e a biblioteca *NumPy*:

```
import numpy as np

# Exemplo de Adicao de Vetores
a = np.array([3, 5, 7])
b = np.array([1, 2, 4])
soma = a + b
print("Soma de vetores a + b:", soma)

# Exemplo de Multiplicacao de Vetor por Escalar
c = np.array([2, -3, 4])
alpha = 3
produto_escalar = alpha * c
print("Multiplicacao do vetor c por escalar alpha:", produto_escalar)

# Exemplo de Produto Escalar (ou Produto Interno)
u = np.array([1, 2, 3])
v = np.array([4, 5, 6])
produto_interno = np.dot(u, v)
print("Produto escalar de u e v:", produto_interno)

# Exemplo de Produto Vetorial
u = np.array([1, 2, 3])
v = np.array([4, 5, 6])
produto_vetorial = np.cross(u, v)
print("Produto vetorial de u e v:", produto_vetorial)
```

```
# Exemplo de Norma de um Vetor
vetor = np.array([3, 4])
norma = np.linalg.norm(vetor)
print("Norma do vetor [3, 4]:", norma)
```

3.2 MATRIZES

As matrizes são um conceito fundamental na álgebra linear e são amplamente utilizadas em métodos numéricos para resolver sistemas de equações, representar transformações lineares e em diversas aplicações científicas e engenheiras. Uma matriz é uma tabela retangular de números, organizada em linhas e colunas.

3.2.1 DEFINIÇÃO DE MATRIZ

Uma matriz A é representada como $A = [a_{ij}]$, em que a_{ij} é o elemento da matriz que está na i-ésima linha e j-ésima coluna. As dimensões de uma matriz são dadas pelo número de linhas e colunas, expressas como $m \times n$, em que m é o número de linhas e n é o número de colunas.

Exemplo de Matriz:

Uma matriz 2×3:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

3.2.2 OPERAÇÕES COM MATRIZES

3.2.2.1 Adição e Subtração de Matrizes

A adição e subtração de matrizes é realizada da mesma forma que a de vetores, ou seja, somando ou subtraindo os elementos correspem quentes. Para que essas operações sejam possíveis, as matrizes devem ter as mesmas dimensões.

Exemplo Numérico:

Sejam as matrizes $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$. Então:

$$C = A + B = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

3.2.2.2 Multiplicação de Matrizes

A multiplicação de matrizes é um pouco mais complexa. Para multiplicar duas matrizes A e B, o número de colunas de A deve ser igual ao número de linhas de B. O elemento c_{ij} da matriz resultante C é obtido pela soma dos produtos dos elementos da i-ésima linha de A pelos elementos da j-ésima coluna de B.

Exemplo Numérico:

Seja $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$:

$$C = A \cdot B = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

3.2.3 SCRIPT PYTHON DE OPERAÇÕES COM MATRIZES

```
# Definicao de matrizes
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Exemplo de Adicao de Matrizes
C = A + B
print("Soma de matrizes A + B:\n", C)

# Exemplo de Subtracao de Matrizes
D = A - B
print("Subtracao de matrizes A - B:\n", D)

# Exemplo de Multiplicacao de Matrizes
E = np.dot(A, B)
print("Multiplicacao de matrizes A * B:\n", E)
```

3.3 DETERMINANTES

Os determinantes são uma ferramenta fundamental na álgebra linear, proporcionando informações importantes sobre as matrizes, como a invertibilidade, o volume e a solução de sistemas de equações lineares. Um determinante é um número associado a uma matriz quadrada, que pode ser calculado por diferentes métodos, dependendo da dimensão da matriz.

3.3.1 DEFINIÇÃO DE DETERMINANTE

Para uma matriz quadrada A de ordem n, o determinante é denotado como $\det(A)$ ou $|A|$. Para matrizes de 2×2 e 3×3, as definições são as seguintes:

- Para uma matriz 2×2:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \Rightarrow \quad \det(A) = ad - bc$$

- Para uma matriz 3×3:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \Rightarrow \quad \det(A) = a(ei - fh) - b(di - fg) + c(dh - eg)$$

Exemplo Numérico:

Para a matriz $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$:

$$\det(A) = (1)(4) - (2)(3) = 4 - 6 = -2$$

Para a matriz $B = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}$:

$$\det(B) = 1(1\cdot 0 - 4\cdot 6) - 2(0\cdot 0 - 4\cdot 5) + 3(0\cdot 6 - 1\cdot 5) = 1(0-24) - 2(0-20) + 3(0-5)$$
$$= -24 + 40 - 15 = 1$$

3.3.2 PROPRIEDADES DOS DETERMINANTES

Os determinantes possuem várias propriedades importantes:

- **Determinante da Matriz Identidade**: $\det(I) = 1$, em que I é a matriz identidade.
- **Matriz Escalar**: Se A é uma matriz $n \times n$ e k é um escalar, então $\det(kA) = k^n \cdot \det(A)$.
- **Matriz Transposta**: $\det(A^T) = \det(A)$.
- **Multiplicação de Matrizes**: $\det(AB) = \det(A) \cdot \det(B)$.
- **Troca de Linhas**: Trocar duas linhas de uma matriz muda o sinal do determinante.
- **Linha Nula**: Se uma linha (ou coluna) de uma matriz é composta apenas por zeros, então $\det(A) = 0$.
- **Dependência Linear**: Se duas linhas (ou colunas) de uma matriz são iguais, então $\det(A) = 0$.

3.3.3 SCRIPT PYTHON PARA CÁLCULO DE DETERMINANTES E INVERSAS

Abaixo é apresentado um *script* em Python® utilizando a biblioteca NumPy para calcular os determinantes das matrizes mencionadas:

```
import numpy as np

# Definicao das matrizes
A = np.array([[1, 2], [3, 4]])
B = np.array([[1, 2, 3], [0, 1, 4], [5, 6, 0]])

# Exemplo de Determinante para matriz 2x2
det_A = np.linalg.det(A)
print("Determinante da matriz A:\n", det_A)

# Exemplo de Determinante para matriz 3x3
det_B = np.linalg.det(B)
print("Determinante da matriz B:\n", det_B)
```

3.4 CONCLUSÃO

A Álgebra fornece a base para diversos métodos numéricos, possibilitando manipular e resolver sistemas lineares, além de fornecer propriedades essenciais para simplificar cálculos. O entendimento das operações com matrizes, determinantes e propriedades algébricas é fundamental para o desenvolvimento de algoritmos numéricos eficientes. Códigos adicionais sobre cálculo explorando outras bibliotecas estão disponíveis no endereço <https://github.com/jonathacosta/NM/tree/main/NM_codes/Un01>

EXERCÍCIOS DE ÁLGEBRA LINEAR: VETORES, MATRIZES E DETERMINANTES

PARTE 1: VETORES

1. **Operações com Vetores** Considere os vetores $\vec{a} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$ e $\vec{b} = \begin{bmatrix} -1 \\ 4 \\ 2 \end{bmatrix}$. Calcule:

   a) $2\vec{a} + 3\vec{b}$

   b) O produto escalar $\vec{a} \cdot \vec{b}$

   c) O módulo de $\vec{a}$ e $\vec{b}$

2. **Vetores Ortogonais** Verifique se os vetores $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ e $\vec{v} = \begin{bmatrix} 4 \\ -8 \\ 0 \end{bmatrix}$ são ortogonais.

3. **Produto Vetorial** Encontre o produto vetorial entre os vetores $\vec{c} = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$ e $\vec{d} = \begin{bmatrix} 1 \\ 4 \\ -1 \end{bmatrix}$.

PARTE 2: MATRIZES

4. **Operações com Matrizes** Considere as matrizes $A = \begin{bmatrix} 2 & 0 \\ -1 & 3 \end{bmatrix}$ e $B = \begin{bmatrix} 1 & 4 \\ 2 & -1 \end{bmatrix}$. Calcule:

   a) $A + B$

   b) $3A - 2B$

   c) O produto AB

5. **Matriz Transposta e Simetria** Dada a matriz $C = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 4 & 6 \end{bmatrix}$:

   a) Encontre a matriz transposta C^T

   b) Verifique se C é uma matriz simétrica.

6. **Inversa de uma Matriz** Determine se a matriz $D = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$ é invertível. Caso seja, calcule D^{-1}.

PARTE 3: DETERMINANTES

7. **Cálculo de Determinantes**
   Calcule o determinante das seguintes matrizes:

   a) $\begin{bmatrix} 3 & 2 \\ -1 & 4 \end{bmatrix}$

   b) $\begin{bmatrix} 1 & 0 & 2 \\ -3 & 5 & 1 \\ 4 & -2 & 3 \end{bmatrix}$

8. **Determinantes e Dependência Linear** Verifique se as colunas da matriz $E = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ são linearmente dependentes usando o determinante.

9. **Teorema de Laplace** Calcule o determinante da matriz $F = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & 4 \\ 0 & -2 & 5 \end{bmatrix}$, expandindo em relação à primeira linha.

4 Sistemas de Numeração

Os sistemas de numeração são a base da representação dos números e formam a estrutura essencial para a computação moderna. Diferentes sistemas de numeração foram desenvolvidos ao longo da história, cada um com suas próprias características e usos. Este capítulo discute os conceitos fundamentais dos sistemas de numeração, com foco em quatro sistemas amplamente utilizados: decimal, binário, octal e hexadecimal.

4.1 CONCEITUAÇÃO DE SISTEMAS DE NUMERAÇÃO

Um sistema de numeração é uma forma de representar números utilizando um conjunto finito de símbolos. Cada sistema é baseado em uma **base**, que determina o número de símbolos distintos que podem ser usados, e na posição dos dígitos, que corresponde a potências da base. Em geral, qualquer número em um sistema de base b pode ser representado da seguinte forma:

$$N = d_n \times b^n + d_{n-1} \times b^{n-1} + \cdots + d_1 \times b^1 + d_0 \times b^0$$

Em que:

- b é a base do sistema de numeração (por exemplo, 2 no binário, 10 no decimal, etc.);
- d_i são os dígitos da representação numérica, com $0 \leq d_i < b$;
- n é o expoente máximo, que corresponde à posição do dígito mais à esquerda.

A base define o número de dígitos possíveis, e cada dígito multiplica uma potência de b, dependendo de sua posição. Quanto mais à esquerda o dígito está, maior a potência de b que o acompanha.

4.1.1 SISTEMA DECIMAL (BASE 10)

O sistema decimal é o mais familiar para os seres humanos, pois é o sistema que usamos no dia a dia. Ele utiliza dez dígitos (0 a 9) e a posição de cada dígito tem um valor baseado em potências de 10.

Por exemplo, o número 345 no sistema decimal é expresso como:

$$345 = 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

4.1.2 SISTEMA BINÁRIO (BASE 2)

O sistema binário é amplamente utilizado em computadores e sistemas digitais. Ele utiliza apenas dois dígitos, 0 e 1, e cada posição representa uma potência de 2.

Por exemplo, o número binário 1011 é expresso como:

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

4.1.3 SISTEMA OCTAL (BASE 8)

O sistema octal utiliza oito dígitos (0 a 7) e cada posição tem um valor baseado em potências de 8. Esse sistema é menos comum, mas ainda é utilizado em algumas aplicações digitais, especialmente na eletrônica.

Por exemplo, o número octal 73 é expresso como:

$$73_8 = 7 \times 8^1 + 3 \times 8^0 = 59_{10}$$

4.1.4 SISTEMA HEXADECIMAL (BASE 16)

O sistema hexadecimal utiliza 16 dígitos, que incluem os números de 0 a 9 e as letras A, B, C, D, E e F, representando os valores de 10 a 15. Esse sistema é muito utilizado em programação e eletrônica devido à sua compactação eficiente de números binários.

Por exemplo, o número hexadecimal 1A3F é expresso como:

$$1A3F_{16} = 1 \times 16^3 + A \times 16^2 + 3 \times 16^1 + F \times 16^0 = 1 \times 16^3 + 10 \times 16^2 + 3 \times 16^1 + 15 \times 16^0 = 6719_{10}$$

4.2 CONVERSÃO ENTRE SISTEMAS DE NUMERAÇÃO

4.2.1 CONVERSÃO DE DECIMAL PARA BINÁRIO

Para converter um número decimal para binário, utilizamos divisões sucessivas por 2 e anotamos os restos. Então, reconstruímos os numero a partir do último quociente e em direção ao primeiro resto das sucessivas divisões.

Por exemplo, para converter o número decimal 13 para binário:

$$13 \div 2 = 6 \text{ resto } \mathbf{1}$$

$$6 \div 2 = 3 \text{ resto } \mathbf{0}$$

$$3 \div 2 = 1 \text{ resto } \mathbf{1}$$

$$1 \div 2 = \mathbf{0} \text{ resto } \mathbf{1}$$

O número ‘n’ em binário é, portanto, ‘**qrrrr**’. Desse modo, o número 13 em binário é 01101.

4.2.2 CONVERSÃO DE BINÁRIO PARA DECIMAL

Para converter um número binário para decimal, multiplicamos cada dígito por sua respectiva potência de 2 e somamos os resultados.

Por exemplo, para converter o número binário 1101 para decimal:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

4.2.3 CONVERSÃO DE DECIMAL PARA HEXADECIMAL

A conversão de decimal para hexadecimal segue um processo similar ao de decimal para binário, mas usamos divisões sucessivas por 16.

Por exemplo, para converter o número decimal 6719 para hexadecimal:

$$6719 \div 16 = 419 \text{ resto } \mathbf{15}\,(F)$$

$$419 \div 16 = 26 \text{ resto } \mathbf{3}\,(3)$$

$$26 \div 16 = 1 \text{ resto } \mathbf{10}\,(A)$$

$$1 \div 16 = 0 \text{ resto } \mathbf{1}\,(1)$$

Assim, o número 6719 em hexadecimal é 1A3F.

4.2.4 CONVERSÃO DE HEXADECIMAL PARA DECIMAL

A conversão de hexadecimal para decimal segue o processo de multiplicar cada dígito pelo valor correspondente da potência de 16.

Por exemplo, para converter o número hexadecimal 1A3F para decimal:

$$1A3F_{16} = 1 \times 16^3 + A \times 16^2 + 3 \times 16^1 + F \times 16^0 = 6719_{10}$$

4.2.5 PASSOS PARA A CONVERSÃO DE UM NÚMERO FRACIONÁRIO DE BASE B PARA DECIMAL

- Para cada dígito à direita da vírgula, multiplicamos o dígito por b^{-n}, onde b é a base do sistema e n é a posição do dígito. A 1ª posição após a vírgula corresponde a b^{-1}, a 2ª posição a b^{-2}, e assim por diante.
- Somamos todos os valores obtidos para chegar ao resultado final em decimal.

Exemplo Genérico: Converter $0.d_1d_2d_3\ldots_b$ para decimal.

$$0.d_1d_2d_3\ldots_b = d_1 \times b^{-1} + d_2 \times b^{-2} + d_3 \times b^{-3} + \ldots$$

Exemplo Específico em Base 3: Converter 0.2103_3 para decimal.

$$0.2103_3 = 2 \times 3^{-1} + 1 \times 3^{-2} + 0 \times 3^{-3} + 3 \times 3^{-4}$$
$$0.2103_3 = \frac{2}{3} + \frac{1}{9} + 0 + \frac{3}{81}$$
$$0.2103_3 = 0.6667 + 0.1111 + 0 + 0.0370 = 0.8148_{10}$$

Esses passos podem ser seguidos para converter qualquer número fracionário de uma base b para o sistema decimal.

4.3 CONVERSÃO DE SISTEMAS COM VÍRGULA

A conversão de números compostos por uma parte inteira e uma parte fracionária é realizada separadamente para cada parte. Primeiramente, a parte inteira é convertida utilizando o processo de conversão para números inteiros descrito anteriormente. Em seguida, a parte fracionária é convertida de acordo com o método específico para frações. Após a conversão de ambas as partes, os resultados são combinados para formar o número completo no sistema de destino.

Matematicamente, isso pode ser expresso da seguinte forma:

$$\text{número convertido} = \text{parte inteira convertida} + \text{parte fracionária convertida}$$

Dessa forma, o número final no sistema de destino será a justaposição da parte inteira e da parte fracionária já convertidas.

4.3.1 PASSOS PARA A CONVERSÃO FRACIONÁRIO PARA DECIMAL

A conversão de números fracionários (com vírgula) segue um processo semelhante ao utilizado para números inteiros contundo, ao invés de utilizar as potências inteiras da base do sistema, utilizamos potências negativas dessa base para os dígitos à direita da vírgula.

- Para cada dígito à direita da vírgula, multiplicamos o dígito por $base^{-n}$, onde n é a posição do dígito (1ª posição após a vírgula corresponde a $base^{-1}$, 2ª posição corresponde a $base^{-2}$, e assim por diante).
- Somamos todos os valores obtidos para chegar ao resultado final em decimal.

4.3.2 APLICAÇÃO DO PROCEDIMENTO DE CONVERSÃO

Considere, portanto, os exemplos a seguir e o procedimento acima descrito:

a) **Converter** 0.1101_2 **para decimal.**

Parte inteira: 0_2
Parte fracionária: 1101_2
Conversão:

$$0.1101_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$
$$0.1101_2 = \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16}$$
$$0.1101_2 = 0.5_{10} + 0.25_{10} + 0_{10} + 0.0625_{10} = 0.8125_{10}$$

Resultado: $0.1101_2 = 0.8125_{10}$.

b) **Converter** 11.1101_2 **para decimal.**

Parte inteira: 11_2
Parte fracionária: 1101_2
Conversão:

Parte inteira: 11_2

$$11_2 = 1 \times 2^1 + 1 \times 2^0$$

$$11_2 = 2_{10} + 1_{10} = 3_{10}$$

Parte fracionária: 0.1101_2

$$1101_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$
$$1101_2 = \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16}$$
$$1101_2 = 0.5_{10} + 0.25_{10} + 0_{10} + 0.0625_{10} = 0.8125_{10}$$

Sendo: $11_2 = 3_{10}$ e $0.1101_2 = 0.8125_{10}$, tem-se que o resultado de $11.1101_2 = 3.8125_{10}$.

c) Converter $0.1A3_{16}$ para decimal.

Parte inteira: 0_{16}
Parte fracionária: $1A3_{16}$
Conversão:

$$0.1A3_{16} = 1 \times 16^{-1} + A \times 16^{-2} + 3 \times 16^{-3}$$
$$0.1A3_{16} = 1 \times \frac{1}{16} + 10 \times \frac{1}{16^2} + 3 \times \frac{1}{16^3}$$
$$0.1A3_{16} = 0.0625 + 10 \times 0.00390625 + 3 \times 0.00024414$$
$$0.1A3_{16} = 0.0625 + 0.0390625 + 0.00073242 = 0.10229492_{10}$$

Resultado: $0.1A3_{16} = 0.10229492_{10}$.

d) Converter $C.1A3_{16}$ para decimal.

Parte inteira: C_{16}
Parte fracionária: $1A3_{16}$
Conversão:

Parte inteira: C_{16}

$$C_{16} = 12_{10}$$

Note que, no caso do sistema hexadecimal, os dígitos A, B, C, D, E, F no hexadecimal correspondem aos valores $10, 11, 12, 13, 14, 15$ em decimal.

$$0.1A3_{16} = 1 \times 16^{-1} + A \times 16^{-2} + 3 \times 16^{-3}$$
$$0.1A3_{16} = 1 \times \frac{1}{16} + 10 \times \frac{1}{16^2} + 3 \times \frac{1}{16^3}$$
$$0.1A3_{16} = 0.0625 + 10 \times 0.00390625 + 3 \times 0.00024414$$
$$0.1A3_{16} = 0.0625_{10} + 0.0390625_{10} + 0.00073242_{10} = 0.10229492_{10}$$

Sendo: $C_{16} = 12_{10}$ e $1A3_{16} = 0.10229492_{10}$, tem-se que o resultado de $C.1A3_{16} = 12.10229492_{10}$.

4.4 SCRIPT PYTHON PARA CONVERSÃO ENTRE SISTEMAS

O código a seguir ilustra uma forma simplificada e compacta de conversão entre sistemas de numeração no Python, mantendo a funcionalidade de conversão de números inteiros e fracionários.

```
def decimal_para_base(n, base):
      parte_inteira = int(n)
      parte_fracionaria = n - parte_inteira

      # Conversao da parte inteira
      if base == 2:
            inteiro_convertido = bin(parte_inteira)[2:]
      elif base == 8:
            inteiro_convertido = oct(parte_inteira)[2:]
      elif base == 16:
            inteiro_convertido = hex(parte_inteira)[2:].upper()
      else:
            raise ValueError("Base nao suportada. Escolha 2 (binario), 8 (octal) ou
                16 (hexadecimal).")

      # Conversao da parte fracionaria
      fracao_convertida = ""
      while parte_fracionaria > 0 and len(fracao_convertida) < 10: # Limite de
          precisao
            parte_fracionaria *= base
            digito = int(parte_fracionaria)
            fracao_convertida += (hex(digito)[2:].upper() if base == 16 else str(
                digito))
            parte_fracionaria -= digito

      return inteiro_convertido + ('.' + fracao_convertida if fracao_convertida else
          "")

def base_para_decimal(num_str, base):
      if '.' in num_str:
            parte_inteira, parte_fracionaria = num_str.split('.')
      else:
            parte_inteira, parte_fracionaria = num_str, ''

      # Conversao da parte inteira
      dec_inteiro = int(parte_inteira, base)

      # Conversao da parte fracionaria
      dec_fracao = sum(int(digito, base) * (base ** -(i + 1)) for i, digito in
          enumerate(parte_fracionaria))

      return dec_inteiro + dec_fracao

# Testes
numero_decimal = 13.8125
print(f"Decimal {numero_decimal} para binario: {decimal_para_base(numero_decimal, 2)
    }")
print(f"Decimal {numero_decimal} para hexadecimal: {decimal_para_base(numero_decimal
```

```
    , 16)}")
print(f"Decimal {numero_decimal} para octal: {decimal_para_base(numero_decimal, 8)}"
    )

numero_binario = "1101.1101"
print(f"Binario {numero_binario} para decimal: {base_para_decimal(numero_binario, 2)
    }")

numero_hexadecimal = "C.1A3"
print(f"Hexadecimal {numero_hexadecimal} para decimal: {base_para_decimal(
    numero_hexadecimal, 16)}")

numero_octal = "15.63"
print(f"Octal {numero_octal} para decimal: {base_para_decimal(numero_octal, 8)}")
```

Respostas do *script*:

Decimal 13.8125 para binário: 1101.1101

Decimal 13.8125 para hexadecimal: D.D

Decimal 13.8125 para octal: 15.64

Binário 1101.1101 para decimal: 13.8125

Hexadecimal C.1A3 para decimal: 12.102294921875

Octal 15.63 para decimal: 13.796875

4.5 CONCLUSÃO

Compreender os sistemas de numeração e as técnicas de conversão entre eles é fundamental para o trabalho com computadores e algoritmos numéricos. Estes conceitos formam a base para uma compreensão mais profunda dos métodos numéricos abordados nos capítulos subsequentes deste livro.

LISTA DE EXERCÍCIOS: SISTEMAS DE NUMERAÇÃO

1. **Conversão de Bases**
   a) Converta o número 1011_2 (base 2) para a base 10.
   b) Converta o número $A3F_{16}$ (base 16) para a base 10.
   c) Converta o número 345_8 (base 8) para a base 10.
2. **Conversão para Diferentes Bases**
   a) Converta o número 205 da base 10 para a base 2.
   b) Converta o número 147 da base 10 para a base 8.
   c) Converta o número 255 da base 10 para a base 16.
3. **Operações em Bases Diferentes**
   a) Calcule $1101_2 + 101_2$ e apresente o resultado em base 2.
   b) Calcule $345_8 - 127_8$ e apresente o resultado em base 8.
   c) Calcule $A3_{16} + 5F_{16}$ e apresente o resultado em base 16.
4. **Multiplicação e Divisão em Bases Diferentes**
   a) Multiplique $1011_2 \times 110_2$ e apresente o resultado em base 2.
   b) Divida 725_8 por 3_8 e apresente o resultado em base 8.
   c) Multiplique $B2_{16} \times 3_{16}$ e apresente o resultado em base 16.
5. **Complemento de 1 e Complemento de 2 em Base 2**
   a) Encontre o complemento de 1 para o número binário 101010.
   b) Encontre o complemento de 2 para o número binário 100110.
6. **Conversão com Ponto Decimal**
   a) Converta o número decimal 10.75_{10} para a base 2.
   b) Converta o número decimal 23.25_{10} para a base 8.
   c) Converta o número decimal 125.5_{10} para a base 16.

5 Aritmética de Ponto Flutuante e Erros Computacionais

Neste capítulo, discutiremos a aritmética de ponto flutuante utilizada em computadores, como os dados numéricos são armazenados, e os diferentes tipos de erros que podem ocorrer em cálculos numéricos, como erros de arredondamento e de truncamento.

5.1 ARITMÉTICA DE PONTO FLUTUANTE

A aritmética de ponto flutuante é amplamente usada em cálculos numéricos, pois permite representar uma ampla gama de números, desde números muito pequenos até números muito grandes. Em geral, um número de ponto flutuante x pode ser representado como:

$$x = \pm m \times 2^e$$

em que m é a mantissa (ou significando), e é o expoente, e a base 2 é utilizada na maioria das representações de ponto flutuante em computadores modernos.

Os sistemas mais comuns de representação de ponto flutuante seguem um padrão normativo. A Tabela 1 mostra a distribuição dos bits na representação IEEE 754/2008 para números de precisão simples (32 bits) e dupla (64 bits).

Tabela 1 – Distribuição dos bits na representação IEEE 754/2008

Tipo	**Simples (32 bits)**	**Dupla (64 bits)**
Sinal	bit 31	bit 63
Expoente	bits 30-23	bits 62-52
Mantissa	bits 22-0	bits 51-0

Fonte: Autor, 2024

5.2 PASSO-A-PASSO GENERALIZADO PARA CONVERSÃO

O padrão IEEE 754 utiliza uma notação de ponto flutuante para representar números em formato binário. No processo genérico para converter um número decimal, como 81, para as representações de precisão simples (32 bits) e dupla (64 bits) devem seguir o seguinte passo-a-passo:

PASSO 1: REPRESENTAÇÃO BINÁRIA DO NÚMERO

Primeiro, converta o número inteiro decimal para binário. O número 81 em binário é:

$$81_{10} = 1010001_2$$

PASSO 2: REPRESENTAÇÃO NORMALIZADA

No formato IEEE 754, a mantissa é representada em notação normalizada. A notação normalizada desloca o ponto binário de modo que haja exatamente um dígito 1 à esquerda do ponto.

Para o número 81, tem-se:

$$1010001_2 = 1.010001 \times 2^6$$

Aqui, o número foi deslocado 6 posições para a esquerda, então o expoente é 6.

PASSO 3: CÁLCULO DO EXPOENTE COM VIÉS

No formato IEEE 754, o expoente armazenado não é o expoente real, mas sim um expoente com viés (bias). Para precisão simples (32 bits), o viés é 127, e para precisão dupla (64 bits), o viés é 1023.

- Para precisão simples:
$$\text{expoente armazenado} = 6 + 127 = 133$$
- Para precisão dupla:
$$\text{expoente armazenado} = 6 + 1023 = 1029$$

Os expoentes armazenados são 133 em decimal (ou 10000101_2 em binário) para precisão simples, e 1029 em decimal (ou 10000000101_2 em binário) para precisão dupla.

PASSO 4: MONTAGEM DOS BITS

Agora, montamos o número com base nos componentes:

- O bit de sinal é 0, pois 81 é positivo.
- O expoente é armazenado em 8 bits para precisão simples e 11 bits para precisão dupla.
- A mantissa é representada pelos dígitos após o ponto binário.

Representação em Precisão Simples (32 bits)

Montando a representação de precisão simples (32 bits):

- **Sinal**: 0
- **Expoente**: 10000101_2
- **Mantissa**: $01000100000000000000000_2$

Logo, o número 81 em precisão simples é:

$$\boxed{0\ 10000101\ 01000100000000000000000}$$

Representação em Precisão Dupla (64 bits)

Montando a representação de precisão dupla (64 bits):

- **Sinal**: 0
- **Expoente**: 10000000101_2
- **Mantissa**: $0100010000000000000000000000000000000000000000000000_2$

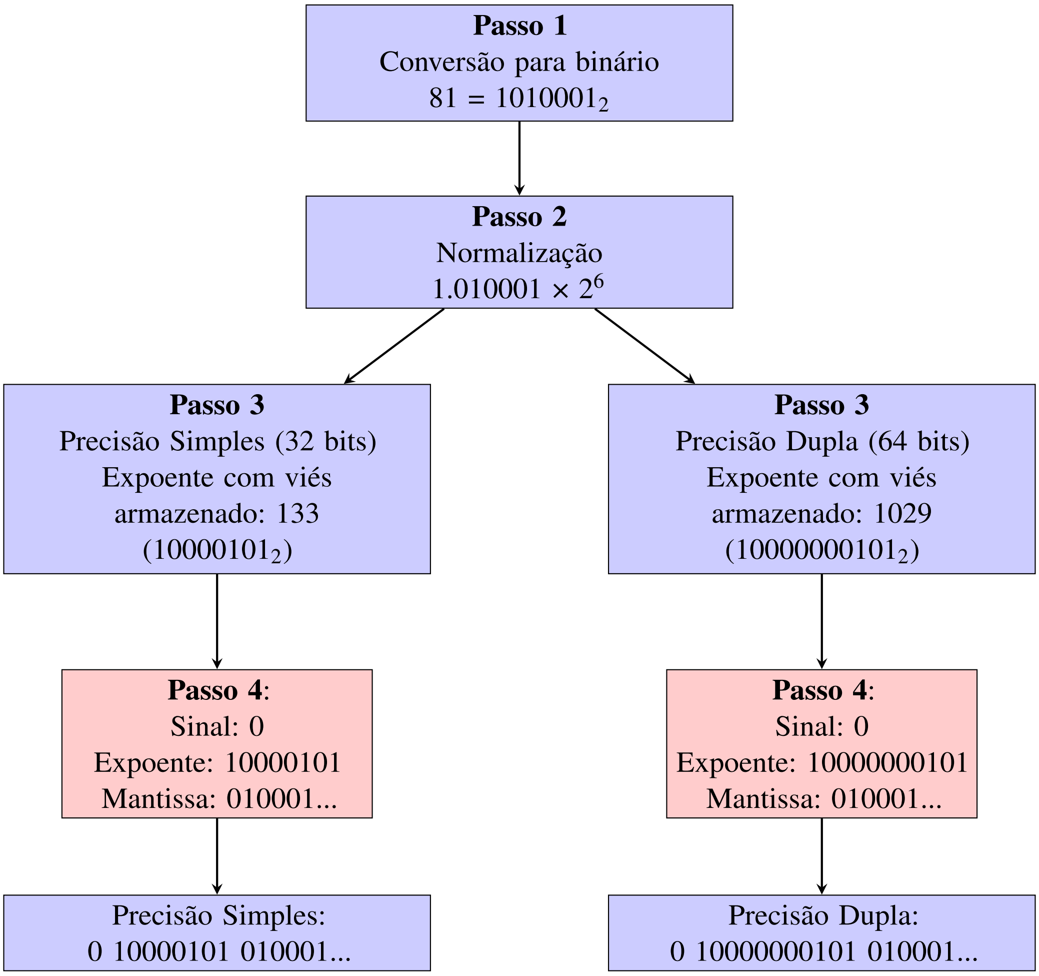
Logo, o número 81 em precisão dupla é:

$$\boxed{0\ 10000000101\ 0100010000000000000000000000000000000000000000000000}$$

5.3 GRAFO DA CONVERSÃO

Na Figura 4 é apresentado o grafo do processo de conversão destacado anteriormente.

Figura 4 – Grafo da conversão pela IEEE 754/2008



Fonte: Autor, (2024)

Note que o valor 81, na Figura 4, é utilizado apenas como exemplo no processo. No entanto, o fluxo segue de forma semelhante para qualquer outro número, respeitando as devidas diferenças nas magnitudes dos valores e correspondências para binário.

5.4 SCRIPT EM PYTHON DE CONVERSÃO PARA PRECISÃO SIMPLES E DUPLA

```
def float_to_binary(num, precision):
"""Converte um numero decimal para ponto flutuante binario."""
    if num < 0:
        sign = 1
        num = -num
    else:
        sign = 0
```

```
    # Encontrar o expoente e a mantissa
    exponent = 0
    while num >= 2:
        num /= 2
        exponent += 1
    while num < 1:
        num *= 2
        exponent -= 1
    # Calcular o expoente com vies
    if precision == 32: # Precisao simples
        bias = 127
    elif precision == 64: # Precisao dupla
        bias = 1023
    else:
        raise ValueError("Precisao deve ser 32 (simples) ou 64 (dupla).")

    exponent += bias
    # Converter expoente para binario
    exponent_bits = bin(exponent)[2:] # Remove o prefixo '0b'
    exponent_bits = exponent_bits.zfill(8 if precision == 32 else 11) # Preenche com zeros a esquerda
    # Calcular a mantissa
    mantissa = num - 1 # Remove o 1 a esquerda do ponto binario
    mantissa_bits = ""
    for _ in range(23 if precision == 32 else 52): # 23 bits para precisao simples, 52 para dupla
        mantissa *= 2
        if mantissa >= 1:
            mantissa_bits += '1'
            mantissa -= 1
        else:
            mantissa_bits += '0'
    # Montar a representacao final
    if precision == 32:
        return f"{sign}{exponent_bits}{mantissa_bits}"
    else:
        return f"{sign}{exponent_bits}{mantissa_bits}"
# Testando a funcao
number = 81
simple_precision = float_to_binary(number, 32)
double_precision = float_to_binary(number, 64)

print(f"Numero: {number}")
print(f"Precisao Simples (32 bits): {simple_precision}")
print(f"Precisao Dupla (64 bits): {double_precision}")
```

5.5 ARMAZENAMENTO DE DADOS NO COMPUTADOR

Os computadores usam sistemas de numeração binários para armazenar dados. Números inteiros são armazenados diretamente em formato binário, enquanto números fracionários e irracionais requerem representações mais complexas, como a de ponto flutuante. Os números armazenados em computadores estão sujeitos a limitações de espaço (bits) e, como resultado, a precisão dos números armazenados é limitada. Esse fato tem implicações em cálculos numéricos, como veremos nas seções sobre erros de arredondamento e truncamento.

5.5.1 LIMITAÇÕES NA REPRESENTAÇÃO

Devido à limitação no número de bits, não é possível armazenar com exatidão todos os números reais. Por exemplo, números irracionais como π e e precisam ser aproximados, assim como frações que não possuem uma representação finita em binário, como $\frac{1}{3}$.

5.6 ERROS DE ARREDONDAMENTO E DE TRUNCAMENTO

Os erros computacionais são inevitáveis devido às limitações da aritmética de ponto flutuante. Dois tipos principais de erros podem ocorrer: erros de arredondamento e erros de truncamento.

5.6.1 ERROS DE ARREDONDAMENTO

Erros de arredondamento ocorrem quando um número não pode ser representado exatamente com o número limitado de bits disponíveis. Por exemplo, ao armazenar o número $\frac{1}{3}$, o computador deve arredondar sua representação para caber em uma quantidade finita de bits. Isso leva a pequenas discrepâncias entre o valor armazenado e o valor exato.

Matematicamente, podemos definir o erro de arredondamento E_r como expresso na Equação 13:

$$E_r = x - \hat{x} \tag{13}$$

em que x é o valor real e $\hat{x}$ é o valor armazenado no computador.

5.6.2 ERROS DE TRUNCAMENTO

Erros de truncamento ocorrem quando um processo de aproximação é interrompido ou truncado. Esses erros são comuns em séries numéricas e métodos iterativos que precisam ser interrompidos após um número finito de passos. Um exemplo típico é a aproximação de derivadas numéricas por diferenças finitas, em que a precisão depende do número de termos utilizados na aproximação.

O erro de truncamento E_t pode ser expresso como na Equação 14:

$$E_t = f(x) - P_n(x) \tag{14}$$

em que $f(x)$ é a função real e $P_n(x)$ é a aproximação da função após n termos.

5.7 CONCLUSÃO

Neste capítulo, exploramos a aritmética de ponto flutuante, a forma como os dados são armazenados em computadores e os erros que surgem em cálculos numéricos. O entendimento desses conceitos é fundamental para a aplicação correta de métodos numéricos, pois eles ajudam a identificar e mitigar erros em cálculos complexos.

6 Derivação Numérica

A derivação numérica é ferramentas fundamentais no campo dos métodos numéricos. Elas permitem a aproximação de derivadas de funções que não podem ser resolvidas analiticamente, ou cuja a solução analítica é demasiamente trabalhosa. Neste capítulo, abordaremos os principais métodos de derivação numérica, com ênfase em suas aplicações e limitações.

6.1 CONCEITUAÇÃO DE DERIVAÇÃO NUMÉRICA

A derivada de uma função $f(x)$ pode ser aproximada numericamente utilizando métodos baseados em diferenças finitas, uma técnica muito útil, especialmente em situações em que a diferenciação analítica, definida na Secção 2.3 é inviável ou excessivamente complexa. A derivação numérica é amplamente empregada para identificar os pontos de máximo e mínimo de uma função e se torna uma ferramenta eficaz quando a função é definida por valores discretos obtidos a partir de dados experimentais ou de medições.

A aproximação da derivada utilizando diferenças finitas baseia-se nos valores da função em diferentes pontos ao redor de $x = a$ para estimar a inclinação da curva. Existem três fórmulas básicas para derivadas finitas, que são as aproximações mais simples para esse tipo de cálculo:

- **Derivação progressiva**: utiliza os valores da função de x_i para x_{i+1}.
- **Derivação regressiva**: emprega os valores de x_i e x_{i-1}.
- **Derivação central**: faz uso dos valores entre x_{i-1} e x_{i+1}.

Esses métodos são fundamentais na derivação numérica e permitem uma estimativa precisa da inclinação da função em cada ponto, mesmo em casos em que a função só está definida em pontos discretos. Nesses métodos, a derivada no ponto (x_i) é calculada a partir do valor de dois pontos. A derivada é estimada como a inclinação da reta que conecta esses dois pontos, conforme também enfatiza (GILAT; SUBRAMANIAM, 2008).

6.2 DERIVAÇÃO PROGRESSIVA

A diferença progressiva é expressa na Equação 15:

$$f'(x) = \left.\frac{df(x)}{dx}\right|_{x=x_i} \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{15}$$

A diferença progressiva (ou para frente) é uma aproximação numérica da derivada de uma função em um ponto x_i. Essa técnica estima a inclinação da tangente à curva da função $f(x)$ em x_i com base nos valores da função em x_i e em um ponto seguinte x_{i+1}. A fórmula apresentada na Equação 15 fornece essa aproximação, em que a derivada $f'(x)$ é aproximada pela razão entre a diferença nos valores da função $f(x_{i+1})$ e $f(x_i)$ e o espaçamento entre os pontos x_{i+1} e x_i.

6.3 DERIVAÇÃO REGRESSIVA

A diferença regressiva (ou para trás) é uma técnica de diferenciação numérica utilizada para aproximar a derivada de uma função em um determinado ponto. Neste método, a derivada da função $f(x)$ no ponto x_i é aproximada utilizando o valor da função no ponto x_{i-1}, que é o ponto imediatamente anterior a x_i.

Essa aproximação é dada pela Equação 16, em que a derivada de $f(x)$ em x_i é calculada pela razão entre a variação da função, $f(x_i) - f(x_{i-1})$, e a variação da variável x, $x_i - x_{i-1}$:

$$f'(x) = \left.\frac{df(x)}{dx}\right|_{x=x_i} \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{16}$$

Este método é particularmente útil quando os dados à frente de x_i não estão disponíveis ou quando se deseja minimizar a influência de valores futuros na aproximação.

6.4 DERIVAÇÃO CENTRADA

A diferença central é uma técnica numérica utilizada para aproximar a derivada de uma função em um ponto, utilizando os valores da função em pontos ao redor. Ao contrário das diferenças progressivas ou regressivas, que consideram apenas pontos à frente ou atrás de x_i, a diferença central faz uso dos valores de $f(x)$ em ambos os lados de x_i, proporcionando uma aproximação mais precisa, especialmente para funções suaves.

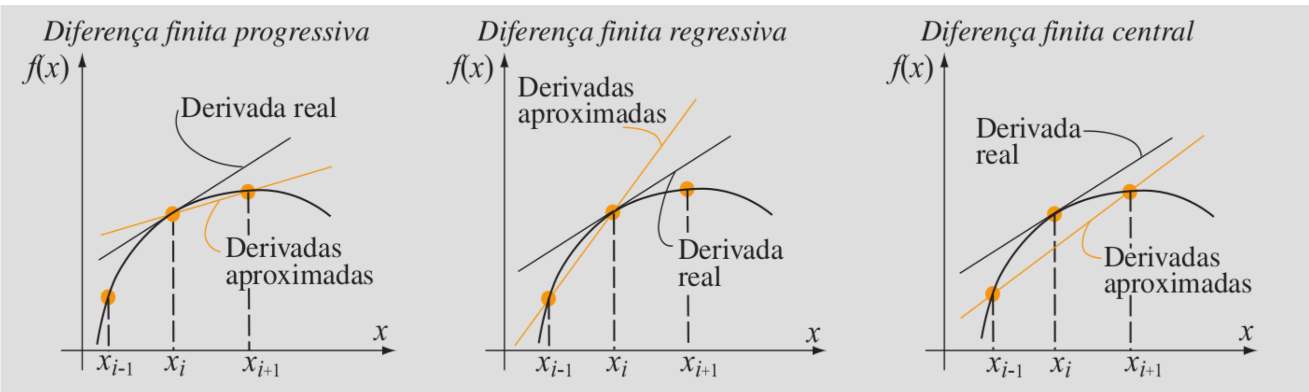
Essa abordagem é representada pela Equação 17, onde a derivada de $f(x)$ no ponto x_i é aproximada pela razão entre a diferença dos valores da função em x_{i+1} e x_{i-1}, e a distância entre esses pontos:

$$f'(x) = \left.\frac{df(x)}{dx}\right|_{x=x_i} \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{17}$$

Este método geralmente oferece uma melhor aproximação da derivada em comparação com as diferenças progressivas ou regressivas, principalmente quando a função é suficientemente suave no intervalo considerado.

Resumidamente, uma percepção gráfica da derivação numérica é apresentada na Figura 5. Note

Figura 5 – Equações de diferença para a derivada primeira



Fonte: GILAT, (2008)

que, a depender do tipo de aproximação utilizada, os resultados apresentarão um erro de magnitude maior ou menor.

6.5 ERRO NA APROXIMAÇÃO DA DERIVAÇÃO

O erro em métodos de diferenças finitas está relacionado com o valor do incremento (h) entre pontos adjacentes. Enquanto um valor menor de h pode reduzir o erro de truncamento, valores muito pequenos podem amplificar erros de arredondamento, conforme demonstrado a seguir na Equação 18:

$$Erro_{relativo} = \left| \frac{f(x)_{aproximado} - f(x)_{real}}{f(x)_{real}} .100 \right| = f(x)\% \tag{18}$$

6.5.1 PERCEPÇÃO NUMÉRICA DO ERRO DE APROXIMAÇÃO

Considere a função $f(x) = x^3$. Calcule numericamente a derivada primeira no ponto $x = 3$, aplicando as fórmulas de diferenças finitas progressiva, regressiva e central, utilizando os pontos $x = 2$, $x = 3$ e $x = 4$. Então, compare os resultados com a derivada exata (analítica).

Solução

1. Aplicando-se a derivação analítica sobre $f(x)$ no ponto $x = 3$, temos:

$$\text{Solução analítica :} \begin{cases} f(x) = x^3 \\ f'(x) = 3 \cdot x^2 \\ f'(3) = 3 \cdot (3)^2 = 27 \\ f'(3) = 27 \end{cases}$$

2. Aplicando-se a derivação numérica no ponto $x = 3$, temos os resultados:

   a) Diferenciação finita progressiva, Equação 15: $\frac{df(x)}{dx}\Big|_{x=3} = \frac{f(4) - f(3)}{4 - 3} = \frac{4^3 - 3^3}{1} = 37$

   b) Diferenciação finita regressiva, Equação 16: $\frac{df(x)}{dx}\Big|_{x=3} = \frac{f(3) - f(2)}{3 - 2} = \frac{3^3 - 2^3}{1} = 19$

   c) Diferenciação finita central, Equação 17: $\frac{df(x)}{dx}\Big|_{x=3} = \frac{f(4) - f(2)}{4 - 2} = \frac{4^3 - 2^3}{2} = 28$

$$\text{Portanto, a derivação numérica :} \begin{cases} progressiva : f'(3) = 37 \\ regressiva : f'(3) = 19 \\ central : f'(3) = 28 \end{cases}$$

3. Analisando-se os erros na aproximação, conforme Equação 18, tem-se :

$$\text{Erros de aproximação:} \begin{cases} progressiva : Erro = \left| \frac{37-27}{27} .100 \right| = 37,04\% \\ regressiva : Erro = \left| \frac{19-27}{27} .100 \right| = 29,63 \\ central : Erro = \left| \frac{28-27}{27} .100 \right| = 3,704\% \end{cases}$$

Neste caso, o erro aproximado é de $37,04\%$ utilizando a derivação progressiva, $29,63\%$ utilizando a derivação regressiva e de apenas $3,704\%$ utilizando a derivação central.

6.6 SCRIPT BÁSICO PYTHON PARA DERIVADAS NUMÉRICAS

```
from autograd import grad
def der_num(x,y,p):
      if p<min(x) or p> max(x):
            print(f"\nPonto 'x={p}' de analise fora do dominio do intervalo!\n")
      return None,None,None

      elif p not in x:
            print(f"\nValor 'x={p}' esta fora da lista.\nUma interpolacao pode ser
                necessaria!\n")
      return None,None,None
      else:
            try:
            # Encontrar o indice de p na lista x
            i = x.index(p)
            dp,dr,dc = None,None,None
            if i==0:
                  dp=( y[i+1] - y[i])/(x[i+1]-x[i])
            elif i==len(x)-1:
                  dr=( y[i] - y[i-1])/(x[i]-x[i-1])
            else:
                  dp=( y[i+1] - y[i])/(x[i+1]-x[i])
                  dr=( y[i] - y[i-1])/(x[i]-x[i-1])
                  dc=( y[i+1] - y[i-1])/(x[i+1]-x[i-1])
      finally:
            return dp,dr,dc

def Resultados(dp,dr,dc,p,f):
      df_dx = grad(f); der = df_dx(float(p))
      print('-'*50, f"\nRESULTADOS para o ponto x='{p}':\n", '-'*50, sep='')
      print("Analitica \tProgr\t \t\tCentral\t \tRegress")
      print(f" {der}\t \t {dp} \t {dc} \t {dr}")
      E=[]
      for derivada_num in [dp,dc,dr]:
            if derivada_num==None: E.append(None)
            else: E.append(round(abs(der-derivada_num)/der,2))
            print(f" Erro(%): \t {E[0]} \t\t\t {E[1]} \t {E[2]}")

if __name__=="__main__":
f=lambda y:y**3
x=[2,3,4,5,6];
y=[f(i) for i in x]
print(x);
for p in [1,5, 2, 3, 6, 6.5]: # Pontos de analise
      dp,dr,dc= der_num(x,y,p)
      Resultados(dp,dr,dc,p,f)
```

6.7 DIFERENÇAS FINITAS POR SÉRIE DE TAYLOR

As fórmulas de diferenças finitas são amplamente utilizadas para aproximar derivadas de funções em pontos discretos. Utilizando expansões de série de Taylor, podemos deduzir fórmulas de diferenças finitas para derivadas de primeira e segunda ordem, aplicáveis em problemas de métodos numéricos.

CONCEITOS BASILARES DA SÉRIE DE TAYLOR

A forma genérica da série de Taylor de uma função $f(x)$ em torno de um ponto $x = a$ é dada por:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n, \tag{19}$$

em que $f^{(n)}(a)$ representa a n-ésima derivada de f avaliada em $x = a$.
ou, de maneira mais expandida,

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots \tag{20}$$

6.7.1 APLICAÇÃO SÉRIE DE TAYLOR ÀS DIFERENÇAS FINITAS

Para uma função $f(x)$ suficientemente diferenciável em torno de um ponto x, a expansão em série de Taylor ao redor deste ponto pode ser expressa como:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots \tag{21}$$

onde h representa um pequeno deslocamento a partir de x. Esta expansão fornece uma ferramenta poderosa para desenvolver aproximações de derivadas, ao manipularmos termos desta série para resolver para $f'(x)$ e $f''(x)$.

6.7.2 APROXIMAÇÃO PARA A DERIVADA PRIMEIRA

Consideremos a derivada primeira de uma função $f(x)$. Podemos obter aproximações para $f'(x)$ reescrevendo a expansão de Taylor para pontos adjacentes e então subtraindo ou somando essas expressões.

Fórmula de Diferença Progressiva

Expandindo $f(x+h)$ em série de Taylor, temos:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + O(h^3) \tag{22}$$

Ao resolvermos para $f'(x)$, obtemos a seguinte aproximação:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{23}$$

Esta é uma fórmula de diferença progressiva de primeira ordem para a derivada primeira de $f(x)$.

Fórmula de Diferença Central

Para melhorar a precisão, utilizamos a fórmula de diferença central, que envolve pontos de ambos os lados de x. Expandindo $f(x+h)$ e $f(x-h)$, temos:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + O(h^3) \tag{24}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - O(h^3) \tag{25}$$

Subtraindo estas equações, obtemos:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{26}$$

Esta é uma fórmula de diferença central de segunda ordem para a derivada primeira, com erro da ordem de $O(h^2)$.

6.7.3 APROXIMAÇÃO PARA A DERIVADA SEGUNDA

Para a derivada segunda de $f(x)$, somamos as expansões de Taylor para $f(x+h)$ e $f(x-h)$.

Fórmula de Diferença Central para a Derivada Segunda

Somando as expansões, obtemos:

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4) \tag{27}$$

Resolvendo para $f''(x)$, temos a seguinte fórmula:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{28}$$

Esta é uma fórmula de diferença central de segunda ordem para a derivada segunda de $f(x)$.

6.7.4 EXEMPLOS DE CÁLCULO NUMÉRICO

Exemplo 1: Aproximação da Derivada Primeira

Suponha que queremos aproximar a derivada da função $f(x) = \sin(x)$ em $x = 0.5$, com $h = 0.1$. Utilizando a fórmula de diferença progressiva, temos:

$$\begin{aligned} f'(0.5) &\approx \frac{f(0.5+0.1) - f(0.5)}{0.1} \\ &= \frac{\sin(0.6) - \sin(0.5)}{0.1} \\ &\approx \frac{0.5646 - 0.4794}{0.1} = 0.852 \end{aligned}$$

Exemplo 2: Aproximação da Derivada Segunda

Consideremos a mesma função $f(x) = \sin(x)$ e queremos calcular uma aproximação para $f''(0.5)$, com $h = 0.1$. Utilizando a fórmula de diferença central para a derivada segunda, obtemos:

$$
\begin{aligned}
f''(0.5) &\approx \frac{f(0.5+0.1) - 2f(0.5) + f(0.5-0.1)}{(0.1)^2} \\
&= \frac{\sin(0.6) - 2\sin(0.5) + \sin(0.4)}{0.01} \\
&= \frac{0.5646 - 2 \cdot 0.4794 + 0.3894}{0.01} \\
&\approx \frac{0.5646 - 0.9588 + 0.3894}{0.01} = -0.048
\end{aligned}
$$

6.8 CONSIDERAÇÕES FINAIS

A derivação numérica possui técnicas poderosas para a resolução de problemas onde as abordagens analíticas não são viáveis. No entanto, o sucesso dessas técnicas depende de uma boa escolha do método e de parâmetros, como o passo h, como também, de uma compreensão dos erros associados a cada método para a obtenção de resultados confiáveis.

7 Integração Numérica

A integração numérica, assim como a derivação numérica, é ferramentas fundamentais no campo dos métodos numéricos. Elas permitem a aproximação de integrais de funções que não podem ser resolvidas analiticamente, ou cuja a solução analítica é demasiamente trabalhosa. Neste capítulo, abordaremos os principais métodos de integração numérica, com ênfase em suas aplicações e limitações.

7.1 CONCEITUAÇÃO INTEGRAÇÃO NUMÉRICA

A integração numérica é uma ferramenta fundamental para calcular integrais analíticas, conforme revisado na Secção 2.4, definidas de funções, especialmente quando a solução analítica não está disponível ou é muito complexa.

- **Métodos de integração numérica**: São técnicas que fornecem aproximações para os valores de integrais definidas. Esses métodos são amplamente utilizados em diversas áreas da engenharia e ciências aplicadas, principalmente quando a solução exata da integral não pode ser obtida de maneira analítica.

- **Quando a integração numérica é útil**:

  - Quando a função $f(x)$ não é conhecida de forma explícita, mas são fornecidos apenas valores discretos de $f(x)$, como em uma tabela de dados experimentais.

  - Quando $f(x)$ é conhecida, mas sua forma é muito complicada, tornando impraticável a obtenção de uma primitiva ou solução analítica.

- **Ideia básica dos métodos de integração**: O conceito central da integração numérica é aproximar a função $f(x)$ por um polinômio, que se ajusta de forma satisfatória dentro de um intervalo $[a,b]$. A partir dessa aproximação, o cálculo da integral é reduzido à integração de um polinômio, tarefa que pode ser resolvida de forma relativamente simples e eficiente.

Existem diversos métodos para o cálculo numérico de integrais. Esses métodos deduzem fórmulas para obter o valor aproximado da integral utilizando pontos discretos do integrando. Em geral, eles são classificados em duas categorias principais:

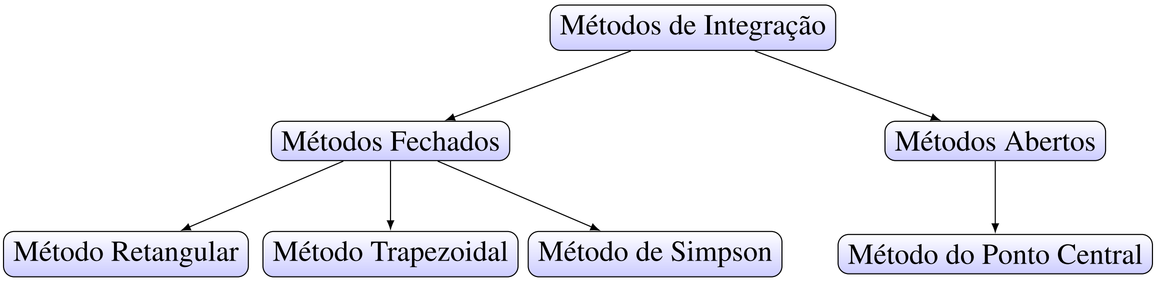
- **Métodos fechados**: Utilizam valores da função em pontos que incluem as extremidades do intervalo de integração.

- **Métodos abertos**: Consideram pontos que não incluem as extremidades, sendo úteis quando os valores da função não estão disponíveis nas bordas do intervalo.

Esses métodos são amplamente empregados para resolver problemas práticos em que as funções envolvidas não podem ser integradas de forma exata, sendo a integração numérica uma solução eficiente para obter resultados aproximados com boa precisão. Na Figura 6 é apresentada uma percepção geral do métodos de integração.

Graficamente, essa percepção é exibida nas figuras 7a e 7b.

Figura 6 – Métodos de interação

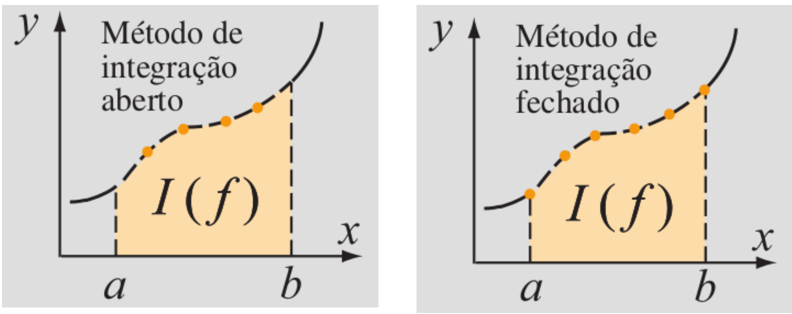


Fonte: Autor,(2024)

Figura 7 – Comparação dos métodos de integração

(a) Métodos Abertos

(b) Métodos Fechados



Fonte: GILAT, (2008)

7.2 MÉTODO DO RETÂNGULO

O Método do Retângulo aproxima a integral pela soma de áreas de retângulos sob a curva.

7.2.1 MÉTODO DO RETÂNGULO SIMPLES

A equação básica para a aproximação da integral de uma função $f(x)$ no intervalo $[a,b]$, pelo Método do Retângulo Simples (MRS), é dada pela Equação 29:

$$I(f) = \int_a^b f(x)dx = f(a)(b-a) \approx f(b)(b-a) \tag{29}$$

Essa aproximação do MRS é, contudo, suscetível de erros mais grosseiros, conforme por ser percebido na Figura 8.

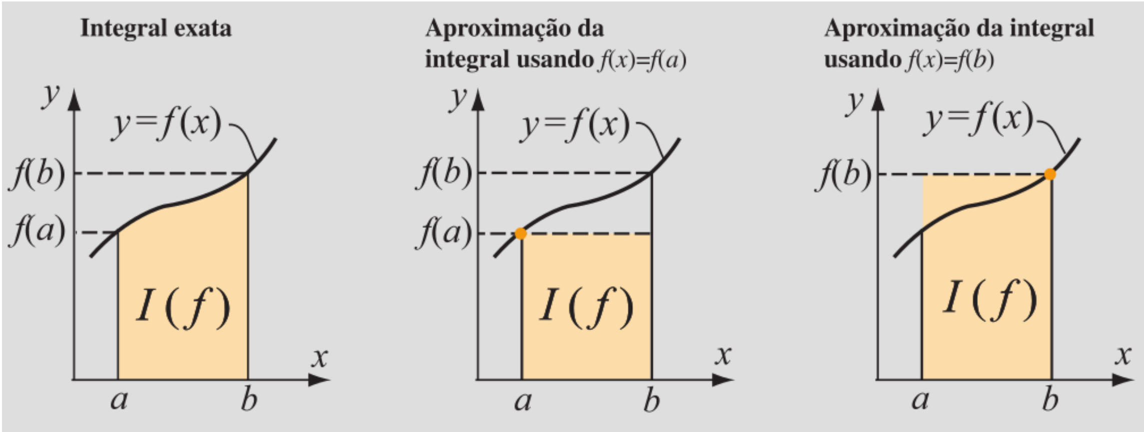
$$\text{Isso implica que para :} \begin{cases} f(x) = f(a), I_{aprox}(f) < I_{exata}(f), tendendo\ a\ I_{aprox}(f) << I_{exata}(f) \\ f(x) = f(b), I_{aprox}(f) > I_{exata}(f), tendendo\ a\ I_{aprox}(f) >> I_{exata}(f) \end{cases}$$

7.2.2 MÉTODO DO RETÂNGULO COMPOSTO

Visando reduzir o erro apresentado no MRS, no Método do Retângulo Composto (MRC) criam-se N subintervalos sobre os quais aplica-se o mesmo método, seguindo-se a soma do resultado do integral de cada intervalo, conforme apresentado na Figura 9 e definido na Equação 30.

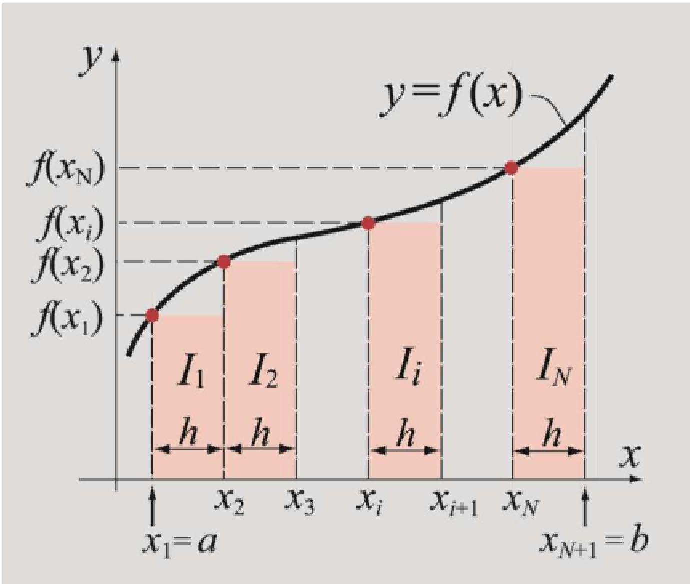
$$I(f) = \int_a^b f(x)dx \approx h \sum_{i=1}^{N} f(x_i) \tag{30}$$

Figura 8 – Método do Retângulo Simples



Fonte: GILAT,(2008)

Figura 9 – Método do Retângulo Composto



Fonte: GILAT,(2008)

Note que a Equação 30 é aplicável quando os subintervalos têm a mesma largura h. Doutro modo, os vários intervalos permanecem desagrupados e o somatório é realizado individualmente intervalo-a-intervalo.

7.3 MÉTODO DO PONTO CENTRAL

O Método do Ponto Central aproxima a integral pela soma de áreas de retângulos sob a curva, utilizando o ponto médio do intervalo da área sob análise, em lugar dos pontos de extremos.

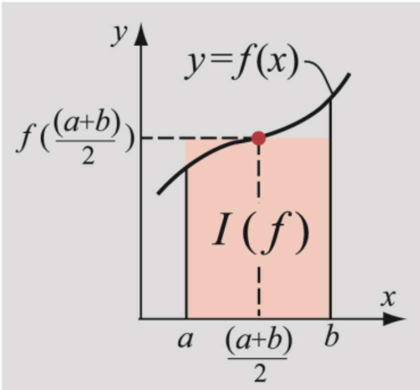
7.3.1 MÉTODO DO PONTO CENTRAL SIMPLES

A equação básica para a aproximação da integral pelo Método do Ponto Central Simples (MPCS) de uma função $f(x)$ no intervalo $[a,b]$ é dada pela Equação 31:

$$I(f) = \int_a^b f(x)dx = f(\frac{a+b}{2})(b-a) \quad (31)$$

Essa aproximação é, contudo, suscetível de erros mais grosseiros, conforme por ser percebido na Figura 10.

Figura 10 – Método do Ponto Central Simples

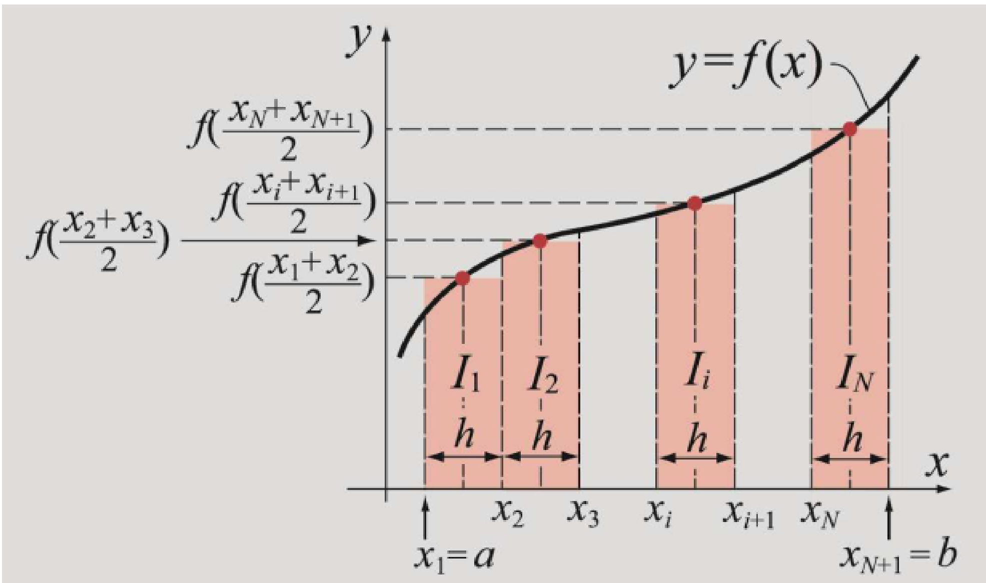


Fonte: GILAT,(2008)

7.3.2 MÉTODO DO PONTO CENTRAL COMPOSTO

Visando reduzir o erro apresentado no MPCS, no Método do Ponto Central Composto (MPCC) criam-se N subintervalos sobre os quais aplica-se o mesmo método, seguindo-se a soma do resultado do integral de cada intervalo, conforme apresentado na Figura 11 e definido na Equação 32.

Figura 11 – Método do Ponto Central Composto



Fonte: GILAT,(2008)

$$I(f) = \int_a^b f(x)dx \approx h\sum_{i=1}^{N} f(x_i) \quad (32)$$

Note que a Equação 32 é aplicável quando os subintervalos têm a mesma largura h. Doutro modo, os vários intervalos permanecem desagrupados e o somatório é realizado individualmente intervalo-a-intervalo.

7.4 MÉTODO DO TRAPÉZIO

O Método do Trapézio aproxima a integral pela soma de áreas de trapézios sob a curva.

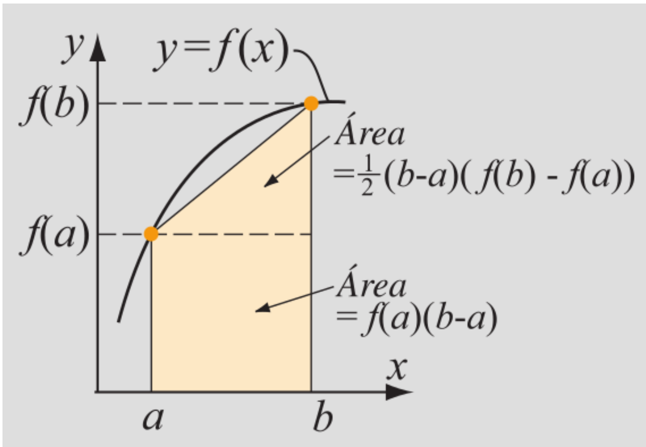
7.4.1 MÉTODO DO TRAPÉZIO SIMPLES

A equação básica para a aproximação da integral de uma função $f(x)$ no intervalo $[a,b]$, pelo Método do Trapézio Simples (MTS) Simples, é dada pela Equação 33:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}[f(a)+f(b)] \tag{33}$$

Essa aproximação é, contudo, suscetível de erros mais grosseiros, conforme por ser percebido na Figura 12.

Figura 12 – Método do Trapézio Simples



Fonte: GILAT,(2008)

7.4.2 MÉTODO DO TRAPÉZIO COMPOSTO

Visando reduzir o erro apresentado no MTS, no Método do Trapézio Simples (MTC) criam-se N subintervalos sobre os quais aplica-se o mesmo método, seguindo-se a soma do resultado do integral de cada intervalo, conforme apresentado na Figura 13 e definido na Equação 34.

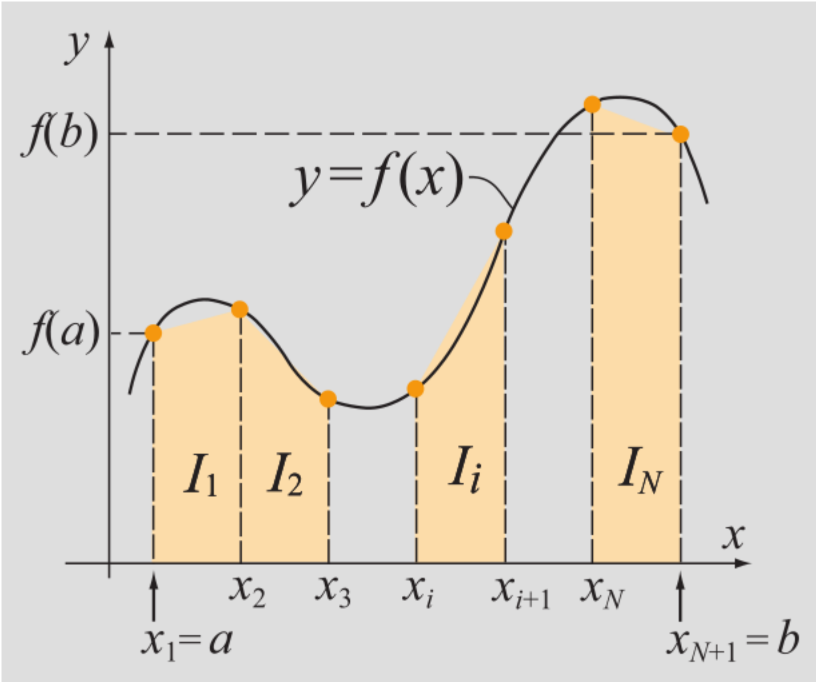
Note que a aplicação do método trapezoidal em cada subintervalo $[x_i, x_{i+1}]$ resulta em:

$$I_i(f) = \int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{[f(x_i)+f(x_{i+1})]}{2}(x_{i+1}-x_i)$$

Substituindo-se a aproximação trapezoidal, tem-se:

$$I(f) = \int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{N}[f(x_i)+f(x_{i+1})](x_{i+1}-x_i) \tag{34}$$

Figura 13 – Método do Trapézio Composto



Fonte: GILAT,(2008)

A Equação 34 é a equação geral do MTC. Neste caso, os subintervalos $[xi, xi+1]$ não precisam ser igualmente espaçados, podendo, portanto, cada um dos subintervalos pode ter uma largura diferente.

Contudo, se todos os subintervalos tiverem o mesmo tamanho, a referida expressão pode ser reduzida a uma fórmula útil para a programação, na qual a soma é expandida:

Para efeitos de linguagem de programação essa Equação pode ser representada como:

$$I(f) \approx \frac{h}{2}[f(a)+f(b)]+h\sum_{i=1}^{N} f(x_i),$$

caso h tenha larguras idênticas.

Note que a Equação 34 é aplicável quando os subintervalos têm a mesma largura h. Doutro modo, os vários intervalos permanecem desagrupados e o somatório é realizado individualmente intervalo-a-intervalo.

7.5 REGRA DE SIMPSON

A Regra de Simpson é uma técnica de aproximação numérica usada para calcular integrais definidas. Ela é especialmente útil quando a função $f(x)$ não pode ser integrada de forma analítica. Existem duas variações principais da Regra de Simpson: a **Regra de Simpson 1/3** e a **Regra de Simpson 3/8**. Ambas se baseiam na aproximação da integral através de polinômios interpoladores de grau 2 ou 3.

7.5.1 REGRA DE SIMPSON 1/3

A Regra de Simpson 1/3 se baseia na aproximação da função $f(x)$ por um polinômio de segundo grau (parábola) dentro de um intervalo $[a,b]$. Esse método divide o intervalo de integração em dois subintervalos de igual comprimento e faz uso de três pontos igualmente espaçados, a, b, e o ponto médio $(a+b)/2$.

A Equação 35 define a Regra de Simpson 1/3::

$$I(f) = \int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{35}$$

Em que:

- a e b são os limites de integração;
- $f(a)$, $f(b)$ e $f\left(\frac{a+b}{2}\right)$ são os valores da função nos pontos a, b e no ponto médio.

Essa equação utiliza pesos 1, 4 e 1, sendo que a função avaliada no ponto médio recebe um peso maior, o que melhora a aproximação. A Regra de Simpson 1/3 é exata para polinômios de até segundo grau.

7.5.2 REGRA DE SIMPSON 3/8

A Regra de Simpson 3/8 se baseia na aproximação da função $f(x)$ por um polinômio cúbico (terceiro grau) dentro de um intervalo $[a,b]$. Diferente da Regra de Simpson 1/3, aqui o intervalo de integração é dividido em três subintervalos iguais e utiliza-se quatro pontos igualmente espaçados: a, $a+h$, $a+2h$ e b, onde $h = \frac{b-a}{3}$.

A Equação 36 define a Regra de Simpson 3/8 :

$$I(f) = \int_a^b f(x)dx \approx \frac{3(b-a)}{8}\left[f(a) + 3f(a+h) + 3f(a+2h) + f(b)\right] \tag{36}$$

Em que:

- a e b são os limites de integração;
- $h = \frac{b-a}{3}$ é a distância entre os pontos;
- $f(a)$, $f(a+h)$, $f(a+2h)$ e $f(b)$ são os valores da função nesses quatro pontos.

Diferente da Regra de Simpson 1/3, essa equação utiliza pesos 1, 3, 3 e 1. Assim como a Regra de Simpson 1/3, a Regra de Simpson 3/8 é exata para polinômios de grau até **três**, mas pode ser mais precisa em alguns casos, especialmente para funções que exibem maior curvatura.

7.6 SCRITPS PYTHON PARA MÉTODOS INTEGRAÇÃO

7.6.1 INTEGRAÇÃO NUMÉRICA SIMPLES

O *script* em Python a seguir mostre como resultado um comparativo entre os métodos simples de integração numérica.

```
"""
Integracao via Metodos de retangulo,trapezio e ponto central simples
"""
f=lambda x: 97000*x/(5*x**2 + 570000)
a,b=40,93
```

```
I=[]
s=f(a)*(b-a) # Retangulo simples extremo a
I.append(s); s=0
s=f(b)*(b-a) # Retangulo simples extremo b
I.append(s); s=0
s=f((a+b)/2)*(b-a) # Ponto central
I.append(s); s=0
s=((f(a)+f(b))/2)*(b-a) # Trapezio Simples
I.append(s); s=0

M=["Retangulo simples altura 'a'","Retangulo simples altura 'b'",
'Ponto central','Trapezio Simples']
print()
for i in range(len(I)):
print(round(I[i],4),' >> Metodo:',M[i])
# Integral Analitica via biblioteca scipy
import scipy.integrate as integrate
g,e=integrate.quad(f,a,b)
print(f'\nSolucao analitica {round(g,4)}, via comando quad - scipy')
```

7.6.2 INTEGRAÇÃO NUMÉRICA COMPOSTA

O *script* em Python a seguir mostre como resultado um comparativo entre os métodos compostos de integração numérica.

```
import numpy as np
import scipy.integrate as integrate

def metod_retan(a,b,f,Ns:list=[10,100,1000]):
    ''' Metodos de trapezio composto com variacao da largura dos intervalos '''
    print("\nMetodo de retangulo composto:")
    for j in Ns:
    N=j #Total de intervalos
    h=(b-a)/N # Largura de cada intervalo
    x=np.arange(a,(b+h),h) # Intervalo particionado a:h:b
    s=0
    for i in range(N): # Retangulo composto
    s+=f(x[i])*h

    print(f'Solucao para {N} partes eh {round(s,4)}')

def metod_p_central(a,b,f,Ns:list=[10,100,1000]):
    ''' Metodos do ponto central composto com variacao da largura dos intervalos
       '''
    print("\nMetodo do Ponto Central Composto")
    for j in Ns:
    N=j #Total de intervalos
    h=(b-a)/N # Largura de cada intervalo
    x=np.arange(a,(b+h),h) # Intervalo particionado a:h:b
    s=0
```

```
    for i in range(N): # Ponto central composto
    s+=f((x[i+1] + x[i])/2)*h
    print(f'Solucao para {N} partes eh {round(s,4)}')

def metod_trapz(a,b,f,Ns:list=[10,100,1000]):
    ''' Metodos de trapezio composto com variacao da largura dos intervalos '''
    print("\nMetodo de trapezio composto:")
    for j in Ns:
    N=j #Total de intervalos
    h=(b-a)/N # Largura de cada intervalo
    x=np.arange(a,(b+h),h) # Intervalo particionado a:h:b
    s=0
    for i in range(N): # Trapezio composto
    s+=0.5*( f(x[i]) + f(x[i+1]))*h
    print(f'Solucao para {N} partes eh {round(s,4)}')

if __name__ == "__main__":

    f=lambda x: 97000*x/(5*x**2 + 570000)
    a,b = 40,93
    Ns=[10,100,1000,10000,100000]

    metod_retan(a,b,f,Ns)
    metod_p_central(a,b,f,Ns)
    metod_trapz(a,b,f,Ns)

    g,e=integrate.quad(f,a,b)
    print(f'\nSolucao analitica {round(g,4)}, via comando quad - scipy')
```

7.7 CONSIDERAÇÕES FINAIS

A integração numérica possui técnicas poderosas para a resolução de problemas onde as abordagens analíticas não são viáveis. No entanto, o sucesso dessas técnicas depende de uma boa escolha do método e de parâmetros, como o número de subintervalos para a integração, como também, de uma compreensão dos erros associados a cada método para a obtenção de resultados confiáveis.

8 Métodos de Interpolação

Os métodos de interpolação são fundamentais no campo dos métodos numéricos, permitindo a construção de funções aproximadas a partir de um conjunto discreto de pontos. Esses métodos são amplamente utilizados em engenharia para prever dados ou relacionar grandezas medidas experimentalmente.

8.1 CONCEITOS SOBRE INTERPOLAÇÃO

A interpolação consiste em encontrar uma função que passe exatamente por um conjunto de pontos. Dado um conjunto de pontos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, podemos utilizar diferentes métodos para encontrar uma função $f(x)$ tal que $f(x_i) = y_i$ para todo i.

8.2 INTERPOLAÇÃO POLINOMIAL DE LAGRANGE

O método de interpolação polinomial busca encontrar um polinômio $P(x)$ de grau n que passe por todos os pontos dados. A forma mais comum é a interpolação de Lagrange, que define o polinômio interpolador é apresentada na Equação 37:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{37}$$

em que $L_i(x)$ são os polinômios de Lagrange definidos por:

$$L_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}$$

Exemplo

Dado o conjunto de pontos $(0,1)$, $(1,2)$ e $(2,0)$, encontre o valor de y para $x = 1.5$

1) Calculando o polinômio de Lagrange, conforme Equação 37.

$$P(x) = 1 \cdot \frac{(x-1)(x-2)}{(0-1)(0-2)} + 2 \cdot \frac{(x-0)(x-2)}{(1-0)(1-2)} + 0 \cdot \frac{(x-0)(x-1)}{(2-0)(2-1)}$$

Simplificando:

$$P(x) = \frac{(x-1)(x-2)}{2} - 2 \cdot x(x-2)$$

2) Substituindo $x = 1.5$ no polinômio encontrado $P(x)$:

$$P(1.5) = \frac{(1.5-1)(1.5-2)}{2} - 2 \cdot 1.5(1.5-2)$$

Calculando os termos:

$$P(1.5) = \frac{(0.5)(-0.5)}{2} - 2 \cdot 1.5(-0.5)$$

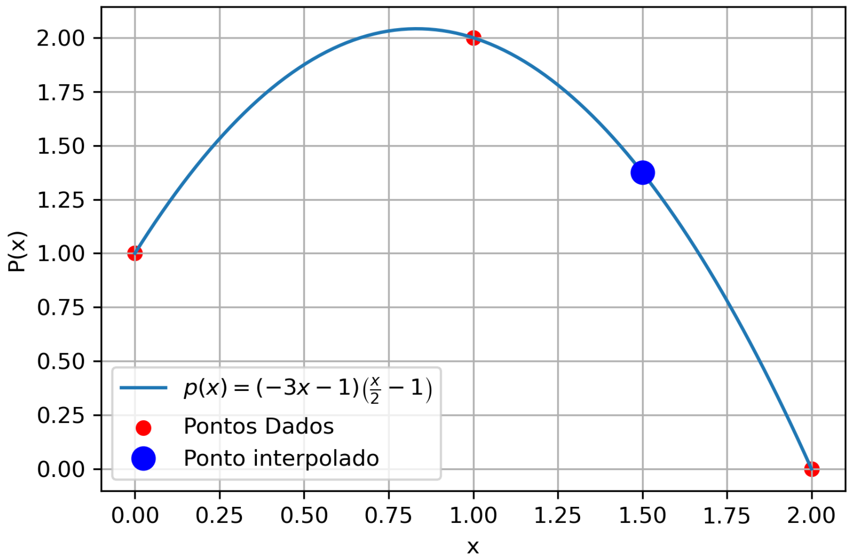
$$P(1.5) = \frac{-0.25}{2} - 2 \cdot (-0.75)$$

$$P(1.5) = -0.125 + 1.5$$

$$P(1.5) = 1.375$$

Assim, o valor interpolado para $x = 1.5$ usando o método de Lagrange é $P(1.5) = 1.375$. Na Figura 14 é apresentada a percepção gráfica desta interpolação.

Figura 14 – Interpolação de Lagrange



Fonte: Autor,(2024)

Observe na Figura 14 o polinônio $p(x)$. Note que o *script* computacional produziu, ocasionalmente, um $p(x)$ equivalente àquele imediantamente expresso a partir da Equação 37. Isso ocorre por efeito de simplificação, bem como das aproximações e ajustes da linguagem de programação utilizada, contudo, sem divergência nos resultados.

8.2.1 SCRITP PYTHON PARA INTERPOLACAO DE LAGRANGE

```
import numpy as np
def PolIntLagr(x,y,p):
      ''' Exibe o resultado da interpolacao de lagrange um valor 'p' num pares de
         vetores X,Y.'''
      k=np.ones(len(x))
      for i in range(len(x)):
      for j in range(len(x)):
      if (i!=j):
      k[i]=k[i]*(p-x[j])/(x[i]-x[j])
      Yint=sum(y*k)
      return Yint

if __name__=="__main__":
      # Ponto de interpolacao e vetores de entrada x,y
      p=1.5;
      x=np.array([0,1,2])
      y=np.array([1,2,0])

      # Resultado da interpolacao do ponto 'p'
      Yint= PolIntLagr(x,y,p)
```

```
print('\nA aproximacao encontrada para f(%.2f) = %.3f'%(p,Yint))
```

8.3 INTERPOLAÇÃO POR DIFERENÇAS DIVIDIDAS DE NEWTON

Outra abordagem é a interpolação por diferenças divididas de Newton, na qual o polinômio interpolador é descrito pela Equação 38::

$$P(x) = f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) + \dots \tag{38}$$

em que $f[x_0,x_1,\dots,x_k]$ são as diferenças divididas calculadas a partir dos pontos.

Exemplo

Utilizando os mesmos pontos $(0,1)$, $(1,2)$ e $(2,0)$ e o mesmo valor a ser interpolado $x = 1.5$, temos os seguintes valores de diferenças divididas:

1) Calculando o polinômio de Newton, conforme Equação 38.

$$f[x_0,x_1] = \frac{f(1)-f(0)}{1-0} = 1$$

$$f[x_0,x_1,x_2] = \frac{\frac{f(2)-f(1)}{2-1} - f[x_0,x_1]}{2-0} = \frac{\frac{0-2}{2-1} - 1}{2-0} = \frac{-2-1}{2} = -1.5$$

Portanto, o polinômio interpolador de Newton é:

$$P(x) = 1 + 1(x-0) - 1.5(x-0)(x-1)$$

Simplificando:

$$P(x) = 1 + x - 1.5x(x-1)$$

2) Substituindo $x = 1.5$ no polinômio encontrado $P(x)$:

$$P(1.5) = 1 + 1.5 - 1.5(1.5)(1.5-1)$$

$$P(1.5) = 1 + 1.5 - 1.5(0.75)$$

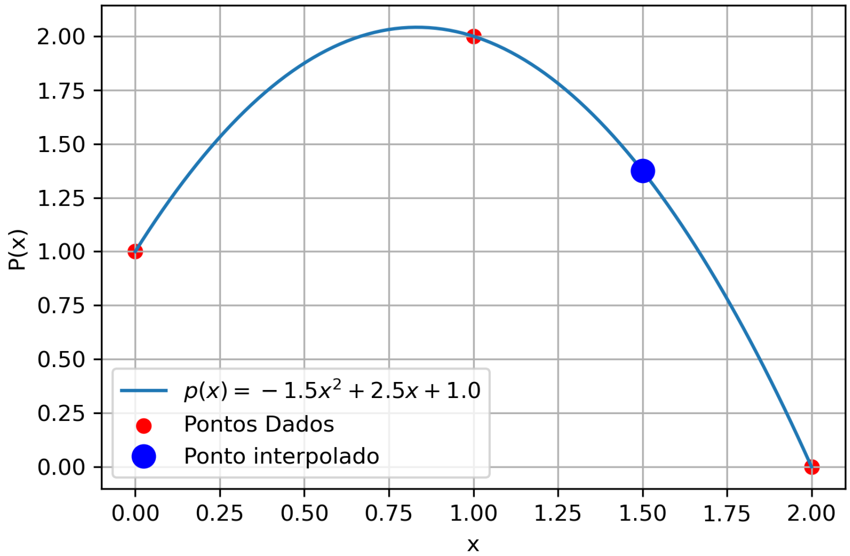
$$P(1.5) = 2.5 - 1.125$$

$$P(1.5) = 1.375$$

Na Figura 15 é apresentada a percepção gráfica desta interpolação.

Observe na Figura 15 o polinônio $p(x)$. Note que o *script* computacional produziu, ocasionalmente, um $p(x)$ equivalente àquele imediantamente expresso a partir da Equação 38. Isso ocorre por efeito de simplificação, bem como das aproximações e ajustes da linguagem de programação utilizada, contudo, sem divergência nos resultados.

Figura 15 – Interpolação de Newton



Fonte: Autor,(2024)

8.3.1 SCRITP PYTHON PARA INTERPOLACAO DE NEWTON

```
import numpy as np
def PolInterNewton(x,y,p):
      ''' Exibe o resultado objetivo da interpolacao de Newton para um valor 'p' num
           pares de vetores X,Y.
      '''
      a,s= [y[0]],y # s eh vetor das divisoes
      for i in range(len(x)-1):
      y=s # Atualiza o vetor y
      s=(y[1:]-y[:-1])/(x[(1+i):]-x[:-(1+i)]) # Reduz o vetor x
      a.append(s[0])
      xn,Yint=1,a[0]
      for k in range(1,len(x)):
      xn=xn*(p - x[k-1])
      Yint=Yint+a[k]*xn

      return Yint

if __name__=="__main__":
      # Ponto de interpolacao e vetores de entrada x,y
      p=1.5;
      x=np.array([0,1,2])
      y=np.array([1,2,0])

      # Resultado da interpolacao do ponto 'p'
      Yint= PolInterNewton(x,y,p)
      print('\nA aproximacao encontrada para f(%.2f) = %.3f'%(p,Yint))
```

8.4 CONCLUSÃO E COMPARAÇÃO DOS MÉTODOS

No método de Lagrange, o polinômio é construído a partir de uma combinação de polinômios base, em que cada um deles é responsável por garantir que o polinômio completo passe pelos pontos conhecidos. Esse método tende a ser mais direto para uma quantidade fixa de pontos, porém pode ser mais suscetível a erros numéricos quando o número de pontos aumenta, especialmente devido ao comportamento das funções base.

Por outro lado, o método de Newton constrói o polinômio de forma incremental, utilizando diferenças divididas entre os pontos de dados. Esse processo permite adicionar novos pontos ao polinômio sem a necessidade de recalcular toda a expressão, o que pode ser vantajoso em certos casos, como em aplicações em que se deseja ajustar novos dados progressivamente. No entanto, o cálculo das diferenças divididas pode levar a resultados diferentes, especialmente quando há mais sensibilidade a arredondamentos numéricos.

Em resumo, ambos os métodos geram polinômios que interpolam os mesmos pontos, mas a escolha de qual utilizar pode depender da aplicação. O método de Lagrange é simples e eficiente para conjuntos de pontos pequenos e fixos, enquanto o método de Newton oferece flexibilidade e eficiência quando há necessidade de ajustes dinâmicos ou de manipulação incremental dos dados.

REFERÊNCIAS BIBLIOGRÁFICAS

CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia-7ª Edição**. [S.l.]: McGraw Hill Brasil, 2016.

COSTA, R. F. **8 alternativas open sources para o MatLab**. [s.n.], 2017. [Online; accessed 20-Dezembro-2019]. Disponível em: <https://www.linuxdescomplicado.com.br/2017/03/8-alternativas-open-sources-para-o-matlab.html>.

GILAT, A.; SUBRAMANIAM, V. **Métodos numéricos para engenheiros e cientistas: uma introdução com aplicações usando o MATLAB**. Porto Alegre - RS: Bookman, 2008. ISBN 978-85-7780-297-5.

HUNT, J. **A beginners guide to Python 3 programming**. [S.l.]: Springer, 2019.

MATHEWS, J. H. et al. **Métodos numéricos con Matlab**. [S.l.]: Prentice Hall, 2000. v. 2.

PINTO, J. C. **Métodos numéricos em problemas de engenharia química**. [S.l.]: E-papers, 2001.

SILVA, W. M. da et al. **Uma Alternativa Robusta, Rápida e Gratuita para Pesquisadores que Utilizam MATLAB**. s.l: [s.n.], 2014. Disponível em: <https://www.researchgate.net/profile/Flavio-Rossini/publication/324089118_Uma_Alternativa_Robusta_Rapida_e_Gratuita_para_Pesquisadores_que_Utilizam_MatLabr/links/5abd06860f7e9bfc0457a66d/Uma-Alternativa-Robusta-Rapida-e-Gratuita-para-Pesquisadores-que-Utilizam-MatLabr.pdf>. Acesso em: Out. 2023.