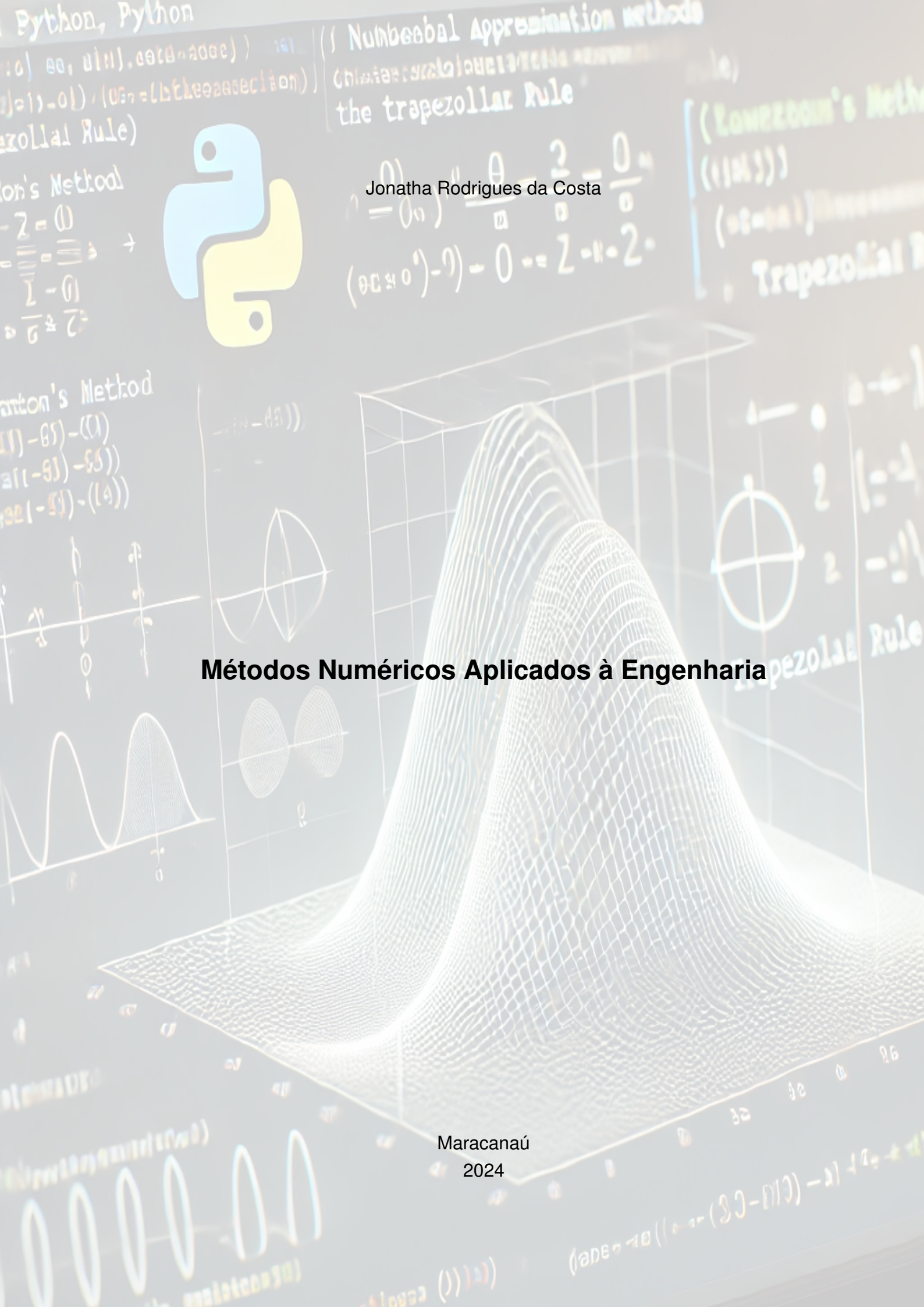


Jonatha Rodrigues da Costa

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.

1.1.1 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/

1.1.2 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®

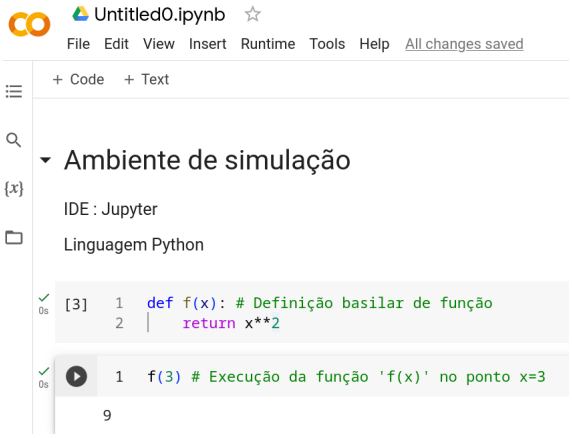
Fonte: Autor, 2024

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).

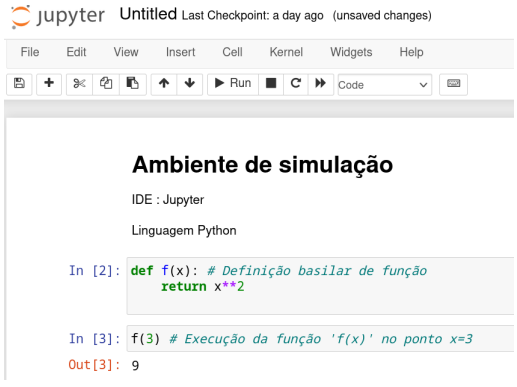
Essa linguagem apresenta-se incorporada a um *(Integrated Development Environment* (ambiente de desenvolvimento integrado) (IDE) de empresas de serviços de *web* como a Google®, pelo recurso *google colaboratory*, Figura 3a. Nesse mesmo contexto o Jupyter Notebook®, Figura 3b, apresenta-se também como 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.

1.1.3 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 Fundamentos Matemáticos

Neste capítulo, revisaremos os principais conceitos matemáticos que servem como base para os estudos de MN. Estes incluem álgebra linear, 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 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.

2.1.1 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.

2.1.1.1 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:

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

2.1.1.2 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.

2.1.1.3 Script Python de operações básicas com limites

```
# 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.

2.1.2 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.

2.1.2.1 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.

2.1.2.2 Script Python de operações básicas com derivadas

```
import sympy as sp
# Exemplo: Linearidade
x = sp.symbols('x')
f = 2*x**2 + 5*(3*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)
```

2.1.3 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.

2.1.3.1 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.

2.1.3.2 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)
```

2.2 ÁLGEBRA LINEAR

A álgebra linear é uma das áreas centrais da matemática aplicada. A seguir, revisaremos conceitos importantes como vetores, matrizes, sistemas lineares e suas propriedades.

2.2.1 VETORES

Um vetor $\mathbf{v}$ em um espaço n-dimensional é uma lista ordenada de números:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \tag{13}$$

Vetores podem ser somados entre si e multiplicados por escalares. Se $\mathbf{u}$ e $\mathbf{v}$ são vetores em $\mathbb{R}^n$, temos as seguintes propriedades:

a) **Comutatividade**: $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$

Exemplo: Se $\mathbf{u} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ e $\mathbf{v} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$, temos:

$$\mathbf{u} + \mathbf{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

ou

$$\mathbf{v} + \mathbf{u} = \begin{pmatrix} 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

Logo, $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.

b) **Associatividade**: $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$

Exemplo: Se $\mathbf{u} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\mathbf{v} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ e $\mathbf{w} = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$, temos:

$$(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 9 \\ 12 \end{pmatrix}$$

ou

$$\mathbf{u} + (\mathbf{v} + \mathbf{w}) = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \left(\begin{pmatrix} 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 5 \\ 6 \end{pmatrix} \right) = \begin{pmatrix} 9 \\ 12 \end{pmatrix}$$

Portanto, $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$.

c) **Elemento neutro**: $\exists$ um vetor nulo $\mathbf{0}$, tal que $\mathbf{v} + \mathbf{0} = \mathbf{v}$

Exemplo: Para $\mathbf{v} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$, temos:

$$\mathbf{v} + \mathbf{0} = \begin{pmatrix} 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

Logo, o vetor nulo $\mathbf{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ é o elemento neutro.

d) **Inverso aditivo**: Para todo vetor $\mathbf{v}$, $\exists$ um vetor $-\mathbf{v}$, tal que $\mathbf{v}+(-\mathbf{v})=\mathbf{0}$

Exemplo: Para $\mathbf{v}=\begin{pmatrix}3\\4\end{pmatrix}$, o inverso aditivo é $-\mathbf{v}=\begin{pmatrix}-3\\-4\end{pmatrix}$, então:

$$\mathbf{v}+(-\mathbf{v})=\begin{pmatrix}3\\4\end{pmatrix}+\begin{pmatrix}-3\\-4\end{pmatrix}=\begin{pmatrix}0\\0\end{pmatrix}$$

Portanto, $\mathbf{v}+(-\mathbf{v})=\mathbf{0}$.

2.2.1.1 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

# Definir vetores u, v, w e o vetor nulo
u = np.array([1, 2])
v = np.array([3, 4])
w = np.array([5, 6])
zero = np.array([0, 0])

# 1. Comutatividade: u + v = v + u
comut_esq = u + v
comut_dir = v + u
print(f"Comutatividade: u + v = {comut_esq}, v + u = {comut_dir}")

# 2. Associatividade: (u + v) + w = u + (v + w)
assoc_esq = (u + v) + w
assoc_dir = u + (v + w)
print(f"Associatividade: (u + v) + w = {assoc_esq}, u + (v + w) =
    {assoc_dir}")

# 3. Elemento neutro: v + zero = v
elemento_neutro = v + zero
print(f"Elemento neutro: v + zero = {elemento_neutro}")

# 4. Inverso aditivo: v + (-v) = 0
inverso_aditivo = v + (-v)
print(f"Inverso aditivo: v + (-v) = {inverso_aditivo}")
```

Saída esperada do código

```
Comutatividade: u + v = [4 6], v + u = [4 6]
Associatividade: (u + v) + w = [ 9 12], u + (v + w) = [ 9 12]
Elemento neutro: v + zero = [3 4]
Inverso aditivo: v + (-v) = [0 0]
```

2.2.2 MATRIZES

Matrizes são essenciais para representar sistemas de equações lineares e muitas outras aplicações em engenharia. Uma matriz $A \in \mathbb{R}^{m \times n}$ tem m linhas e n colunas. Algumas operações importantes são:

- **Transposta**: A transposta de uma matriz A, denotada por $A^\top$, é obtida trocando-se as linhas pelas colunas.
- **Multiplicação por escalar**: Dada uma constante $c \in \mathbb{R}$ e uma matriz A, a matriz cA é obtida multiplicando cada elemento de A por c.
- **Multiplicação de matrizes**: Para $A \in \mathbb{R}^{m \times n}$ e $B \in \mathbb{R}^{n \times p}$, o produto $AB \in \mathbb{R}^{m \times p}$ é dado por:

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{14}$$

2.2.2.1 Propriedades das Matrizes

- **Associatividade**: $A(BC) = (AB)C$
- **Distributividade**: $A(B+C) = AB + AC$
- **Elemento neutro**: $\exists$ a matriz identidade $I \in \mathbb{R}^{n \times n}$ tal que $AI = IA = A$

2.2.2.2 Script Python de operações com matrizes

```
import numpy as np

# Definir matrizes A, B, C
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.array([[9, 10], [11, 12]])

# 1. Transposta de A
A_transposta = A.T
print(f''Transposta de A:\n{A_transposta}'')

# 2. Produto por escalar: cA, em que c = 2
c = 2; A_escalar = c * A
print(f"Produto de A por escalar (c=2):\n{A_escalar}")

# 3. Produto de matrizes: A * B
AB = np.dot(A, B)
print(f"Produto de A e B:\n{AB}")

# 4. Associatividade: A(BC) = (AB)C
BC = np.dot(B, C)
associativa_esq = np.dot(A, BC)
AB_C = np.dot(AB, C)
print(f"Associatividade: A(BC) =\n{associativa_esq}\n(AB)C = \n {
    AB_C}")
```

```
                # 5. Distributividade: A(B + C) = AB + AC
                B_somado_C = B + C
                A_BC = np.dot(A, B_somado_C)
                A_B_somado_A_C = np.dot(A, B) + np.dot(A, C)
                print(f"Distributividade: A(B + C) =\n{A_BC}\nAB + AC =\n{
                    A_B_somado_A_C}")

                # 6. Elemento neutro: AI = IA = A (matriz identidade I)
                I = np.eye(2) # Matriz identidade 2x2
                A_I = np.dot(A, I)
                I_A = np.dot(I, A)
                print(f"Elemento neutro: AI =\n{A_I}\nIA =\n{I_A}")
```

Saída esperada do código

```
Transposta de A:
[[1 3]
[2 4]]
Multiplicação de A por escalar (c=2):
[[2 4]
[6 8]]
Multiplicação de A e B:
[[19 22]
[43 50]]
Associatividade: A(BC) =
[[467 518]
[681 756]]
(AB)C =
[[467 518]
[681 756]]
Distributividade: A(B + C) =
[[100 112]
[228 256]]
AB + AC =
[[100 112]
[228 256]]
Elemento neutro: AI =
[[1. 2.]
[3. 4.]]
IA =
[[1. 2.]
[3. 4.]]
```

2.2.3 DETERMINANTES

O determinante de uma matriz quadrada $A \in \mathbb{R}^{n\times n}$ é uma função escalar que fornece informações importantes sobre a matriz, como se ela é invertível.

2.2.3.1 Propriedades do Determinante

- **Determinante da transposta**: $\det(A^\top) = \det(A)$
  Exemplo: Para a matriz
  $$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$
  temos:
  $$\det(A) = 1\cdot 4 - 2\cdot 3 = -2$$
  E a transposta é
  $$A^\top = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}, \quad \det(A^\top) = 1\cdot 4 - 3\cdot 2 = -2$$
- **Determinante do produto**: $\det(AB) = \det(A)\det(B)$
  Exemplo: Se
  $$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$
  então,
  $$\det(AB) = \det(A)\det(B)$$
- **Invertibilidade**: Uma matriz A é invertível se, e somente se, $\det(A) \neq 0$
  Exemplo: Para a matriz
  $$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$
  temos $\det(A) = -2 \neq 0$, logo A é invertível.

2.2.3.2 Inversão de Matrizes

A inversa de uma matriz $A \in \mathbb{R}^{n\times n}$ é uma matriz A^{-1} tal que:

$$AA^{-1} = A^{-1}A = I \tag{15}$$

Exemplo: Para a matriz

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

a inversa é dada por:

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \frac{1}{-2} \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix}$$

2.2.3.3 Script Python para Cálculo de Determinantes e Inversas

O código a seguir ilustra como calcular determinantes e a inversa de matrizes utilizando Python e a biblioteca `NumPy`:

```
import numpy as np

# Definir matriz A
A = np.array([[1, 2], [3, 4]])

# 1. Calculando o determinante
det_A = np.linalg.det(A)
print(f"Determinante de A: {det_A}")

# 2. Calculando a transposta e seu determinante
A_T = A.T
det_AT = np.linalg.det(A_T)
print(f"Determinante de A^T: {det_AT}")

# 3. Verificanado a propriedade do determinante do produto
B = np.array([[5, 6], [7, 8]])
det_AB = np.linalg.det(np.dot(A, B))
det_A_times_B = det_A * np.linalg.det(B)
print(f"Determinante do produto AB: {det_AB}, igual a det(A) * det(B): {
    det_A_times_B}")

# 4. Calculando a inversa de A
if det_A != 0:
      A_inv = np.linalg.inv(A)
      print(f"Inversa de A:\n{A_inv}")
else:
      print("A matriz nao possui inversa!")
```

Saída esperada do código

```
Determinante de A: -2.0000000000000004
Determinante de A^T: -2.0000000000000004
Determinante do produto AB: -2.000000000000002,
igual a det(A) * det(B): -2.000000000000002
Inversa de A:
[[-2.   1. ]
[ 1.5 -0.5]]
```

2.3 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>.

3 Sistemas Lineares

Um sistema linear é um conjunto de equações lineares que podem ser expressas na forma matricial. Os sistemas lineares são fundamentais na modelagem de problemas em várias disciplinas de engenharia, incluindo controle, eletricidade e mecânica.

3.1 SISTEMA LINEAR

Um **sistema linear** consiste em um conjunto de equações lineares com múltiplas variáveis. Ele pode ser representado na forma matricial:

$$A\mathbf{x} = \mathbf{b}$$

em que:

- A é a matriz dos coeficientes,
- $\mathbf{x}$ é o vetor coluna das incógnitas,
- $\mathbf{b}$ é o vetor coluna dos termos independentes.

Soluções de um Sistema Linear

Um sistema linear é dito **dependente**, quando as equações são múltiplas uma da outra, resultando em infinitas soluções. Doutro modo, sistema linear é dito **independente**, quando há uma única solução.

3.1.1 SISTEMA LINEAR DEPENDENTE

Considere o seguinte sistema de duas equações com duas variáveis x_1 e x_2:

$$\begin{aligned} 2x_1 + 3x_2 &= 5 \\ 4x_1 + 6x_2 &= 10 \end{aligned}$$

Esse sistema pode ser representado na forma matricial como:

$$\begin{pmatrix} 2 & 3 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 10 \end{pmatrix}$$

Neste caso, a matriz dos coeficientes A, o vetor de incógnitas $\mathbf{x}$ e o vetor dos termos independentes $\mathbf{b}$ são:

$$A = \begin{pmatrix} 2 & 3 \\ 4 & 6 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 5 \\ 10 \end{pmatrix}.$$

Solução

Este sistema é **dependente**, pois as equações são múltiplas uma da outra, resultando em infinitas soluções. Uma solução possível é $x_1 = 1$ e $x_2 = 1$. Substituindo esses valores nas equações originais, temos:

$$2(1)+3(1)=5 \quad \text{e} \quad 4(1)+6(1)=10.$$

Portanto, $x_1 = 1$ e $x_2 = 1$ é uma solução válida.

3.1.2 SISTEMA LINEAR INDEPENDENTE

Um **sistema linear independente** é aquele em que as equações não são múltiplas uma da outra, resultando em uma única solução.

Exemplo

Considere o seguinte sistema de duas equações com duas variáveis x_1 e x_2:

$$\begin{aligned} 2x_1 + 3x_2 &= 5 \\ 4x_1 + x_2 &= 6 \end{aligned}$$

Esse sistema pode ser representado na forma matricial como:

$$\begin{pmatrix} 2 & 3 \\ 4 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

Neste caso, a matriz dos coeficientes A, o vetor de incógnitas $\mathbf{x}$ e o vetor dos termos independentes $\mathbf{b}$ são:

$$A = \begin{pmatrix} 2 & 3 \\ 4 & 1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 5 \\ 6 \end{pmatrix}.$$

Solução

Para resolver esse sistema, podemos usar o método de substituição ou eliminação. Usando o método de eliminação:

Multiplicamos a primeira equação por 2 e subtraímos da segunda equação para eliminar x_1:

$$\begin{aligned} (4x_1 + x_2) - 2(2x_1 + 3x_2) &= 6 - 2(5) \\ -5x_2 &= -4 \\ x_2 &= \frac{4}{5} \end{aligned}$$

Agora, substituímos $x_2 = \frac{4}{5}$ na primeira equação:

$$2x_1 + 3\left(\frac{4}{5}\right) = 5 \quad \Rightarrow \quad 2x_1 + \frac{12}{5} = 5$$

Multiplicando ambos os lados por 5 para eliminar a fração:

$$10x_1 + 12 = 25 \quad \Rightarrow \quad 10x_1 = 13 \quad \Rightarrow \quad x_1 = \frac{13}{10}$$

Portanto, a solução única para o sistema é:

$$x_1 = \frac{13}{10}, \quad x_2 = \frac{4}{5}$$

Este sistema é independente, pois tem uma solução única.

3.1.3 SCRIPT PYTHON PARA RESOLVER SISTEMA LINEAR

O código Python abaixo mostra como resolver o sistema linear acima utilizando Python e a biblioteca `NumPy`:

```
import numpy as np

# Definindo a matriz A e o vetor b
A = np.array([[2, 3],
[4, 1]])
b = np.array([5, 6])

# Solucao basilar
x = np.linalg.solve(A, b)

# Exibindo a Solucao
print(f"Solucao do sistema: x1 = {x[0]}, x2 = {x[1]}")
```

Saída esperada do código

```
Solução do sistema: x1 = 1.3, x2 = 0.8
```

Neste exemplo, a solução do sistema é $x_1 = 1.3$ e $x_2 = 0.8$, que corresponde à solução única do sistema independente.

3.2 APLICAÇÕES

Os sistemas lineares são amplamente utilizados em engenharia para:

- Análise de circuitos elétricos.
- Modelagem de sistemas mecânicos.
- Solução de problemas de otimização.

3.3 CONCLUSÃO

Os sistemas lineares são uma parte essencial dos métodos numéricos em engenharia. A compreensão de suas propriedades e métodos de resolução é crucial para a aplicação eficaz em problemas práticos.

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.