2ª EDIÇÃO

CÁLCULO NUMÉRICO APLICADO À ENGENHARIA ELÉTRICA COM PYTHON

Fabrício R. de Santana
Sérgio L. Avila

PUBLICAÇÃO DO IFSC

pecce

Reitor
Zizimo Moreira Filho

Diretor do Campus Florianópolis
Rogério de Souza Versage

Chefe do Departamento Acadêmico de Eletrotécnica
Adriano de Andrade Bresolin

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE SANTA CATARINA – IFSC – CAMPUS FLORIANÓPOLIS
AV. MAURO RAMOS N. 950, FLORIANÓPOLIS, SC.

FABRICIO DE SANTANA RODRIGUES

SÉRGIO LUCIANO ÁVILA

CÁLCULO NUMÉRICO
APLICADO À ENGENHARIA ELÉTRICA
COM PYTHON

2° EDIÇÃO

PUBLICAÇÃO DO IFSC

FLORIANÓPOLIS
2025

Catalogação na fonte pelo
Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina – IFSC

A958c Ávila, Sérgio Luciano
Cálculo numérico aplicado à engenharia elétrica com PYTHON, 2° Edição [recurso eletrônico] / Sérgio Luciano Ávila. - Florianópolis: Publicações do IFSC, 2025.
160 p. : il. color.

Inclui índice remissivo.
ISBN 978-65-83787-08-8

1. Cálculo numérico. 2. Raízes numéricos. 3. Sistemas lineares. 4. Sistemas não-lineares. I. Título

CDD 521.3

Elaborada pela Bibliotecária Raquel Matys Cardenuto Gugelmin – CRB-14/855

GITHUB

HTTPS://GITHUB.COM/PECCE-IFSC/CALCULO_NUMERICO_PYTHON

TODOS OS ALGORITMOS ESTÃO DISPONIBILIZADOS

PYTHON ©

HTTPS://WWW.PYTHON.ORG/

ESTA OBRA É UMA ADAPTAÇÃO DO LIVRO

"CÁLCULO NUMÉRICO APLICADO À ENGENHARIA ELÉTRICA COM PYTHON, 1° EDIÇÃO"

GRUPO DE PESQUISAS EM COMPUTAÇÃO CIENTÍFICA PARA ENGENHARIA

IFSC.EDU.BR/PECCE

Discente:

Fabricio Rodrigues de Santana

fabricio.r11@aluno.ifsc.edu.br

lattes.cnpq.br/3781905503362633

Docente:

Sérgio Luciano Avila

Sergio.Avila@ifsc.edu.br

lattes.cnpq.br/7864845374871073

“Naturalmente deves trabalhar de maneira a não atentar contra a própria consciência.”

Umberto Eco

Sumário

Motivação

Área de concentração:	Engenharia Elétrica
Público:	graduandos e pós-graduandos
Palavras-chave:	cálculo numérico, métodos numéricos, aprendizagem baseada em problemas

A motivação deste livro é ter um material em que a partir de um problema e entendendo a sua natureza, o leitor possa escolher a ferramenta numérica mais apropriada para a sua resolução. A inspiração são as práticas pedagógicas conhecidas por metodologia da problematização ou Aprendizagem Baseada em Problemas ($Problem - Based\ Learning$).

Assim, apresenta-se aqui um livro de cálculo numérico aplicado à engenharia elétrica e as suas principais técnicas numéricas.

Neste contexto, não serão tratadas em profundidade as teorias de cada método, sua origem ou a sua validação matemática. O objetivo é proporcionar ao leitor conhecimento suficiente para entender a natureza do seu problema, e assim o sendo, que ele possa escolher e aplicar um método numérico apto a sua resolução.

O livro é organizado por capítulos, e cada um deles trata de uma natureza de problemas dentro da engenharia. Cada capítulo possui: apresentação de um problema; sua solução analítica ou explicação da inviabilidade em resolvê-la; debate sobre método(s) numérico(s) que o resolva; a resolução do problema pelo método numérico escolhido; justificativa se a solução obtida é adequada; apresentação de pseudocódigo computacional e bibliotecas numéricas sobre o tema; lista de problemas similares; referências bibliográficas básicas e aplicadas.

A estrutura da obra é assim apresentada: faz-se uma discussão preliminar sobre a modelagem numérica, o uso de computadores, suas vantagens e desvantagens; em seguida, um capítulo sobre a linguagem interpretada Python também se faz necessário; após, seguem-se capítulos sobre a busca por raízes, a resolução de sistemas lineares e não lineares, a interpolação de dados, a derivação e integração de funções, a resolução de equações diferenciais ordinárias; e, por fim, a resolução de equações integrais.

Optou-se por não inserir temas, tais como, métodos estocásticos para resolução de sistemas de equações, pesquisa operacional, otimização, aprendizado de máquina e inteligência artificial – tais temas, apesar de relevantes, serão provavelmente abordados em edições futuras.

Reforça-se que o propósito maior deste livro é abrir portas, nunca esgotar assuntos. Para obter maior detalhamento dos métodos, bem como suas validações, sugere-se a busca pelas obras clássicas sobre cálculo numérico ou específicos de cada tema, inclusive sobre Python. Este documento fornece convenções de codificação para o código Python que compreende a biblioteca padrão na distribuição principal do Python.Este livro é inspirado nas seguintes referências:

[1] CEVALLOS-TORRES, L.; BOTTO-TOBAR, M. **Problem-Based Learning: A Didactic Strategy in the Teaching of System Simulation**. Cham: Springer, 2019. ISBN 978-3-030-13392-4 (impresso). ISBN 978-3-030-13393-1 (eBook). DOI: 10.1007/978-3-030-13393-1.

[2] PYTHON SOFTWARE FOUNDATION. **Python: documentação da linguagem** (versão 3.6). Disponível em: <https://docs.python.org/3.6/>. Acesso em: 24 ago. 2025.

[4] REAMAT. **Cálculo Numérico: um livro colaborativo**. Porto Alegre: UFRGS, [s.d.]. Disponível em: <https://www.ufrgs.br/reamat/CalculoNumerico/>. Acesso em: 24 ago. 2025.

[5] ÁVILA, S. L. **Cálculo Numérico Aplicado à Engenharia Elétrica com MATLAB** [recurso eletrônico]. Florianópolis: Publicações do IFSC, 2019. 137 p. ISBN 978-85-8464-138-3.

Sobre Cálculo Numérico

Engenharia é a geração e a aplicação do conhecimento para inventar, construir e melhorar soluções que promovam avanços na qualidade de vida da sociedade. Portanto, o(a) engenheiro(a) pode ser definido como aquele(a) que conjuga conhecimentos multidisciplinares e os aplica tendo em conta a sociedade, a técnica, a economia e o meio ambiente.

Engenheiros constroem suas hipóteses tendo por início os princípios de conservação da natureza, os quais levam a equações de continuidade e de balanço [1]. O que o engenheiro faz é observar e entender um fenômeno ou problema, traduzi-lo numa modelagem matemática coerente, aplicar técnicas numéricas eficientes para resolvê-lo e, por fim, validar as soluções encontradas. Caso as soluções não sejam adequadas, volta-se à definição do problema. A Figura 2.1 ilustra esse processo iterativo de desenvolvimento da solução.

Nas últimas décadas, tal processo pela busca do conhecimento tornou-se mais simples com a ajuda dos computadores.

Resolvendo problemas da engenharia com computador

Cálculo numérico, métodos numéricos, engenharia assistida por computador, projeto assistido por computador, *computer aided engineering* (CAE), *computer aided design* (CAD), matemática computacional e computação científica. Esses são alguns dos nomes utilizados para designar o campo de estudo interessado na construção de modelos matemáticos e de técnicas de soluções numéricas utilizando computadores para analisar e resolver problemas de engenharia [1]-[3].

Quando a resolução analítica é possível, o problema está resolvido. Entretanto, quanto mais próximo se deseja a modelagem de um problema real, mais complexo será o seu entendimento e, consequentemente, mais complexa a sua formulação matemática.

Utiliza-se a modelagem e a simulação computacional quando o sistema real é de difícil entendimento na sua totalidade. Outros aspectos que as motivam são a redução de custos para a realização de ensaios ou de provas, bem como a necessidade de repetibilidade de resultados ou o estudo de situações críticas.

Figura 2.1: Metodologia de modelagem numérica – processo iterativo.

A observação de situações aguça a busca por conhecimento para entendê-las. Experimentos, testes e buscas por teorias existentes podem ser feitos para aumentar o conhecimento prévio. Dessa forma, a experiência do engenheiro acerca do tema é de grande valia neste momento inicial.

A análise estatística (*numerical design of experiments* – *DoE*) ajuda a definir uma sequência ordenada de ações na definição do problema [4]. Além disso,

procedimentos de análise de sensibilidade podem fornecer uma compreensão abrangente das influências dos diversos parâmetros e suas consequências nos resultados do modelo [5].

Nesta fase inicial, "problema em estudo"conforme Figura 2.1, os objetivos são avaliar a influência dos parâmetros e quantificar a incerteza do modelo. Com base nos resultados das ferramentas do DoE e da análise de sensibilidade, um modelo reduzido com um conjunto menor de parâmetros significativos pode ser obtido. Isso pode implicar em modelos numéricos mais simples e, consequentemente, a dificuldade de resolução pode diminuir.

Uma vez que o problema está claro, a sua representação física e matemática pode ser definida. Dessa forma, novas escolhas devem ser feitas: equações algébricas podem ser lineares (capítulo 5) ou não lineares (capítulo 6); equações diferenciais podem ser ordinárias (capítulo 10) ou parciais; e equações diferenciais podem ser convertidas em equações integrais equivalentes (capítulo 11). A escolha por uma formulação é um equilíbrio entre a abstração física e a correção da solução.

Outras tomadas de decisão devem ser feitas entre a definição do problema e a validação da solução. Cada escolha pode levar a soluções mais ou menos precisas, com mais ou menos esforço computacional.

Uma discussão pertinente é a escolha da linguagem de programação a ser utilizada. É cada vez mais comum encontrar softwares comerciais para simulação, e eles são úteis e geralmente fornecem muitas ferramentas de pós-processamento que ajudam a visualizar os resultados.

Por outro lado, uma desvantagem no uso de software comercial é a falta de controle sobre todas as variáveis que um engenheiro deve abordar durante uma investigação. Assim, o desenvolvimento de programação própria é válido.

Pascal, Fortran e C, entre outras linguagens de programação, exigem um grande conhecimento em programação compilada [6]. Esse foi o paradigma dominante na criação de software até a programação orientada a objetos, sendo as mais populares C ++ e Visual C ++.

Recentemente, surgiu a linguagem interpretada. Ela utiliza simplificações na maneira de se escrever um código. Muitas vezes, isso resulta em um formalismo pobre e no uso de bibliotecas prontas, gerando dependência similar aos softwares comerciais. Como enorme vantagem, a linguagem interpretada permite o engenheiro testar suas ideias sem a necessidade de dedicar um tempo considerável com a rigidez da programação compilada. Matlab [7] e Python [8] são exemplos de linguagens interpretadas.

Além disso, o uso de bibliotecas $open\ source$ também está se tornando uma opção regular. Pode-se encontrar muitos exemplos de ferramentas de código aberto. No entanto, até mesmo um código bem escrito pode conter erros.

Sobre aproximações e erros

Exatidão pode ser definida como o quão próximo os valores calculados ou medidos estão próximos do verdadeiro. Importante não confundir com precisão. Precisão é o quão próximo os valores individuais medidos ou calculados estão uns dos outros.

Como discutido antes, a primeira questão é saber até que ponto o modelo matemático bem representa a situação em estudo. Deve-se ter conhecimentos relativos às etapas de modelagem suficientes para, pelo menos, ter ciência de qual o grau de exatidão se pode conseguir e, consequentemente, qual o erro tolerável.

Associam-se a isto os chamados erros numéricos. Erros numéricos são aqueles causados pelo uso de aproximações para representar quantidades matemáticas exatas. Eles podem ser de dois tipos: erros de arredondamento e de truncamento.

Erros de arredondamento surgem porque os computadores têm limites de tamanho e precisão em sua capacidade de representar números. Em alguns casos, esses erros podem direcionar os cálculos para instabilidades numéricas, fornecendo resultados incorretos ou mal condicionados. Erros de arredondamento podem levar a discrepâncias sutis, difíceis de detectar.

Erros de truncamento resultam do uso de uma aproximação numérica no lugar

de um procedimento matemático exato, geralmente uma tomada de decisão do engenheiro. Um exemplo disso seria um número máximo de repetições em um processo iterativo.

Além disso, na ciência da computação, a eficiência pode ser uma característica relacionada ao número de recursos computacionais utilizados pelo algoritmo, como tempo, espaço e memória. Então, na análise numérica, os erros de discretização são aqueles que resultam do fato de que uma função de uma variável contínua é representada por um número finito de avaliações. De forma geral, os erros de discretização podem ser reduzidos usando um número maior de avaliações. Logo, tem-se o aumento do custo computacional.

Finalmente, outras ferramentas numéricas podem ser usadas para entender melhor a solução numérica obtida. Cita-se como exemplo a regressão, a interpolação e as splines (capítulo 7). Entender o comportamento do problema, não apenas o valor bruto da solução final, também pode ser possível em muitos casos.

Utilizada em uma ampla gama de situações, técnicas de aprendizado de máquina ou de forma genérica, a chamada inteligência artificial, está sendo aplicada para a modelagem numérica de problemas. O uso dessas técnicas é um marco importante, a saber, como interpretar dados sem assistência humana [9].

Em resumo, a metodologia para desenvolver uma representação numérica exige uma grandeza de detalhes e cuidados proporcionais à complexidade da situação em estudo. Simplificações, aproximações e erros são comuns. Entender tais limitações é fundamental para se obter e julgar soluções como adequadas.

A motivação deste livro é ter um material em que a partir de um problema real e entendendo a sua natureza, o leitor possa escolher a ferramenta numérica mais apropriada para a sua resolução.

2.1 Referências

[1] JEWETT, J.; SERWAY, R. **Física para cientistas e engenheiros**. São Paulo: Cengage Learning, 2011. ISBN 9788522111107.

[2] SHEN, W. **An Introduction to Numerical Computation**. Singapura: World Scientific Publishing, 2018. ISBN 9789814730068.

[3] DRISCOLL, T. A.; BRAUN, R. J. **Fundamentals of Numerical Computation**. Filadélfia: SIAM, 2017. ISBN 9781611975079.

[4] ELSER, T. **Factorial Design: Understanding Design of Experiments (DoE) and Applying it in Practice**. Scotts Valley: CreateSpace Independent Publishing Platform, 2017. ISBN 97815429061111.

[5] PARK, I. H. **Design Sensitivity Analysis and Optimization of Electromagnetic Systems: Mathematical and Analytical Techniques with Applications to Engineering**. Singapura: Springer, 2019. ISBN 9789811343643.

[6] BACK, M. **Programação na Engenharia! E agora?**. Belo Horizonte: Editora Bibliomundi Serviços Digitais Ltda, 2020. 148 p. (v. 3).

[7] THE MATHWORKS. **MATLAB**. 2020. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 24 ago. 2025.

[8] ASSOCIAÇÃO PYTHON BRASIL. **Python**. 2020. Disponível em: <https://python.org.br/>. Acesso em: 24 ago. 2025.

[9] AWAD, M.; KHANNA, R. **Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers**. Berkeley: Apress, 2015. 268 p. ISBN 9781430267000.

Python para Cálculo Numérico

Python é considerada uma linguagem de programação de alto nível. Isto significa que entendê-la pode ser mais simples do que entender outras, pois ela se assemelha mais à forma como os humanos se comunicam, enquanto que linguagens de baixo nível têm sua compreensão dificultada por serem mais próximas da forma de comunicação dos computadores. No entanto, como qualquer linguagem, ela exige conhecimento prévio em lógica e algoritmos.

Algumas referências para aprender Python, além do próprio [www.python.org] e o seu guia de estilo para código (PIP8), são:

1. Curso Intensivo de Python [1]
2. *Effective Computation in Physics: Field Guide to Research with Python* [2]
3. Python para análise de dados [3]
4. Plataforma Kaggle [4]

Para o melhor entendimento deste livro, é necessário que o leitor possua, previamente, um conhecimento básico de programação em Python 3, uma vez que aqui o foco não será em conceitos elementares, mas sim aplicações dessa ferramenta para a solução de problemas da Engenharia. Neste capítulo são apresentadas as principais bibliotecas que serão usadas ao longo do livro, em particular a NumPy, a Matplotlib e a SciPy.

3.1 NumPy

A biblioteca Numpy é a base da maior parte dos algoritmos deste livro. Fundamental para agilizar a manipulação dos números, ela permite realizar operações com matrizes, encontrar curvas polinomiais, interpolar, e efetuar quase todo tipo de manipulação numérica.

3.1.1 Arrays

Os chamados Numpy Arrays são estruturas que representam matrizes ou vetores. Eles possibilitam que as operações sejam feitas com strides, o que otimiza significativamente o algoritmo quando comparados às tradicionais listas, dispensando muitas vezes a necessidade de loops que escaneiam as estruturas de dados e selecionam valores de acordo com condicionais.

Os arrays da NumPy, por características construtivas da biblioteca, possuem uma relação com a linguagem C. Ocorre que, em algumas operações, é utilizado código pré-compilado em C para obter uma velocidade maior do que seria possível somente com o Python. Esse sistema está presente em produtos matriciais dessa biblioteca. Apesar da linguagem C realizar um produto de matrizes de forma mais rápida do que o Python, programar esse tipo de operação em C é muito mais complexo e trabalhoso. A NumPy une o melhor de cada linguagem, utilizando a eficiência do C, sem abrir mão da simplicidade da programação em Python.

Por padrão, todas as operações básicas entre arrays são elemento por elemento, assim, é preciso que haja uma especificação ou o uso de funções internas para realizar algumas operações matriciais.

O problema

Escreva um algoritmo que realize o seguinte produto matricial:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

Para escrever esse código, precisamos especificar que a operação não é elemento por elemento. Nesse caso, em vez do operador de multiplicação padrão do Python `*` , utilizaremos o caractere `@` entre o nome das variáves como mostrado a seguir.

Resolução assistida por computador

```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
B = np.array([[10,11,12],
              [13,14,15],
              [16,17,18]])
C = A @ B
print(C)
```

Primeiramente, importamos a NumPy com o nome de np para facilitar o processo de chamar suas funções. Em seguida, usamos a função `numpy.array()` para declarar os arrays a serem multiplicados. Finalmente, a operação entre os arrays é realizada e o resultado é armazenado na variável C. Outra forma de resolver o problema seria utilizando a função `numpy.dot()` que realiza o produto de matrizes.

O resultado do produto é:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 \\ 201 & 216 & 231 \\ 318 & 342 & 366 \end{bmatrix}$$

3.1.2 Manipulações e funções internas

Para poder trabalhar bem com a NumPy é preciso conhecer algumas técnicas de manipulação de arrays e algumas funções internas. A seguir é demonstrado como criar uma matriz 3×5 sem digitar manualmente todos os números.

Primeiramente, se faz o uso da função `numpy.arange(n)` que vai retornar um array unidimensional de n números inteiros crescentes, começando com o 0 e terminando com $n-1$. No exemplo a seguir $n = 15$:

```
import numpy as np

a = np.arange(15)
print(a)
```

A variável a recebeu o seguinte array:

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

Em seguida, para mudar o formato desse array para o formato 3×5, utilizamos o método `reshape()`, que encaixará os dados no formato que se quer.

```
import numpy as np

a = np.arange(15)
print(a)
a = a.reshape(3, 5)
print(a)
```

Finalmente, a variável a recebeu a sua versão após passar pelo método `reshape(3, 5)`, resultando no seguinte array.

array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14]])

Para conferir o formato de uma matriz, chamamos o atributo `.shape`, para ver a quantidade de valores dentro da matriz, chamamos o atributo `.size` e para ver o número de linhas, utilizamos a função `len()`, conforme a seguir:

```
import numpy as np

a = np.arange(64)
a = a.reshape(16, 4)
print(a)
print(a.shape)  # Formato de a: 16 por 4
print(a.size)  # Quantidade de valores em a: 64
print(len(a))  # Quantidade de linhas em a: 16
```

Outra forma de solução, seria usar os índices para selecionar partes dos arrays, isto é, realizar o seu fatiamento.

O problema

Com o array anterior, forme um array novo composto pelos elementos das linhas 2 e 3 e das colunas 4 e 5.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 9 \\ 13 & 14 \end{bmatrix}$$

Para se resolver este problema, serão utilizadas as técnicas de fatiamento de estruturas de dados. É importante lembrar que os índices no Python começam em zero, ou seja, o elemento a_{11} da matrix(array) a é especificado como `a[0, 0]`. Se a especificação fosse `a[1, 1]`, o elemento a_{22} da matrix, que nesse caso é o 6. Pode-se especificar linhas e colunas. Por exemplo, para especificar a primeira linha da matrix, escreve-se `a[0, :]`, ou seja, para o array a, a linha 0 e todas as colunas. Para especificar a primeira coluna da matrix, escreve-se `a[:, 0]`, ou seja, para o array a, todas as linhas e a coluna 0.

Caso seja necessário fatiar uma parte dos elementos de uma linha ou coluna, utiliza-se o índice do primeiro elemento antes dos ':' e, para o último elemento, coloca-se o seu índice acrescentado em 1. Por exemplo, se desejar selecionar os 3 elementos do meio da última linha de a (11,12,13), a escrita será `a[2, 1:4]`, ou seja, para a linha de índice 2 (os índices das 3 linhas são 0, 1 e 2), serão selecionados os elementos que ocupam os lugares dos índices 1 ao 3. Note que em vez de colocar o número 3 após os ':', foi colocado 4, pois é 3+1. Na especificação de intervalos, é comum a necessidade de adicionar 1 ao índice do último elemento, devido às características internas do interpretador Python.

```
1 import numpy as np
2
3 a = np.arange(15)
4 print(f'a =\n {a}')  # \n é usado para quebra de linha
5 a = a.reshape(3, 5)
6 print(f'a =\n {a}')  # f-strings facilitam a formatação
7 b = a[1:3, 3:5]
8 print(f'b =\n {b}')
```

O resultado foi o seguinte array:

```
array([[ 8, 9],
       [13, 14]])
```

A função `numpy.linspace(start, stop, num)` também poderia ter sido utilizada no lugar de `numpy.arange(n)` para gerar os números do array a. Ela opera recebendo os valores de início($start$) e parada($stop$) do intervalo, bem como o número de valores que ocuparão esse intervalo(num). Como retorno, `numpy.linspace(start, stop, num)` entrega um array de números linearmente espaçados de acordo com o intervalo e o número especificados. Essa função é muito utilizada para plotagem, como será demonstrado mais à frente. Outras funções são `numpy.zeros` e `numpy.ones` que retornam arrays compostos por zeros ou uns de acordo com o tamanho e o formato especificado.

3.2 Matplotlib

A Matplotlib é uma biblioteca especializada na confecção de gráficos. Os princípios básicos estão apresentados a seguir.

3.2.1 Plotagem 2D

A principal função desta biblioteca, para o uso que se pretende fazer, é a função `matplotlib.pyplot.plot(x, y)` que serve para a plotagem de gráficos 2D. x e y são as coordenadas dos pontos a serem plotados, podendo ser apenas as coordenadas de um ponto ou arrays com as coordenadas de vários pontos. O exemplo a seguir mostra a plotagem simples de duas funções: uma senoidal e uma cossenoidal. Para agilizar a confecção do código a biblioteca NumPy, foi importada como np e o submódulo `pyplot` da Matplotlib, que contém a função `plot()`, como plt. No exemplo, os valores de x foram obtidos pelo uso de `linspace(start, stop, num)`, em que $start = -\pi$, $stop = \pi$ e a quantidade de valores $num = 35$. A Figura 3.1 mostra o gráfico gerado.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 35)  # 35 valores entre -pi e pi
y_sen = np.sin(x)  # array senos dos valores de x
y_cos = np.cos(x)  # array cossenos dos valores de x
plt.plot(x, y_sen, label='seno')
plt.plot(x, y_cos, label='cosseno')
plt.xlim(-np.pi, np.pi)
plt.xlabel('Ângulo [rad]')
plt.ylabel('Função trigonométrica(x)')
plt.grid(True)
plt.legend()
plt.show()  # mostra o gráfico
print(f'x =\n{x} ')
print(f'y_sen =\n{y_sen} ')
print(f'y_cos =\n{y_cos} ')
```

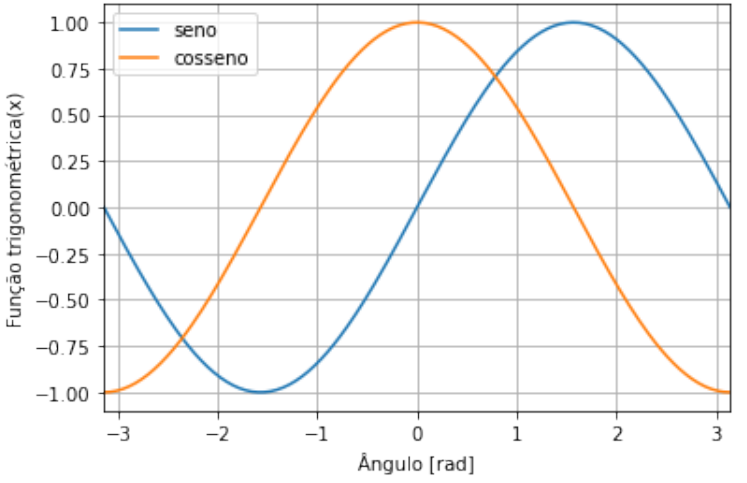


Figura 3.1: Plotagem de função senoidal e cossenoidal

Além da plotagem simples, é possível plotar mais de um gráfico por figura com o uso da ferramenta `matplotlib.pyplot.subplots()`. Dessa forma, em vez de comandar a plotagem utilizando o $plt.$, separa-se a figura em 2 gráficos ($ax1$ e $ax2$) e especificam-se os parâmetros de cada gráfico separadamente. O algoritmo a seguir ilustra o uso `matplotlib.pyplot.subplots()` utilizando os mesmos dados do exemplo anterior. A Figura 3.2 mostra os gráficos.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 35)
y_sen = np.sin(x)
y_cos = np.cos(x)

fig, (ax1, ax2) = plt.subplots(2)
ax1.grid(True)
ax1.plot(x, y_sen)
ax1.set(title='Funções Trigonométricas', ylabel='sen(x)')
ax1.set_xlim(-np.pi, np.pi)

ax2.grid(True)
ax2.plot(x, y_cos, color='orange')
ax2.set(xlabel='Ângulo [rad]', ylabel='cos(x)')
ax2.set_xlim(-np.pi, np.pi)
```

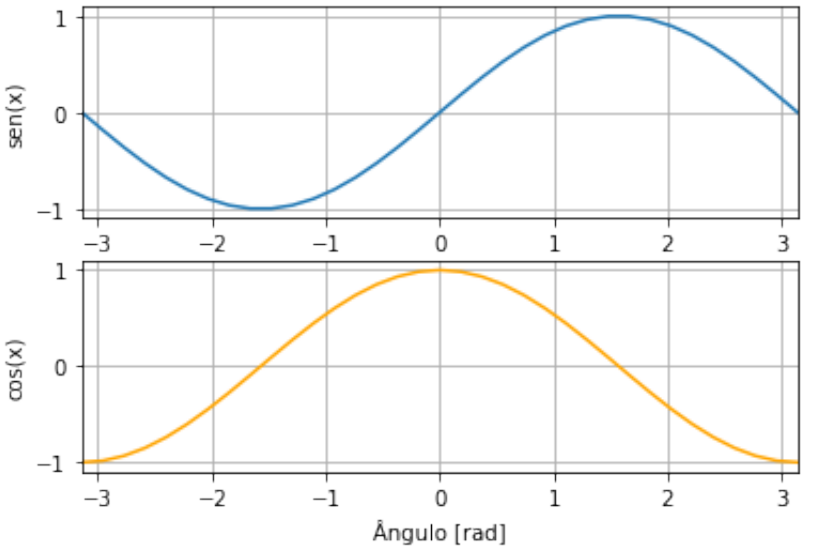


Figura 3.2: Uso do subplots para plotar dois gráficos adjacentes

3.2.2 Plotagem 3D

Existem diversas formas de se fazer gráficos tridimensionais usando a Matplotlib. Note que aqui `np.arange(start, stop, step)` foi usado de uma maneira diferente da que foi mostrada anteriormente. Nesse caso, ele funciona

quase como o `np.linspace(start, stop, num)`, porém, em vez de se especificar a quantidade de valores no intervalo(num), se especifica a distância entre os valores($step$). Outra ferramenta fundamental para esta plotagem é `plt.meshgrid(X, Y)` que cria uma grade com as coordenadas dos arrays X e Y, representando um plano bidimensional. Finalmente, os valores de Z são definidos em função trigonométrica de X e Y e, com todas as coordenadas; a superfície tridimensional é plotada no gráfico, como mostrado na Figura 3.3.

```
import matplotlib.pyplot as plt
import numpy as np

X = np.arange(-np.pi, np.pi, 0.5)
Y = np.arange(-np.pi, np.pi, 0.5)
# Plano bidimensional
X, Y = np.meshgrid(X, Y)
f_xy = np.sqrt(X ** 2 + Y ** 2)
# Plano tridimensional
Z = np.cos(f_xy)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X, Y, Z, cmap='cool')  # plotagem da superfície 3D
plt.show()
```

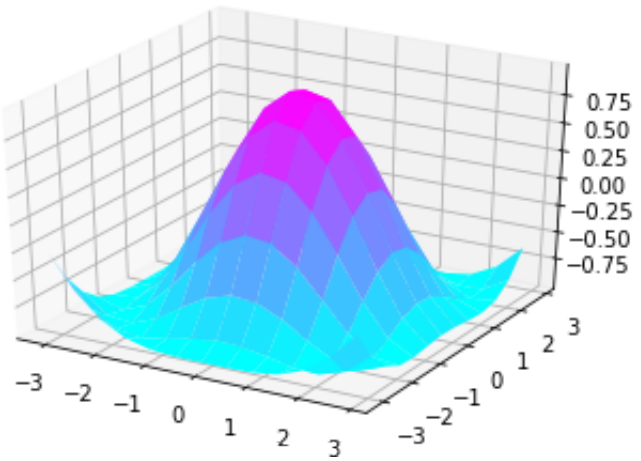


Figura 3.3: Plotagem 3D de Z em função de X e Y

3.3 SciPy

Baseada na NumPy, a SciPy possui ferramentas para operações mais complexas. Suas inúmeras funções internas são muito úteis para a resolução de problemas, como funções de interpolação, integração e derivação, transformadas de Fourier, entre outras. Além do mais, seu módulo de constantes contém diversas constantes físicas e matemáticas, dispensando a necessidade de escrevê-las manualmente no código.

O exemplo a seguir foi retirado do site oficial da SciPy e mostra como utilizá-la para a interpolação de dados:

```
1  import matplotlib.pyplot as plt
2  from scipy import interpolate
3
4  x = np.arange(0, 10)
5  y = np.exp(-x / 3.0)
6  f = interpolate.interp1d(x, y)
7  xnew = np.arange(0, 9, 0.1)
8  ynew = f(xnew)   # use 'interp1d'
9
10 plt.plot(x, y, 'o', label='Dados')
11 plt.plot(xnew, ynew, '-', label='Interpolação')
12 plt.legend()
13 plt.show()
14 Adaptado de [4]
```

Primeiramente, os dados que servirão de base para a interpolação são gerados com `np.arange(n)` e `np.exp()`, representando um comportamento exponencial $f(x) = a^{x/3}$.

Em seguida, `interpolate.interp1d(x, y)` retorna uma função resultado da interpolação para f. Novos dados são gerados para $xnew$ e $ynew$ usando a função f e, finalmente, todos os dados são plotados, conforme Figura 3.4.

No decorrer do livro, outros exemplos de uso dessas três bibliotecas serão demonstrados.

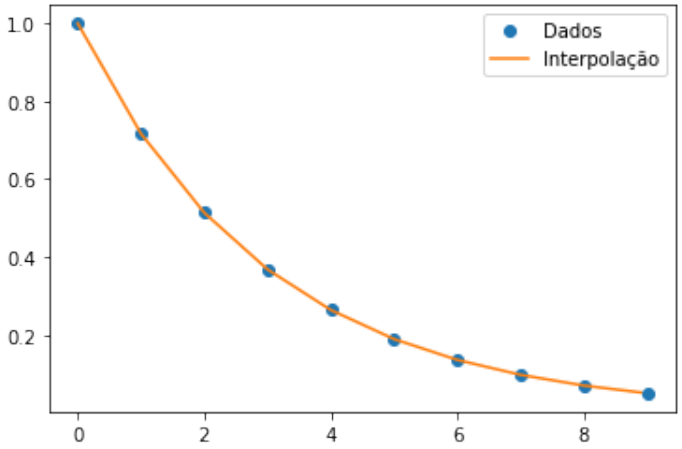


Figura 3.4: Interpolação com a Scipy

3.4 Pandas

Pandas é uma biblioteca essencial para manipulação e análise de dados em Python. Com ela, é possível trabalhar com estruturas de dados como DataFrames e Series, que facilitam o processamento e a análise de grandes volumes de dados. A biblioteca oferece ferramentas poderosas para limpeza, agregação, transformação e visualização de dados, tornando-a um dos pilares para quem trabalha com análise de dados. O exemplo a seguir ilustra como utilizar o Pandas para carregar e manipular um conjunto de dados simples em formato CSV. O código acima começa importando a biblioteca `pandas` com o alias `pd`. Em seguida, carrega um arquivo CSV contendo dados em um DataFrame, que é uma estrutura de dados bidimensional com rótulos de linha e coluna. A função `head()` exibe as primeiras linhas do DataFrame para inspeção rápida, e a função `mean()` é utilizada para calcular a média de uma coluna específica, destacando a capacidade do Pandas para realizar operações agregadas de forma simples. O Pandas é amplamente utilizado em diversos campos, como ciência de dados, finanças, e pesquisa, devido à sua flexibilidade e desempenho em manipulação de dados. Ao longo deste livro, exemplos mais avançados de uso do Pandas serão explorados para demonstrar sua aplicabilidade em diferentes contextos.

```python
import pandas as pd

# Carregar dados de um arquivo CSV
df = pd.read_csv('dados.csv')
# Exibir as primeiras linhas do DataFrame
print(df.head())
# Calcular a média de uma coluna específica
media = df['coluna'].mean()
print("Média da coluna:", media)
```

3.5 Referências básicas

[1] MATTHES, E. **Curso Intensivo de Python**: uma introdução prática e baseada em projetos à programação. São Paulo: Novatec, 2017. 656 p.

[2] SCOPATZ, A.; HUFF, K. D. **Effective Computation in Physics**: field guide to research with Python. Sebastopol: O'Reilly Media, 2015. 552 p.

[3] MCKINNEY, W. **Python para análise de dados**: tratamento de dados com Pandas, NumPy e IPython. São Paulo: Novatec Editora, 2019. 616 p.

[4] THE SCIPY COMMUNITY. **scipy.interpolate.interp1d**. SciPy, 2020. Disponível em: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html>. Acesso em: 24 ago. 2025.

3.6 Referências Complementares

[5] THE NUMPY COMMUNITY. **NumPy v1.21 Manual**. NumPy, 2020. Disponível em: <https://numpy.org/doc/stable/>. Acesso em: 24 ago. 2025.

[6] THE SCIPY COMMUNITY. **SciPy documentation**. SciPy, 2020. Disponível em: <https://docs.scipy.org/doc/scipy/reference/>. Acesso em: 24 ago. 2025.

[7] MATPLOTLIB DEVELOPMENT TEAM. **Matplotlib**: visualization with Python. Matplotlib, 2020. Disponível em: <https://matplotlib.org/stable/index.html>. Acesso em: 24 ago. 2025.

Raízes

As equações da engenharia são usualmente baseadas em princípios de conservação, como por exemplo as leis de Kirchhoff para análise de circuitos elétricos. Esses princípios são explicados por funções de continuidade e de balanço que explicitam o comportamento do fenômeno em estudo.

Caso haja uma única função que bem represente esse fenômeno, uma possível solução é buscar a sua raiz. Uma raiz ou zero da função consiste em determinar os pontos de intersecção da função com o eixo das abscissas no plano cartesiano.

Para tal, baseia-se no teorema de Bolzano sobre existência e unicidade:

$$Assumindo\ que\ f : [a,b] \to \mathbb{R}\ , e\ f(x)\ é\ uma\ função\ contínua\ tal\ que\ f(a) \cdot f(b) < 0\ , então,\ existe\ x^* \in (a,b)\ tal\ que\ f(x^*) = 0.\ [1]$$

Este capítulo apresenta duas maneiras para a aproximação numérica na busca de raízes: um método intervalar (bissecção) e um método aberto (Newton-Raphson).

Quando o problema em estudo apresenta aspectos de interdependência entre variáveis, certamente resultará em um conjunto de funções acopladas. Elas devem ser resolvidas simultaneamente por meio de um sistema de equações. Esse assunto será tratado nos capítulos sobre sistemas lineares e sistemas não-lineares.

Importante observar a necessidade de distinção entre raiz, valores máximos e valores mínimos de uma função. Processos para a determinação de valores máximos e mínimos são denominados otimização – tema não abordado neste livro. A busca por raízes pode ser transformada em otimização e vice-versa. Para tal, manipulações na equação devem ser realizadas. A Figura 4.1 ilustra essa diferença.

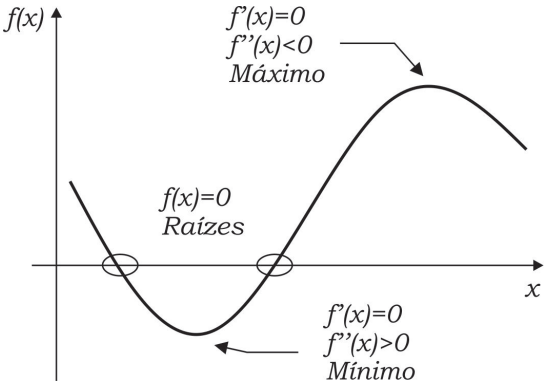
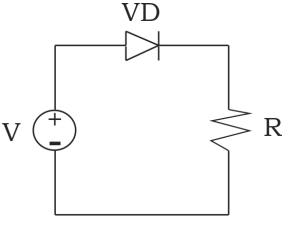


Figura 4.1: Raizes, máximos e mínimos de uma função

O problema

Calcule a corrente que circulará no circuito eletrônico ilustrado.

Admitindo um diodo de silício próximo da idealidade, sua queda de tensão é de 0,7 V.



Por análise de malha:

$$I = \frac{(V - V_d)}{R}. \qquad (4.1)$$

A equação 4.1 é uma equação algébrica com simples resolução analítica. Considerando V = 24 V e R = 10 Ω, tem-se I = 2,3300 A.

Saindo da idealidade, sabe-se que a operação do diodo varia com a sua temperatura. Um possível modelo que relaciona a tensão (V_d), a corrente (I_d) e a temperatura de operação (T) no diodo pode ser:

$$V_d = \left(n\frac{kT}{q}\right) log\left(\frac{I_d}{I_{CR}} + 1\right) \qquad (4.2)$$

onde I_{CR} é a corrente de condução reversa, n é o coeficiente de emissão, k é

a constante de Boltzmann e q é a carga do elétron [2][3].

Entendendo que a corrente no diodo é a corrente no circuito, por análise de malha, tem-se:

$$V = V_d + RI_d, \tag{4.3}$$

$$\text{então } V_d = \left(n\frac{kT}{q}\right) log\left(\frac{I_d}{I_{CR}} + 1\right) + RI_d$$

A partir daqui, por mais que se tente manipular, não se consegue resolver a equação de forma explícita para I_d. O termo $log(I_d/I_{CR} + 1)$ não permite. Assim sendo, uma resolução direta fica inviável.

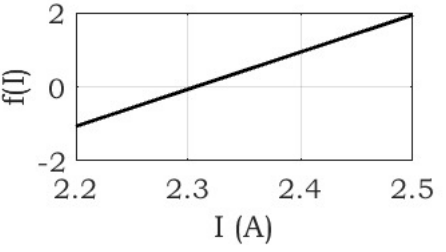
Uma maneira alternativa para a resolução do problema é subtrair dos dois lados da igualdade o termo a sua esquerda, adquirindo uma nova função:

$$f(I) = \left(n\frac{kT}{q}\right) log\left(\frac{I_d}{I_{CR}} + 1\right) + RI_d - V \tag{4.4}$$

Agora, a resposta para o problema é encontrar um valor para I_d que torna a função igual a zero, portanto, a sua raiz.

Uma maneira para isto é graficar o comportamento de $f(I)$. Por inspeção visual e pouco precisa, obtém-se I. Admitindo um diodo 1N4001 com I_{CR} = 31,9824 nA, n = 2, k = 1,3806 x 10^{23} J/K, T = 300K (26,85 °C) e q = 1,6022 x 10^{19} C, consegue-se o comportamento ilustrado.

A corrente tem valor aproximado de 2,3000 A. Assim sendo, conforme (4.1), obtém-se V_d de 0,9353 V.



Pode-se utilizar métodos numéricos para se resolver o problema com mais precisão, caso necessário. Existem inúmeros métodos para a identificação

de raízes, que podem ser divididos em duas famílias: métodos intervalares e métodos iterativos. Talvez os mais conhecidos e eficientes dentro dessas famílias para o objetivo aqui proposto sejam os métodos de bissecção e Newton-Raphson, respectivamente.

4.1 Método intervalar: Bissecção

De forma geral, se $f(x)$ for real e contínua no intervalo $[x_1, x_2]$ e $f(x_1)$ e $f(x_2)$ tiverem sinais opostos, portanto:

$$f(x_1) \times f(x_2) < 0, \tag{4.5}$$

$$x = \frac{x_1 + x_2}{2}$$

então, existe pelo menos uma raiz x real entre x_1 e x_2.

O método intervalar de bissecção utiliza esee entendimento. A nova solução x é obtida pela média entre x_1 e x_2. A determinação do novo intervalo é feita pela checagem de (4.5) para x, x_1 e x_2. O critério de parada pode ser definido como a diferença de $f(x)$ entre duas iterações subsequentes ou pela sua proximidade com o zero. A Figura 4.2. ilustra três iterações do método intervalar de bissecção.

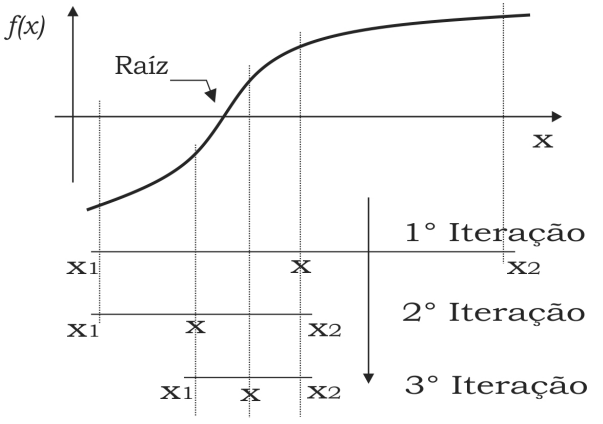


Figura 4.2: Método de Bissecção.

O método da bisseção é um procedimento iterativo que garante a convergência para uma raiz real dentro de um intervalo em que ocorre uma mudança de

sinal. Essa característica faz com que ele seja um dos métodos numéricos mais seguros para a resolução de equações não lineares, especialmente quando a continuidade da função é garantida. No entanto, apesar de sua confiabilidade, o método apresenta uma taxa de convergência relativamente lenta, uma vez que a precisão da solução aumenta apenas de forma linear a cada iteração. Além disso, ele pode não ser eficiente para funções que possuam múltiplas raízes dentro do intervalo considerado, pois sempre localizará apenas uma delas. Ainda assim, sua simplicidade e robustez o tornam uma ferramenta essencial para a solução de equações em diversas aplicações científicas e de engenharia.

A função de bissecção pode ser expressa através de um pseudocógico:

```
1
2  FUNCTION Bisseccao(x1, x2, TOL, iter=16)
3      IF f(x1) * f(x2) > 0 THEN
4          PRINT "Nenhuma raíz encontrada."
5          RETURN
6      END IF
7
8      c = 0  // Contador de iterações
9      ERRO = abs(x2 - x1)  // Erro inicial
10
11     WHILE ERRO > TOL AND c < iter DO
12         hp = (x1 + x2) / 2.0
13
14         IF f(hp) = 0 THEN
15             RETURN {"raiz": hp, "iterações": c}
16         ELSE IF f(x1) * f(hp) < 0 THEN
17             x2 = hp
18         ELSE
19             x1 = hp
20         END IF
21
22         ERRO = abs(x2 - x1)  // Atualiza o erro
23         c = c + 1
24     END WHILE
25
26     RETURN {"raiz": hp, "iterações": c}
27 END FUNCTION
```

Adaptado de [5]

O código em Python pode ser representado da seguinte maneira:

```
from math import log, e  # importa função logarítimo neperiano e a cte de euler

def f(Id,  # Corrente no diodo (A)
        n=2,  # Coeficiente de emissão
        k=1.3806 * (10 ** (-23)),  # Constante de Boltzmann
        V=24,  # Tensão da fonte (V)
        T=300,  # Temperatura de operação (K)
        q=1.6022 * (10 ** (-19)),  # Carga do elétron
        Icr=31.9824 * (10 ** (-9)),  # Corrente de condução reversa (A)
        R=10):  # Resistência (Ohms)

    return (n * (k * T / q)) * m.log((Id / Icr) + 1, m.e) + R * Id - V

def bisseccao(x1,  # x1 início do intervalo
              x2,  # x2 fim do intervalo
              TOL,  # erro tolerado
              iter=16):  # número máximo de iterações

    hp = (x1 + x2)/2  # Ponto médio entre os valores x1 e x2
    if f(x1) * f(x2) > 0:
        print("Nenhuma raíz encontrada.")  # nenhuma raíz.
    else:
        c = 0  # variável contador
        ERRO = abs(f(x2) - f(x1))  # diferença entre os valores de y
        while ERRO > TOL or c < iter:  # loop iterativo com critérios de parada
            hp = (x1 + x2) / 2.0
            if f(hp) == 0:
                return [hp, c]
            elif f(x1) * f(hp) < 0:
                x2 = hp
                c += 1  # contagem
            else:
                x1 = hp
            ERRO = abs(f(x2) - f(x1))
        return {"hp": hp, "iteração": c}  # raíz da função; número de iterações

resp = bisseccao(x1=2, x2=2.5, TOL=0.0001, iter=16)
print(f'raíz aprox {resp["hp"]:.4f}')
print(f'O número de iterações foi {resp["iteração"]}')
```

Adaptado de [5]

A Tabela 4.1. apresenta o resultado numérico para o problema do circuito eletrônico. A função em estudo é a eq. (4.4).

Os valores iniciais de x_1 e x_2 foram adotados tendo por base a experiência prévia sobre o problema, no caso a resolução analítica próximo à idealidade. Aqui foi usado [2,0000 2,5000] A.

Tabela 4.1. Resolução pelo método de bissecção

Interação	x_1	x_2	x	$f(x_1)$	f(x)	Erro
1	2,0000	2,5000	2,2500	-3,0719	-0,5658	5,0115
2	2,2500	2,5000	2,2500	-0,5658	-0,5658	2,5054
3	2,2500	2,3750	2,3750	-0,5658	0,6870	1,2528
4	2,2500	2,3125	2,3125	-0,5658	0,0606	0,6264
5	2,2812	2,3125	2,2812	-0,2526	-0,2526	0,3132
...						
16	2,3064	2,3065	2,3064	-0,0001	-0,0001	0,0002

O conhecimento prévio acelera processos numéricos; entretanto, deve ser usado com cuidado para não induzir falsas ou não tão boas soluções.

O critério de parada utilizado foi a diferença de $f(x)$ entre duas iterações subsequentes iguais ou menores que 0,0002. O valor de x na 16^a iteração é de 2,3064 A. Com esse valor, conforme (4.1), Vd = 0,9355 V.

Destaca-se aqui o cuidado com o número de algarismos significativos. O cálculo realizado, bem como a Tabela 4.1, foram truncados em quatro algarismos e podem incutir em erro. O erro por arredondamentos mal feitos em aproximações numéricas é bem comum. Aqui, uma forma de mitigá-lo é reduzir o erro tolerado ($<$ 0,0002). Isso aumenta o esforço computacional.

4.2 Método iterativo: Newton-Raphson

A derivada de primeira ordem de uma função ($f'(x)$) é equivalente à inclinação da função. Assim sendo, uma aproximação da raiz é:

$$x_{(i+1)} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4.6}$$

Dessa forma, a informação de onde o eixo x será atravessado por uma reta tangente ao ponto $[x_i\ ,\ f(x_i)]$ pode ser utilizada como aproximação para a raiz de $f(x)$.

O método de Newton-Raphson usa esse princípio geométrico, conforme ilustrado na Figura 4.3. A cada iteração, um novo x_{i+1} é determinado.

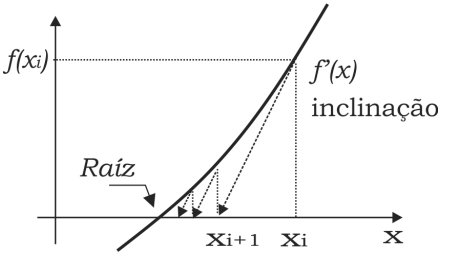


Figura 4.3: Método de Newton-Raphson.

Uma das principais vantagens do método de Newton-Raphson é sua rápida taxa de convergência, especialmente quando a aproximação inicial está próxima da raiz exata. Devido à sua natureza quadrática, o erro tende a diminuir significativamente a cada iteração, tornando-o um dos métodos mais eficientes para a resolução de equações não lineares. No entanto, essa eficiência vem acompanhada de certas limitações. Caso a derivada da função seja muito pequena ou nula em algum ponto do processo iterativo, o método pode falhar, resultando em aproximações divergentes ou em oscilações indesejadas. Além disso, se a função apresentar pontos críticos próximos à raiz buscada, a convergência pode se tornar instável, exigindo uma análise cuidadosa da função antes de sua aplicação.

A função de Newton-Raphson pode ser expressa através de um pseudocógico:

```
FUNCTION NewtonRaphson(x, maxit=10, TOL=0.0001)
    a = f(x) / d(x)  // Cálculo inicial de a
    iter = 0  // Contador de iterações
    WHILE abs(a) > TOL AND iter < maxit DO
        a = f(x) / d(x)  // Atualiza o valor de a
        x = x - a  // Atualiza o valor de x utilizando o método de Newton-Raphson
        iter = iter + 1  // Incrementa o contador de iterações
    END WHILE
    RETURN [x, iter]  // Retorna a raiz e o número de iterações
END FUNCTION
```

Adaptado de [6]

Com essa estrutura bem compreendida, o código em Python pode ser representado da seguinte forma:

```python
import math as m
from scipy.misc import derivative
import numpy as np

def f(Id,  # Corrente no diodo (A)
        n=2,  # Coeficiente de emissão
        k=1.3806 * (10 ** (-23)),  # Constante de Boltzmann
        V=24,  # Tensão da fonte (V)
        T=300,  # Temperatura de operação (K)
        q=1.6022 * (10 ** (-19)),  # Carga do elétron
        Icr=31.9824 * (10 ** (-9)),  # Corrente de condução reversa (A)
        R=10):  # Resistência (Ohms)

    return (n * (k * T / q)) * m.log((Id / Icr) + 1, m.e) + R * Id - V

def d(x):
    return derivative(f, x)  # derivada função da SciPY

def newtonRaphson(x,  # x inicial
                  maxit=10,  # número máximo de iterações
                  TOL=0.0001):  # erro tolerado

    a = f(x) / d(x)
    iter = 0
    while abs(a) > TOL and iter < maxit:  # critério de parada
        a = f(x) / d(x)
        # x(i+1) = x(i) - f(x) / f'(x)
        x = x - a
```

```
29          iter += 1
30      return [x, iter]
31
32 # Exemplo
33 x0 = 2  # Valor inicial
34 ans = newtonRaphson(x0)
35 raiz = ans[0]
36 print(f'A raiz é {raiz:.5f}. Com {ans[1]} iterações.')
```

Adaptado de [6]

A função em estudo é (4.3). A sua derivada é

$$f'(I) = \frac{0,4343\left(n\frac{kT}{q}\right)}{(I_d + I_{CR})} + R \tag{4.7}$$

O valor inicial adotado foi $x = 2$ A. O erro tolerado foi obtido na terceira iteração. O esforço computacional necessário aqui é menor que o dispendido no método de bissecção. A Tabela 4.2. apresenta o resultado numérico para o problema proposto.

Tabela 4.2. Resolução pelo Método de Newton-Raphson

Iteração	x	f(x)	f'(x)	Erro
1	2,000	-3,0719	10,0112	0,0133
2	2,3068	0,0039	10,0097	0,0002
3	2,3064	0,0000	10,0097	<0,0001

Em contrapartida, nem sempre é fácil obter a derivada de uma função. Erros de convergência também podem acontecer quando houver características peculiares do comportamento da derivada da função em estudo. São exemplos de dificuldades: a função possuir regiões paralelas ou quase perpendiculares ao eixo x (inclinação próxima a 0° ou 90°). Nesses casos, a derivada pode apresentar problemas numéricos.

4.3 Exercícios propostos

1. Calcule o comprimento do cabo (C) entre duas torres de transmissão (i.e. a catenária) [4]. A distância entre as torres é de d = 500m. A flecha máxima permitida é f_{max} = 50m. Flecha é a distância vertical entre uma reta que liga os dois pontos de fixação. A flecha (f) depende do comprimento do vão (d) e da tração (C) aplicada ao cabo. O seu modelo matemático pode ser:

$$f = C.\left[cosh\left(\frac{d}{2C}\right) - 1\right] \tag{4.8}$$

Resposta: $633,1621$m

2. Um retificador de meia onda a diodo alimenta uma carga indutiva-resistiva (f = 1 kHz, L = 100 mH e R = 1 kΩ). Encontre o ângulo para o qual a corrente I_d no diodo se anula. Considere o seguinte modelo matemático:

$$I_d = sin(\beta - \phi) + sin(\phi)e^{(-\frac{\beta}{tan(\phi)})} \tag{4.9}$$

$tan(\phi) = 2\pi f.L/R$ Resposta: $\beta = 212,2284°$

3. Determine o ponto de operação de um transistor TBJ, encontrando os valores de Tensão Base-Emissor (Vbe) e Corrente de Coletor (Ic). Sabendo que o valor da corrente I_S é de $1.10^{-15}A$ e da tensão V_T é de $25.10^{-3}V$, o ponto de operação é determinado pela seguinte equação:

$$I_C = I_S.(e^{\frac{V_{BE}}{V_T}} - 1) \tag{4.10}$$

onde: $V_{BE} = V_T.ln(\frac{I_S}{I_C} + 1)$

Resposta: $V_{BE} = 0,6884V \quad I_C = 908,5538uA$

4. Encontre a raiz da equação transcendental. Determine o valor de x que satisfaz a seguinte equação:

$$x^3e^x - 4x + 2 = 0 \tag{4.11}$$

Resposta: $x = 0,5210$

5. Encontre a raiz da equação transcendental. Determine o valor de x que satisfaz a seguinte equação:

$$xe^x = 2 \tag{4.12}$$

Resposta: $x = 0,8526$

4.4 Referências básicas

[1] LAY, D. C.; LAY, S. R.; MCDONALD, J. **Álgebra Linear e suas Aplicações**. 5. ed. Rio de Janeiro: LTC, 2018. ISBN 9788521634959.

[2] NILSSON, J. W.; RIEDEL, S. A. **Circuitos Elétricos**. 10. ed. São Paulo: Pearson, 2016. ISBN 9788543004785.

[3] CAMARGO, C. V.; DALPOSSO, G. H. Cálculo numérico computacional: aplicação na microeletrônica. Revista Jr de Iniciação Científica em Ciências Exatas e Engenharia, v. 1, n. 6, p. 7-15, 2018. ISSN 22360093.

[4] SUGDEN, S. J. Construction of transmission line catenary from survey data. Applied Mathematical Modelling, v. 18, n. 5, p. 274-280, 1994. doi: 10.1016/0307-904X(94)90335-2.

[5] DAVID, K. **The Bisection Method With Python Code**. Toronto, 2018. steemit: @dkmathstats. Disponível em: <https://steemit.com/mathematics/@dkmathstats/the-bisection-method-with-python-code>. Acesso em: 24 ago. 2025.

[6] SMIT, A. **Program for Newton Raphson Method**. GeeksforGeeks, 2020. Disponível em: <https://www.geeksforgeeks.org/program-for-newton-raphson-method/>. Acesso em: 24 ago. 2025.

4.5 Referências complementares – Aplicações

[1] SOARES, A. C. M.; VIEIRA, E.; CASARO, M. M. Simulation of a photovoltaic model using bisection method. In: Brazilian Power Electronics Conference. Anais [...]. 2011. p. 807-811. doi: 10.1109/COBEP.2011.6085239.

[2] XING, H.; MOU, Y.; FU, M.; LIN, Z. Distributed Bisection Method for Economic Power Dispatch in Smart Grid. IEEE Transactions on Power Systems, v. 30, n. 6, p. 3024-3035, 2014. doi: 10.1109/TPWRS.2014.2376935.

[3] PETROVIĆ, P. B.; ROZGIĆ, D. Computational effective modified Newton–Raphson algorithm for power harmonics parameters estimation. IET Signal Processing, v. 12, n. 5, p. 590-598, 2017. doi: 10.1049/iet-spr.2017.0573.

Sistemas Lineares

Como dito no capítulo anterior, os problemas da engenharia são usualmente baseados em princípios que levam a equações de continuidade e de balanço. Caso tenhamos uma única função que bem represente esse fenômeno, uma possível solução é buscar a sua raiz.

Quando o problema em estudo apresenta aspectos de interdependência entre variáveis, certamente resultará em um conjunto de funções acopladas que devem ser resolvidas simultaneamente: um sistema de equações. Dessa forma:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad\qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases} \tag{5.1}$$

Esses sistemas podem ser de natureza linear ou não-linear [1]. Aqui serão tratados os sistemas ditos lineares (o próximo capítulo é destinado aos sistemas não-lineares).. Um sistema de equações lineares a coeficientes constantes pode ser representado por:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \quad \text{ou} \quad [A]\{x\} = \{b\}, \tag{5.2}$$

onde a_{nn} são constantes chamadas coeficientes, agrupadas como elementos de uma matriz com n linhas e n colunas que chamaremos de A. b_n são constantes chamadas de termo independente e x_n são as variáveis ou incógnitas. N é o número de variáveis e pode ser qualquer número inteiro positivo. Assim sendo,

as equações representam retas. Qualquer variação nesse formato tornará o sistema não-linear, portanto curvas.

A resolução direta de um sistema linear é dada por:

$$\{x\} = [A]^{-1}\{b\}. \tag{5.3}$$

Os sistemas lineares podem ser classificados de acordo com o seu número de soluções:

- Um sistema dito possível e determinado possui uma única solução (intersecção das retas);
- Um sistema dito possível e indeterminado possui infinitas soluções (retas sobrepostas);
- Um sistema que não possui solução é chamado de impossível ou incompatível (retas paralelas).

As duas últimas situações são chamadas de singularidades. Outra dificuldade é quando as inclinações das retas são tão próximas que o ponto de intersecção é de difícil detecção, os chamados sistemas mal condicionados. Aplicar diferentes abordagens para buscar a solução pode ser necessário por causa dessas dificuldades. Em outras palavras:

Questão 1 – a matriz [A] é inversível ou é não inversível? O segundo caso é chamado de matriz singular.

Questão 2 – alguns problemas lineares são mais custosos de serem resolvidos, pois os erros de arredondamento se propagam de forma mais significativa que em outros problemas. Isso acontece, sobretudo, pela ocorrência de muitos zeros ou números próximos a zero. Nesse caso, chama-se de problemas mal condicionados.

Em resumo: um problema bem-condicionado é um problema cujos erros de arredondamento se propagam de forma menos importante; enquanto

problemas mal condicionados são problemas cujos erros se propagam de forma mais relevante.

Uma métrica possível para medir o condicionamento de uma matriz é:

$$k(A) := \|A\|\|A^{-1}\| \ , \tag{5.4}$$

onde k é o número de condicionamento e $||.||$ é a norma da matriz $[A]$ em estudo. Se $[A]$ é bem condicionada, k é próximo de 1. Se $[A]$ é mal condicionada, k tende a zero. O número de condicionamento é um indicador para a determinação da confiabilidade na solução de sistemas algébricos.

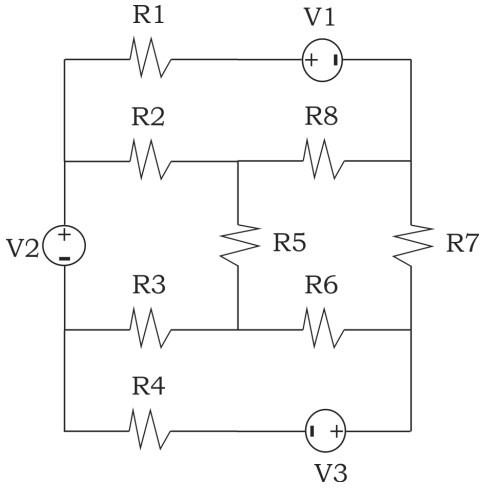
São inúmeros os livros que tratam com mais propriedade dos conceitos de sistemas lineares, como por exemplo [1]. O resumo aqui exposto é o conhecimento mínimo necessário para o adequado entendimento e aplicação dos métodos numéricos que se seguem.

Duas famílias de métodos para a resolução de sistemas lineares são aqui apresentadas: métodos diretos (eliminação de Gauss e decomposição LU) e métodos iterativos (Gauss-Siedel).

Métodos iterativos lidam melhor com problemas mal condicionados, entretanto, podem exigir um esforço computacional maior que os métodos diretos. Os métodos iterativos necessitam de um critério de parada, portanto, levam a uma aproximação da solução.

O problema

Dado o circuito elétrico ilustrado, calcule a corrente no resistor R5. As fontes têm valor de V1 = 10 V, V2 = 15 V e V3 = 10 V. Os resistores são iguais e tem valor de 1 Ω.



O circuito elétrico pode ser equacionado pela lei das tensões de Kirchhoff.
Isto resultará num sistema linear com quatro incógnitas, a saber, as correntes de cada malha.
Um equacionamento possível é:

$$\begin{cases} (R_1+R_2+R_8)I_1 + R_2I_2 + 0I_3 - R_8I_4 = V_1 \\ R_2I_1 + (R_2+R_3+R_5)I_2 + R_3I_3 + R_5I_4 = V_2 \\ 0I_1 + R_3I_2 + (R_3+R_4+R_6)I_3 - R_6I_4 = V_3 \\ -R_8I_1 + R_5I_2 - R_6I_3 + (R_5+R_6+R_7+R_8)I_4 = 0 \end{cases}$$

ou, na forma matricial, $\begin{bmatrix} 3 & 1 & 0 & -1 \\ 1 & 3 & 1 & 1 \\ 0 & 1 & 3 & -1 \\ -1 & 1 & -1 & 4 \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \\ I_3 \\ I_3 \end{Bmatrix} = \begin{Bmatrix} 10 \\ 15 \\ 10 \\ 0 \end{Bmatrix}$

A solução pode ser obtida pela aplicação da regra de Cramer (uso de determinantes) [1] ou de forma direta $x = [A]^{-1}\{b\}$.

Uma forma simples de inverter matrizes em Python é através da biblioteca "NumPy":

Inversão de matrizes

```
import numpy as np

 matrix_a = np.array([[3, 1, 0, -1],[1, 3, 1, 1],[0, 1, 3,
    -1],[-1, 1, -1, 4]])

inverse_matrix_a = np.linalg.inv(matrix_a)
```

Quanto maiores forem as dimensões das matrizes, maior será a dificuldade numérica para o cálculo dos determinantes ou para a inversão da matriz.

Por Cramer [1], a corrente em R5 é de 3,6667 A.

5.1 Método direto: Eliminação de Gauss

$$\left[\begin{array}{ccc:c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

$$\downarrow (1)$$

$$\left[\begin{array}{ccc:c} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a'_{22} & a'_{23} & b'_2 \\ 0 & 0 & a''_{33} & b''_3 \end{array}\right]$$

$$\downarrow (2)$$

$$x_3 = b''_3/a''_{33}$$
$$x_2 = (b'_2 - a'_{23}x_3)/a'_{22}$$
$$x_1 = (b_1 - a_{13}x_3 - a_{12}x_2)/a_{11}$$

Dois passos:

1. Eliminação progressiva – as equações são manuseadas para suprimir uma das variáveis das equações (escalonamento). A última ficará com uma incógnita caso haja solução.
2. Substituição regressiva – a última equação é resolvida diretamente, e o resultado substituído regressivamente nas equações anteriores para determinar as demais incógnitas. O processo é o mesmo usando matriz triangular superior ou inferior.

Para evitar divisões por zero ou por valores muito próximos de zero, que pode levar a erros numéricos, sugere-se uma análise prévia para a troca na ordem das linhas do sistema. É o chamado pivotamento.

O método de Eliminação de Gauss pode ser representado por uma função através de um pseudocógico:

Pseudocódigo

```
FUNCTION GaussEliminacao(A, b, x)
    n = tamanho de b
    x = vetor de zeros com tamanho n

    // Eliminação progressiva
    FOR k = 0 TO n-2
        FOR i = k+1 TO n-1
            IF A[i, k] = 0 THEN
                CONTINUE
            END IF
            fator = A[k, k] / A[i, k]
            FOR j = k TO n-1
                A[i, j] = A[k, j] - A[i, j] * fator
            END FOR
            b[i] = b[k] - b[i] * fator
        END FOR
    END FOR

    // Substituição regressiva
    x[n-1] = b[n-1] / A[n-1, n-1]
    FOR i = n-2 TO 0 STEP -1
        sum_ax = 0
        FOR j = i+1 TO n-1
            sum_ax = sum_ax + A[i, j] * x[j]
        END FOR
        x[i] = (b[i] - sum_ax) / A[i, i]
    END FOR

    RETURN x
END FUNCTION
```

Adaptado de [3]

Com o pseudocógico bem compreendido, pode-se aplicar a solução diretamente:

```
from numpy import array, zeros

def GaussEliminação(A, b):  # cria-se a função do método
    n = len(b)
    x = zeros(n, float)
    # Eliminação progressiva
    for k in range(n-1):
        for i in range(k+1, n):
```

```
            if A[i, k] == 0:
                continue
            fator = A[k, k] / A[i, k]
            for j in range(k, n):
                A[i, j] = A[k, j] - A[i, j]*fator
            b[i] = b[k] - b[i]*fator
    print(f'A =\n {A}')
    print(f'b =\n {b}')
    # Substituição regressiva
    x[n-1] = b[n-1] / A[n-1, n-1]
    for i in range(n-2, -1, -1):
        sum_ax = 0
        for j in range(i+1, n):
            sum_ax += A[i, j] * x[j]
            x[i] = (b[i] - sum_ax) / A[i, i]
    f = 'SOLUÇÃO DO SISTEMA'
    print('-'*(len(f)+32))
    print(f'{f:^50}')
    print('-'*(len(f)+32))
    print(x)
    print('Onde: ')
    for c in range(0, len(x)):
        print(f'\t x[{c}] = {x[c]} \n')

# Uso do método
A = array([[3, 1, 0, -1],
           [1, 3, 1, 1],
           [0, 1, 3, -1],
           [-1, 1, -1, 4]])
b = array([10, 15, 10, 0])
GaussEliminação(A, b)
```

Adaptado de [3]

Por eliminação de Gauss, o resultado obtido é:

$$I = \begin{bmatrix} 2,3333 & 3,3333 & 2,3333 & 0,3333 \end{bmatrix}.$$

Fazendo $I_{R5} = I_2 + I_4$, tem-se 3,6667 A.

5.2 Método direto: Decomposição LU

$$[A]\{x\} = \{b\}$$

$$\downarrow (1)$$

$$[A] = [L] \times [U]$$

$$\downarrow (2)$$

$$[L]\{d\} = \{b\}$$

$$\downarrow (3)$$

$$[U]\{x\} = \{d\}$$

A decomposição LU se trata de três passos:

1. $[A]$ é decomposta em matrizes triangulares inferiores $[L]$ e superiores $[U]$;
2. o vetor intermediário $\{d\}$ é obtido por substituição progressiva;
3. por fim, por substituição regressiva se obtém $\{x\}$.

Recomenda-se o uso de decomposição LU quando a matriz $[A]$ é fixa e se deseja entender o comportamento do problema com variações em $\{b\}$.

Nesse contexto, percebe-se que os novos cálculos de $\{d\}$ e $\{x\}$, feitos por substituição em matrizes triangulares, são computacionalmente mais simples do que a completa inversão de [A].

O método de Decomposição LU pode ser representado por uma função através de um pseudocógico:

```
1  FUNCTION LU_Decomposicao(A, b, x)
2      m = tamanho de A
3      y = vetor de zeros com tamanho m
4      x = vetor de zeros com tamanho m
5      U = A
6      L = matriz de zeros de tamanho [m, m]
7
8      // Construção das matrizes L e U
9      FOR k = 0 TO m-1
10         FOR r = 0 TO m-1
11             IF k = r THEN
12                 L[k, r] = 1
13             END IF
14             IF k < r THEN
15                 fator = A[r, k] / A[k, k]
16                 L[r, k] = fator
17                 FOR c = 0 TO m-1
18                     U[r, c] = A[r, c] - (fator * A[k, c])
19                 END FOR
20             END IF
```

```
        END FOR
    END FOR

    // Resolução das equações matriciais
    y = Resolver_Sistema(L, b)
    x = Resolver_Sistema(U, y)

    RETURN x
END FUNCTION
```

Adaptado de [4]

Com o pseudocógico bem compreendido, pode-se aplicar a solução diretamente em Python:

```
import numpy as np

def LU_Decomposition_method(A, b):  # cria-se a função do método
    m = len(A)
    y = np.zeros(m)
    x = np.zeros(m)
    U = A
    L = np.zeros([m, m])
    for k in range(0, m):
        for r in range(0, m):
            if (k == r):
                L[k, r] = 1
            if (k < r):
                factor = (A[r, k]/A[k, k])
                L[r, k] = factor
                for c in range(0, m):
                    U[r, c] = A[r, c] - (factor * A[k, c])
    A = np.zeros([m, m])
    for r in range(0, m):
        for c in range(0, m):
            for k in range(0, m):
                A[r, c] += (L[r, k] * U[k, c])
    print('A')
    print(A)
    print('L')
    print(L)
    print()
    print('U')
    print(U)
    # RESOLUÇÃO DAS EQUAÇÕES MATRICIAIS
    ###################################
    y = np.linalg.solve(L, b)
    x = np.linalg.solve(U, y)
    ###################################
```

```
35      f = 'SOLUÇÃO DO SISTEMA'
36      print('-'*(len(f)+32))
37      print(f'{f:^50}')
38      print('-'*(len(f)+32))
39      print('x =')
40      print(x)
41      print('Onde: ')
42      for c in range(0, len(x)):
43          print(f'\t x[{c}] = {x[c]} \n')
44
45  # Uso do método
46  A = np.array([[3, 1, 0, -1],
47                [1, 3, 1, 1],
48                [0, 1, 3, -1],
49                [-1, 1, -1, 4]])
50  b = np.transpose([10, 15, 10, 0])
51  LU_Decomposition_method(A, b)
```

Adaptado de [4]

Por decomposição LU, o resultado obtido foi 3,6667 A.

5.3 Método iterativo: Gauss-Siedel

O método de Gauss-Siedel para um sistema algébrico mn pode ser equacionado da seguinte maneira:

$$\begin{cases} x_1^j = (b_1 - a_{12}x_2^{j-1} - \cdots - a_{1n}x_n^{j-1})/a_{11} \\ x_2^j = (b_1 - a_{21}x_1^j - \cdots - a_{2n}x_n^{j-1})/a_{22} \\ \vdots \\ x_n^j = (b_m - a_{m1}x_1^j - \cdots - a_{mn-1}x_{n-1}^j)/a_{mn} \end{cases} \tag{5.5}$$

onde j é a iteração corrente.

Percebe-se que o processo de cálculo x_2^j considera x_1^j. Portanto, utilizam-se as aproximações obtidas dentro da mesma iteração. O critério de parada pode ser:

$$\varepsilon_{a,i} = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| .100\% \leq \varepsilon_s, \tag{5.6}$$

onde i é a variável x em estudo, j é a iteração, ε_a é o erro absoluto e ε_s é um valor de erro desejado.

Computacionalmente, o processo iterativo pode ficar mais eficiente usando matrizes:

$$\{x\} = \{d\} - [C]\{x\} \quad \text{onde}$$

$$\{d\} = \begin{Bmatrix} b_1/a_{11} \\ b_2/a_{22} \\ \cdots \\ b_n/a_{mn} \end{Bmatrix} \text{ e } [C] = \begin{bmatrix} 0 & a_{12}/a_{11} & \cdots & a_{1n}/a_{11} \\ a_{21}/a_{22} & 0 & \cdots & a_{2n}/a_{22} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1}/a_{mn} & a_{m2}/a_{mn} & \cdots & 0 \end{bmatrix} \quad (5.7)$$

.

.

Os métodos iterativos ou de aproximação fornecem uma alternativa aos métodos de eliminação. Como se busca uma aproximação da solução, sua convergência pode ser "relaxada" em problemas mal condicionados. Um exemplo desse relaxamento pode ser um erro global considerável ou mesmo erros admissíveis diferentes para cada variável.

Outra forma de relaxamento é utilizar um fator de peso (λ). Após a determinação de (5.5), tais valores são modificados por uma média ponderada dos resultados da iteração anterior e da atual:

$$x_i^j = \lambda x_i^j + (1 - \lambda) x_i^{j-1} \quad (5.8)$$

onde λ deve ser um valor entre 0 e 2. Caso λ =1, o resultado não é alterado.

- Se λ for entre 0 e 1, tem-se uma média ponderada do valor atual com o anterior.Esse processo é chamado de sub-relaxamento. O subrelaxamento é usado para facilitar a convergência em sistemas mal condicionados.
- Se λ for entre 1 e 2, um valor atual é supervalorizado. Chamada de sobrerrelaxamento, essa supervalorização parte da confiança na convergência do método e deseja-se apenas acelerar o processo

A escolha do valor de λ depende da experiência sobre o problema a ser resolvido.

O método de Gauss-Seidel pode ser representado por uma função através de um pseudocógico:

```
FUNCTION Gauss_Seidel(A, b, x, TOL, maxit)
    m = tamanho de A
    n = tamanho de A[0]
    x = vetor de zeros de tamanho m
    erro = vetor vazio
    k = 0

    DO WHILE k < maxit
        k = k + 1
        PRINT "Iteração: ", k

        FOR r = 0 TO m-1
            soma = 0
            FOR c = 0 TO n-1
                IF c != r THEN
                    soma = soma + A[r, c] * x[c]
                END IF
            END FOR
            x[r] = (b[r] - soma) / A[r, r]
            PRINT "x[", r, "]: ", x[r]
        END FOR

        erro = vetor vazio
        FOR r = 0 TO m-1
            soma = 0
            FOR c = 0 TO n-1
                soma = soma + A[r, c] * x[c]
            END FOR
            erro[r] = ABS(soma - b[r])
            PRINT "Erro em x[", r, "] = ", erro[r]
        END FOR

        IF TODOS(erro <= TOL) THEN
            EXIT
        END IF
    END DO

    PRINT "Convergência alcançada em ", k, " iterações."
    RETURN x
END FUNCTION
```

Adaptado de [5]

Com o pseudocógico bem compreendido, pode-se aplicar a solução diretamente:

```
from numpy import array, zeros

def Gauss_Seidel(matriz_aumentada, vetor):  # função do método
    m = len(matriz_aumentada)
    n = len(matriz_aumentada[0])
    x = zeros((m))
    comp = zeros((m))
    error = []
    TOL = 0.0002  # erro tolerado
    maxit = 5000
    k = 0
    while k < maxit:
        suma = 0
        k = k + 1
        print(f'Iteração: {k}')
        for r in range(0, m):
            suma = 0
            for c in range(0, n):
                if (c != r):
                    suma = suma+matriz_aumentada[r, c] * x[c]
            x[r] = (vetor[r]-suma)/matriz_aumentada[r, r]
            print(f'x[{r}]: {x[r]}')
        del error[:]
        for r in range(0, m):
            suma = 0
            for c in range(0, n):
                suma = suma+matriz_aumentada[r, c] * x[c]
            comp[r] = suma
            dif = abs(comp[r]-vetor[r])
            error.append(dif)
            print(f'Erro em x[{r}] = {error[r]}')
        if all(i <= TOL for i in error) is True:
            break
    print(f'Com {k} iterações.')
    for c in range(0, m):
        print(f'x[{c}]: {round(x[c],4)}')

# Uso do método
matriz_aumentada = array([[3, 1, 0, -1],
                          [1, 3, 1, 1],
                          [0, 1, 3, -1],
                          [-1, 1, -1, 4]], float)
vetor = array([[10], [15], [10], [0]])
Gauss_Seidel(matriz_aumentada, vetor)
```

Adaptado de [5]

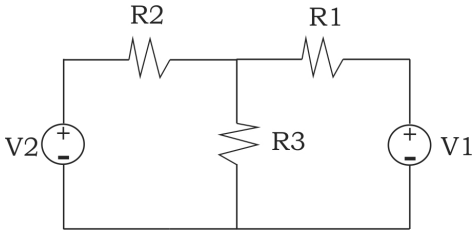
Por Gauss-Siedel, o resultado obtido foi $I_{R5} = 3{,}6667$ A em 17 iterações, com erro na quarta casa decimal. A tabela 5.1 apresenta um extrato do cálculo.

Tabela 5.1. Resolução pelo Método de Gauss-Siedel

#	Correntes	Erros
1	[3,3333 3,8889 2,0370 0,3704]	[1,0000 1,0000 1,0000 1,0000]
5	[2,3236 3,3444 2,3259 0,3263]	[0,0026 0,0020 0,0020 0,0133]
10	[2,3319 3,3350 2,3322 0,3323]	[0.0004 0,0003 0,0003 0,0019]
15	[2,3332 3,3335 2,3332 0,3332]	[0,0000 0,0000 0,0000 0,0002]
17	[2,3333 3,3334 2,3333 0,3333]	[0,0000 0,0000 0,0000 0,0001]

5.4 Exercícios propostos

1. Determine o valor da corrente no resistor R3, sabendo que V1 = V2 = 100 V e R1 = R2 = R3 = 10 Ω.

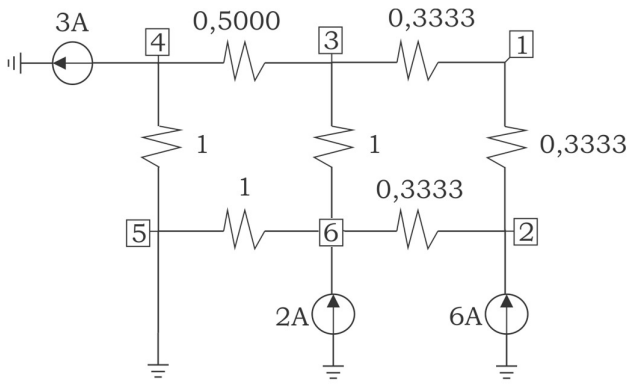


$$\begin{bmatrix} 20 & 10 \\ 10 & 20 \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \end{Bmatrix} = \begin{Bmatrix} 100 \\ 100 \end{Bmatrix}$$

Resposta: IR3 = 6,6667 A

2. Calcule as tensões nos nós tendo por referência o nó 6. O circuito pode ser equacionado da seguinte maneira [2]:

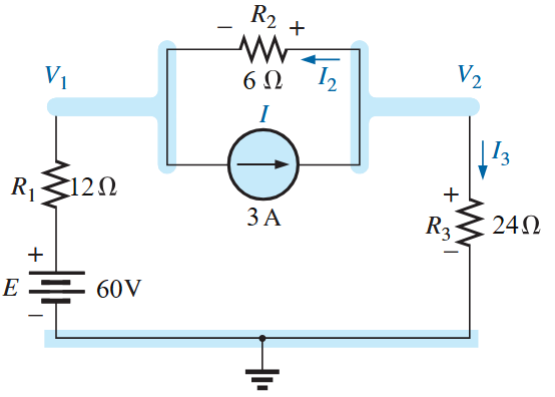
$$\{I\} = [Y]\{V\} \text{ onde,}$$



$$\begin{bmatrix} 20 & 10 \\ 10 & 20 \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \end{Bmatrix} = \begin{Bmatrix} 100 \\ 100 \end{Bmatrix}$$

Resposta: $\{V\} = \{4\ 5\ 3\ 4\ 1\}$ V

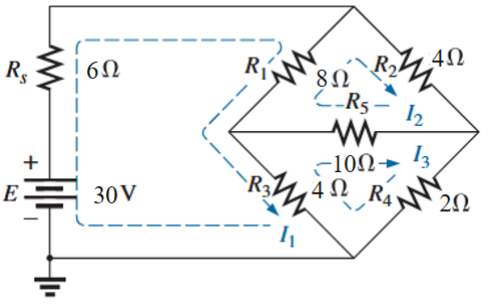
3. Calcule as tensões nos nós V1 e V2 [4]:



$$\begin{bmatrix} 0,250 & -0,167 \\ -0,167 & 0,208 \end{bmatrix} \begin{Bmatrix} V_1 \\ V_2 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 3 \end{Bmatrix}$$

Resposta: V1 = 81,17 V
V2 = 79,59 V

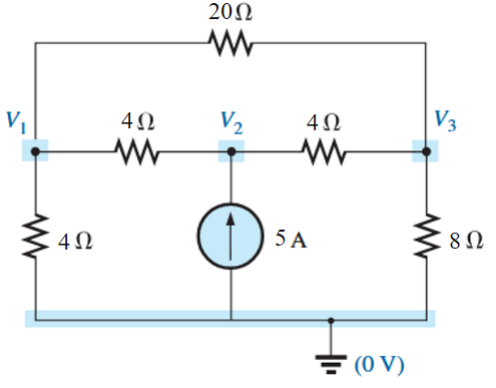
4. Calcule as correntes I1, I2 e I3 nas seguintes malhas [4]:



$$\begin{bmatrix} 18 & -8 & -4 \\ -8 & 22 & -10 \\ -4 & -10 & 16 \end{bmatrix} \begin{Bmatrix} I_1 \\ I_2 \\ I_3 \end{Bmatrix} = \begin{Bmatrix} 30 \\ 0 \\ 0 \end{Bmatrix}$$

Resposta: I1 = -1,80 A
I2 = -5,77 A
I3 = -4,06 A

5. Calcule as tensões nos nós V1, V2 e V3 [4]:



$$\begin{bmatrix} 0,625 & -0,25 & -0,05 \\ 0,5 & -0,25 & -0,25 \\ 0,425 & -0,25 & -0,05 \end{bmatrix} \begin{Bmatrix} V_1 \\ V_2 \\ V_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 5 \\ 0 \end{Bmatrix}$$

Resposta: V1 = 0 V
V2 = 5 V
V3 = 25 V

5.5 Referências básicas

[1] LAY, D. C.; LAY, S. R.; MCDONALD, J. **Álgebra Linear e suas Aplicações**, 5ª ed. LTC, 2018. ISBN 9788521634959.

[2] MONTICELLI, A. J. **Fluxo de carga em redes de energia elétrica**. São Paulo: Ed. Edgard Blucher Ltda, 1983. ISBN 978-85-268-0945-1.

[3] ERLABI. Mechtutor.com. **Gauss Elimination Method Tutorial - Part 1: Basic Procedure | Numerical Methods with Python**. Youtube, 20 nov. 2019. Disponível em: <https://www.youtube.com/watch?v=ZDxONtacA_4>. Acesso em: 24 ago. 2025.

[4] CASTAÑÓN. CCTMexico. **Python: Factorización de matrices LU (Paso a paso, básico)**. Youtube, 22 dez. 2016. Disponível em: <https://www.youtube.com/watch?v=FpVeXhAQg9w>. Acesso em: 24 ago. 2025.

[5] CASTAÑÓN. CCTMexico. **Gauss-Seidel | Código |Python | Métodos Numéricos | Solución de Sistemas de Ecuaciones Lineales**. Youtube, 25 out. 2017. Disponível em: <https://www.youtube.com/watch?v=SuguUYdE67A>. Acesso em: 24 ago. 2025.

5.6 Referências complementares – Aplicações

[1] SANTOS, W. V. D.; PRIMO, A. S. Aplicação da álgebra linear na engenharia elétrica: análise de circuitos elétricos em corrente contínua. In: **Ciências exatas e tecnológicas**, vol. 4, n. 1, p. 53-76, 2015. ISSN 1980-1777.

[2] CAMARGO, C. V.; DALPOSSO, G. H. Cálculo numérico computacional: aplicação na microeletrônica. In: **Revista Jr de Iniciação Científica em Ciências Exatas e Engenharia**, vol. 1, n. 6, p. 7-15, 2017. ISSN 2236-0093.

[3] MORADI, M. H.; FOROUTAN, V. B.; ABEDINI, M. Power flow analysis in islanded Micro-Grids via modeling different operational modes of DGs: A review and a new approach. In: **Renewable and Sustainable Energy Reviews**, vol. 69, p. 248-262, 2017. doi: 10.1016/j.rser.2016.11.156.

[4] BOYLESTAD, R. L. **Introdução à Análise de Circuitos**, 12ª ed. Pearson, 2011. ISBN 978-85-6457-420-5.

Sistemas Não-Lineares

Alguns problemas da engenharia podem ser equacionados como sistemas algébricos lineares. Isso pressupõe aceitar condições de idealidade em muitos casos. Exemplos de não linearidades são a dependência da temperatura na condição de operação de um semicondutor ou a saturação de meios magnéticos.

Ao contrário dos sistemas lineares, cujas equações representam retas, os gráficos associados às equações não-lineares são curvas. A solução é a interseção das curvas, conforme ilustrado na Figura 6.1.

O método numérico para resolução de sistemas não lineares aqui apresentado é o Newton-Raphson. O método Gauss-Siedel, apresentado no capítulo anterior, também pode ser utilizado.

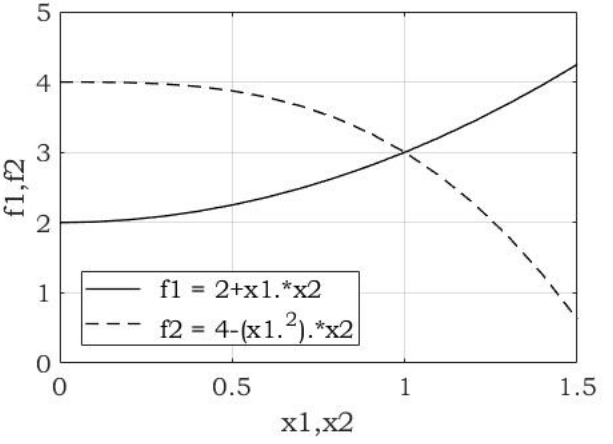
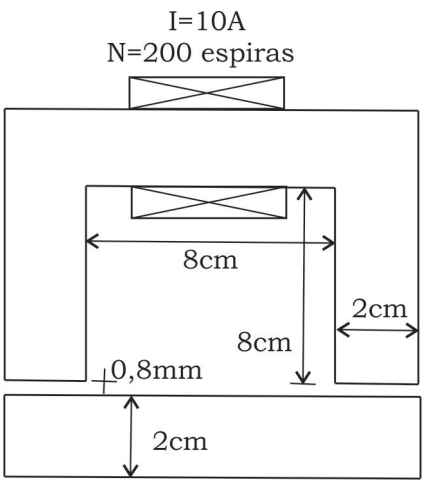


Figura 6.1: Ilustração de um sistema não-linear.

O problema



No circuito magnético ilustrado, a indução (N.I) pode não produzir um aumento proporcional de campo magnético devido a característica de saturação do ferro (curva de histerese) [1].

Assim, o circuito magnético é dito um sistema não linear.

O cálculo do campo magnético no ferro (H_f) e no entreferro (H_e) pode ser equacionado da seguinte forma:

$$\begin{cases} f_1(H_e, H_f) = 2H_e l_e + H_f l_f - NI \\ f_1(H_e, H_f) = -\mu_0 H_e + 1,8(1 - e^{-(H_f/40)}) \end{cases}$$

onde l_f = 0,40 m (comprimento médio no ferro), l_e = 0,80 mm (comprimento médio no entreferro), N = 200 (número de espiras), I = 10 A (corrente na bobina), $\mu_0 = 4\pi 10^{-7}$ H/m (permeabilidade magnética do ar).

Aqui, percebe-se que não é possível isolar uma incógnita e resolver o problema de maneira analítica direta. O termo $e^{(H_f/40)}$ não permite. Esse termo também impõe a não linearidade do problema. Portanto, faz-se necessário o emprego de métodos numéricos.

6.1 Método Newton-Raphson

O capítulo sobre raízes apresenta Newton-Raphson para uma única função:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{6.1}$$

Generalizando-o para n equações, tem-se:

$$\frac{\partial f_{k,i}}{\partial x_1} x_{1,i+1} + \cdots + \frac{\partial f_{k,i}}{\partial x_2} x_{2,i+1} = -f_{k,i} + x_{1,i} \frac{\partial f_{k,i}}{\partial x_1} + \cdots + x_{n,i} \frac{\partial f_{k,i}}{\partial x_n}$$

onde k representa a função em estudo, n é a variável e i é o valor atual. A sua notação matricial pode ser:

$$\{x_{i+1}\} = \{x_i\} - [J]^{-1} \{f\} \tag{6.2}$$

sendo que $[J]$ é a matriz Jacobiana das derivadas parciais, então:

$$[J] = \begin{bmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} & & \frac{\partial f_{1,i}}{\partial x_n} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} & \cdots & \frac{\partial f_{2,i}}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_{n,i}}{\partial x_1} & \frac{\partial f_{n,i}}{\partial x_2} & & \frac{\partial f_{n,i}}{\partial x_n} \end{bmatrix} \tag{6.3}$$

Newton-Raphson é um método iterativo. Portanto, faz-se necessária a definição de uma aproximação inicial (possível solução) e de critérios de parada (número máximo de iterações e erro admissível).

O método de Newton-Raphson para sistemas não-lineares pode ser representado por uma função através de um pseudocógico:

```
1    DO WHILE TRUE
2        f1 = 2 * x[0] * le + x[1] * lf - N * l
3        f2 = -mi * x[0] + 1.8 * (1 - EXP(-x[1] / 40))
4        F = vetor [f1, f2]
5
6        J = matriz [[2 * le, lf],
7                    [-mi, 1.8 * (1 / 40) * EXP(-x[1] / 40)]]
```

```
8
9           dx = RESOLVER_SISTEMA_LINEAR(J, F)
10          x = x - dx
11          iter = iter + 1
12
13          ea = MAX(ABS(dx / x))
14
15          IF iter >= maxit OR ea <= es THEN
16              EXIT
17          END IF
18
19          PRINT "Com ", iter, " iterações:"
20          PRINT "x = ", x
21      END DO
22
23      RETURN x
24  END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente:

```
1  from math import pi, exp
2  import numpy as np
3
4  lf = 0.4  # comprimento médio no ferro (m)
5  le = 0.0008  # comprimento médio no entreferro (m)
6  N = 200  # Número de espiras
7  I = 10  # Corrente elétrica(A)
8  mi = 4 * pi * 10 ** (-7)
9  x = np.array([[1250000], [70]])  # aproximações iniciais
10
11 iter = 0
12 maxit = 50
13 es = 0.001
14 while True:
15     f1 = (2 * x[0] * le + x[1] * lf - N * I)
16     f2 = (-mi * x[0] + 1.8 * (1 -exp(-(x[1]) / 40)))
17     F = np.array([f1, f2])  # Matriz de funções
18     J = np.array([[2 * le, lf],
19                   [-mi, 1.8 * (1 / 40) * exp(-(x[1]) / 40)]])  #
       Matriz Jacobiana
20     dx = np.linalg.lstsq(J, F, rcond=None)[0]  # rcond=None:
       padrão mais recente da função
21     x = x - dx
22     iter += 1
23     ea = max(abs(np.divide(dx, x)))
24     if iter >= maxit or ea <= es:
25         break
26     print(f'Com {iter} iterações: \n x = \n {x}')
```

Considerando a condição inicial de H_e = 1250 kA/m e H_f = 70 A/m, por Newton-Raphson, o resultado obtido foi H_e = 1230410,8397 A/m e H_f = 78,3566 A/m em 3 iterações, com critério de parada de quatro casas decimais. A Tabela 6.1. apresenta as três iterações.

Tabela 6.1. Resolução pelo Método de Newton-Raphson.

Iteração	He (A/m)	Hf (A/m)	Erro
0	1250k	70	-
1	1230,6067k	77,5729	0,0976
2	1230,4126k	78,3493	0,0099
3	1230,4108k	78,3566	0,0000

6.2 Exercícios propostos

1. O problema do circuito com diodo apresentado no capítulo 4 pode ser equacionado por um sistema não linear da seguinte forma:

$$\begin{cases} f_1(I_d, V_d) = I_{CR}\left(e^{\frac{V_d}{n\frac{kT}{q}}} - 1\right) - I_d \\ \\ f_2(I_d, V_d) = \left(\frac{V - V_d}{R}\right) - I_d \end{cases}$$

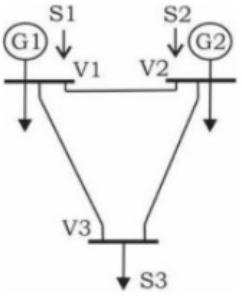
onde I_d é a corrente no diodo e V_d é a tensão no diodo. Admitindo um diodo 1N4001 com I_{CR} = 31,9824 nA, n = 2, k = 1,3806 x 10^{-23} J/K, T = 300K (26,85 °C) e q = 1,6022 x 10^{19} C, encontre I_d e V_d.

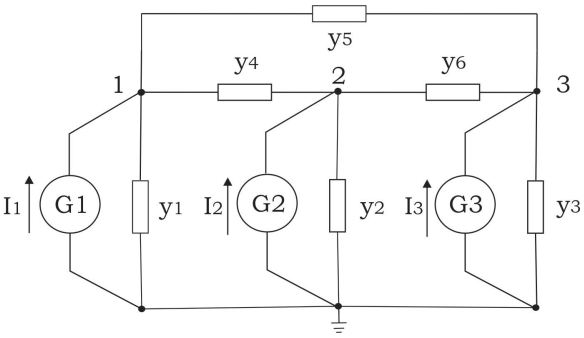
Resposta: Partindo de V_d = 1 V e I_d = 2 A, com 5 iterações obtém-se o critério de parada de quatro casas decimais. V_d = 0,9355 V e I_d = 2,3065 A.

Observação: esta questão apresenta problema de condicionamento. Isso é devido as diferenças nos valores entre as grandezas, particularmente 10^{-9} e 10^{-23}. Rever capítulo 5.

2. O problema de fluxo de potência num sistema elétrico de três barras é apresentado na forma de diagrama unifilar e seu circuito equivalente. As admitâncias y possuem valor de $1 + j1$ pu.

   Admitindo que as correntes nas barras sejam iguais a 1 pu, determine as tensões.





$$\{I\} = [Y]\{V\}$$

$$\begin{Bmatrix} I_1 \\ I_2 \\ I_3 \end{Bmatrix} = \begin{bmatrix} (y_1 + y_4 + y_5) & -y_4 & -y_5 \\ -y_4 & (y_2 + y_4 + y_6) & -y_6 \\ -y_5 & -y_6 & (y_3 + y_5 + y_6) \end{bmatrix} \begin{Bmatrix} V_1 \\ V_2 \\ V_3 \end{Bmatrix}$$

   Resposta: pelo método de Gauss-Siedel, obtém-se $\{V\}$ = 0,5000 + j0,5000 pu na 13° iteração com critério de parada de quatro casas decimais.

3. Determine as raízes das equações múltiplas a seguir utilizando o método de Newton-Raphson. Utilize a aproximação inicial para os cálculos de x = 3 e y = 2 [3].

$$\begin{cases} u(x,y) = x^2 + xy - 22 = 0 \\ \\ v(x,y) = y + 2xy^2 - 48 = 0 \end{cases}$$

   Resposta: x = 3,6 e y = 2,5

4. Determine as raízes das equações múltiplas a seguir utilizando o método de Newton-Raphson. Utilize a aproximação inicial para os cálculos de x = 0 e y = 1 [3].

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

Resposta: x = 0,5886 e y = 1,9114

5. Determine as raízes das equações múltiplas a seguir utilizando o método de Newton-Raphson. Utilize a aproximação inicial para os cálculos de x = 1 e y = 0 [3].

$$\begin{cases} y = -x^2 + x + 0,5 \\ \\ y + 3xy = x^2 \end{cases}$$

Resposta: x = 1,1685 e y = 0,3031

6.3 Referências básicas

[1] BASTOS, J. P. A.; IDA, N. **Electromagnetics and Calculation of Fields**, 2ª ed. Springer Verlag, 2012. ISBN 1461268605.

[2] SALGADO, R. S. **Introdução aos Sistemas de Energia Elétrica**. Apostila Graduação em Engenharia Elétrica, UFSC, 2009.

6.4 Referências complementares – Aplicações

[1] MORADI, M. H.; FOROUTAN, V. B.; ABEDINI, M. Power flow analysis in islanded Micro-Grids via modeling different operational modes of DGs: A review

and a new approach. In: **Renewable and Sustainable Energy Reviews**, vol. 69, p. 248-262, 2017. doi: 10.1016/j.rser.2016.11.156.

[2] TIWARI, S. N.; SINGH, L. P. Six-phase (multiphase) power transmission systems: A generalized investigation of the load flow problem. In: **Electric Power Systems Research**, vol. 5, n. 4, p. 285-297, 2017. doi: 10.1016/0378-7796(82)90024-4.

[3] CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**, 7ª ed. McGraw Hill Brasil, 2016. ISBN 9788580555684.

Ajuste de Curvas

O engenheiro realiza na bancada de um laboratório diversas medidas de tensão e corrente. Sabe-se que muitas vezes pode ser custoso obter muitos pontos ou até mesmo inviável medir determinados valores, por exemplo, quando há restrições da infraestrutura.

Ajuste de curvas é uma família de técnicas matemáticas que buscam encontrar padrões de comportamento num conjunto de dados. O objetivo é estudar o fenômeno físico na forma de uma tendência.

Havendo a possibilidade de se traçar uma reta ou curva, pode-se fazer previsões que, quando internas ao grupo de dados original, podem ser chamadas de interpolações. Quando externas, dá-se o nome de extrapolações. Em ambos os casos, existe a necessidade de controles estatísticos para aferir a qualidade dessas inferências.

O objetivo é buscar uma reta ou curva – um polinômio – que melhor se ajusta aos dados disponíveis. Conhecida a equação do polinômio, pode-se determinar valores dentro ou fora do intervalo conhecido.

Quatro famílias de métodos são aqui apresentadas:

- Ajuste de curvas por regressão: quando se tem uma grande quantidade de dados e eles são pouco precisos e/ou confiáveis;
- Ajuste de curvas por interpolação: quando se tem uma pequena quantidade de dados e eles são precisos e confiáveis;
- Ajuste de curvas por interpolação por partes: quando se tem uma grande quantidade de dados e eles são precisos e confiáveis;
- Ajuste de curvas com funções senoidais: para representar funções infinitas e periódicas dos processos físicos na forma de funções trigonométricas senos e cossenos.

Por fim, este capítulo apresenta uma discussão sobre séries de Fourier, importante ferramenta para análise de problemas da engenharia elétrica no domínio do tempo e da frequência.

7.1 Ajuste de curvas por regressão

Dado um grande conjunto de dados imprecisos, deseja-se extrair uma reta ou curva de tendência. Seja por uma regressão linear (reta) ou não-linear (curva), em ambos os casos existe um problema de minimização de erros, conforme ilustrado na figura 7.1.

As técnicas aqui apresentadas são os mínimos quadrados discretos (regressão linear) e mínimos quadrados contínuos (regressão polinomial).

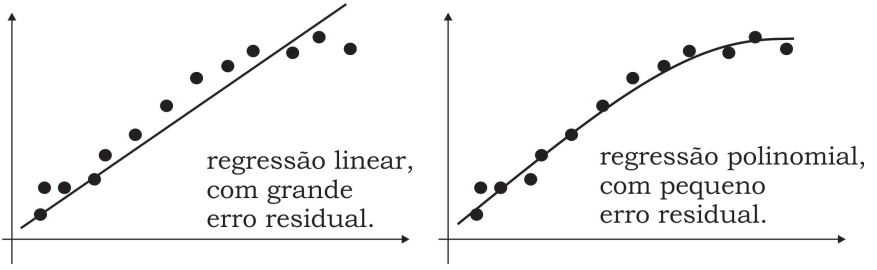


Figura 7.1: Ajuste de curvas por Regressão.

O problema

O engenheiro foi à bancada do laboratório de medidas elétricas e obteve dados de tensão e corrente. Ele esqueceu de medir a corrente para 60V. Como transformar os dados pontuais numa curva, ou seja, uma tendência de comportamento do circuito elétrico estudado? É possível realizar essa transformação através de métodos de regressão linear, e a resolução deste problema é apresentada dentros das próximas seções.

V (V)	0	10	20	30	40	50	70	80	90
I (A)	0	10	19	31	39	52	65	69	70

7.1.1 Método Mínimos Quadrados Discreto – Regressão Linear

O termo linear remete a uma reta. A equação de uma reta pode ser dada por:

$$y = a_0 + a_1 x \tag{7.1}$$

onde (x, y) é o conjunto de dados, a_0 e a_1 são os coeficientes que representam respectivamente a intersecção com o eixo y e a inclinação.

O método de mínimos quadrados busca minimizar o erro quadrático entre os dados (x, y) e os pontos de uma reta, representado por:

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_1 - a_0 - a_1 x_i)^2 \tag{7.2}$$

onde S_r é chamado de soma dos quadrados dos resíduos.

De forma direta, tem-se:

$$a_1 = \frac{n\sum(x_i y_i) - \sum(x_i)\sum(y_i)}{n\sum(x_i)^2 - \left(\sum(x_i)\right)^2} \quad \text{e} \quad a_0 = \bar{y} - a_1 \bar{x} \tag{7.3}$$

onde $\bar{x}$ e $\bar{y}$ são as médias de x e y. Para um ajuste perfeito entre a reta e o conjunto de dados, S_r deve ser 0.

Outra métrica significativa é soma total dos quadrados dos resíduos entre os pontos dados e a média (S_t):

$$S_t = \sum (y_i - \bar{y})^2. \tag{7.4}$$

A diferença $Sr - S_t$ indica a melhora ou a redução de erro em função da descrição dos dados. De forma a regularizarmos a escala, tem-se:

$$r^2 = \frac{S_t - S_r}{S_t}, \tag{7.5}$$

onde r^2 é chamado de coeficiente de determinação e r o coeficiente de correlação. Sendo o ajuste perfeito, $S_r = 0$ e $r^2 = 1$. Para $S_r = S_t$ e $r^2 = 0$, o modelo não representa melhora.

Uma forma de calcular r de forma computacionalmente mais eficiente é:

$$r^2 = \frac{n\sum(x_i y_i) - \sum(x_i)\sum(y_i)}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}. \tag{7.6}$$

Esse cálculo de regressão linear pode ser representado por uma função através de um pseudocógico:

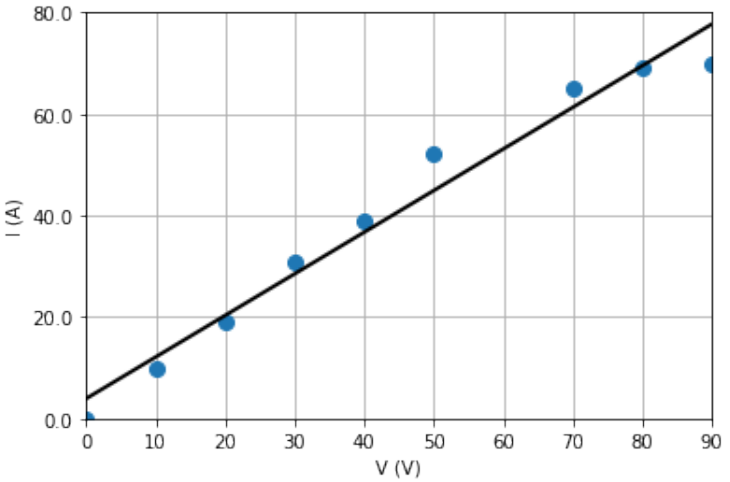
```
FUNCTION Regressão_Linear(x, y)
    n = tamanho de x

    IF tamanho de y      n THEN
        PRINT "x e y devem ter o mesmo tamanho"
    END IF

    sx = SOMA(x)
    sy = SOMA(y)
    sx2 = SOMA(x * x)
    sxy = SOMA(x * y)
    sy2 = SOMA(y * y)

    a[0] = (n * sxy - sx * sy) / (n * sx2 - sx  )
    a[1] = sy / n - a[0] * sx / n

    r2 = ((n * sxy - sx * sy) / (RAIZ(n * sx2 - sx  ) * RAIZ(n *
    sy2 - sy  )))
    r = RAIZ(r2)

    RETURN a, r
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

    x = np.transpose([0, 10, 20, 30, 40, 50, 70, 80, 90])
    y = np.transpose([0, 10, 19, 31, 39, 52, 65, 69, 70])
```

```
    n = len(x)  # número de amostras
    if len(y) != n:
        print('x e y devem ter o mesmo tamanho')
    sx = sum(x)
    sy = sum(y)  # soma
    sx2 = sum(x * x)
    sxy = sum(x * y)
    sy2 = sum(y * y)
    a = np.zeros(2)
    a[0] = (n * sxy - sx * sy) / (n * sx2 - sx ** 2)
    a[1] = sy / n - a[0] * sx / n
    r2 = ((n * sxy - sx * sy) / sqrt(n * sx2 - sx ** 2) / sqrt(n *
     sy2 - sy ** 2)) ** 2
    r = sqrt(r2)
    print(f'a = {a}')

    # gráfico
    xp = np.linspace(min(x), max(x), 2)
    yp = a[0]*xp+a[1]
    fig, ax = plt.subplots()
    ax.set_xlim(min(x), max(x))
    ax.set_ylim(min(y), max(y)+10)
    eixo_y = np.linspace(min(y), max(y)+10, 5)
    ax.set_yticks(eixo_y)
    for c in eixo_y:
        c = str(c)
    ax.set_yticklabels(eixo_y)
    ax.plot(x, y, 'o', 'k', markersize=8)
    ax.plot(xp, yp, 'k-', linewidth=2)
    ax.grid(True)
    ax.set_xlabel('V (V)')
    ax.set_ylabel('I (A)')
    V = 60
    I = a[0] * V + a[1]
    plt.show()
```

Para o problema apresentado, obteve-se:



$sx = 390$
$sy = 355$
$sx2 = 24900$
$sxy = 21940$
$sy2 = 19533$
levando à
$a_0 = 0,8196$
$a_1 = 3,9292$
com $r^2 = 0,9717$

Isso significa que o modelo adotado tem correlação de 98,58 %, ou seja, a reta representa a tendência dos dados com erro menor que 2 %. Para 60 V, I = 53,1042 A.

Apesar do bom resultado numérico, é possível perceber que existe uma saturação (não linearidade) do valor da corrente no final da escala da tensão. Portanto, o modelo adotado de uma reta talvez não seja a melhor maneira para representar este comportamento. Para uma melhor representação, podem ser utilizado métodos de regressão polinomial.

7.1.2 Método Mínimos Quadrados Contínuo – Regressão Polinomial

O termo não linear remete a uma curva. A aproximação para um conjunto de dados $\{(x_i, y_i)\}_{i=1}^{m}$ usando um polinômio de grau n pode ser:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n. \tag{7.7}$$

O grau n deve ser no máximo $n \leq m - 1$. Isto em função dos graus de liberdade: número de equações deve ser maior que o número de variáveis.

O método de mínimos quadrados contínuo busca minimizar o erro entre os dados (x, y) e os pontos da curva, representado por:

$$S_r = \sum_{i=1}^{m} e_i^2 = \sum_{i=1}^{m} (y_i - P_n(x))^2 \tag{7.8}$$

onde S_r é chamado de soma dos quadrados dos resíduos. Trata-se da generalização do método discreto que utiliza polinômio de primeiro grau. Os requisitos para se obter a solução são:

$$\frac{\partial S_r}{\partial a_j} = 0; j = 0, 1, 2, \ldots, n \, . \tag{7.9}$$

As $(n + 1)$ equações para a determinação dos coeficientes são:

$$\begin{bmatrix} \sum_{i=1}^{m} x_i^0 & \sum_{i=1}^{m} x_i^1 & \cdots & \sum_{i=1}^{m} x_i^n \\ \sum_{i=1}^{m} x_i^1 & \sum_{i=1}^{m} x_i^2 & \cdots & \sum_{i=1}^{m} x_i^{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{m} x_i^n & \sum_{i=1}^{m} x_i^{n+1} & \cdots & \sum_{i=1}^{m} x_i^{2n} \end{bmatrix} . \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i x_i^0 \\ \sum_{i=1}^{m} y_i x_i^1 \\ \vdots \\ \sum_{i=1}^{m} y_i x_i^n \end{bmatrix} . \tag{7.10}$$

As técnicas de resolução de sistemas lineares também podem ser aplicadas. Aqui também são válidas as eq. (7.4) e (7.5) para a determinação do coeficiente de correlação (r).

O método de regressão polinomial pelo método mínimos quadrados contínuo pode ser representado por uma função através de um pseudocógico:

```
1  FUNCTION Regressão_Polinomial(x, y)
2      n = size of x
3
4      sx = SUM(x)
5      sy = SUM(y)
6      mx = sx / n
7      my = sy / n
8      sx2 = SUM( x  )
9      sx3 = SUM( x  )
10     sx4 = SUM( x  )
```

```
    sxy = SUM(x * y)
    sx2y = SUM(x   * y)

    A = matrix [[n, sx, sx2],
                [sx, sx2, sx3],
                [sx2, sx3, sx4]]

    b = vector [sy, sxy, sx2y]

    c = SOLVE_LINEAR_SYSTEM(A, b)

    yy = SUM((y - my)  )

    yyy = 0
    FOR i = 0 TO n-1 DO
        yyy = yyy + (y[i] - c[0] - c[1] * x[i] - c[2] * x[i]  )
    END FOR

    r2 = (yy - yyy) / yy
    r = SQRT(r2)

    RETURN c, r
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

Fig, ax = plt.subplots()
ax.grid(True)
x = np.transpose([0, 10, 20, 30, 40, 50, 70, 80, 90])
y = np.transpose([0, 10, 19, 31, 39, 52, 65, 69, 70])
n = len(x)  # número de amostras
sx = sum(x)
sy = sum(y)  # soma
mx = sx/n
my = sy/n
sx2 = sum(x * x)
sx3 = sum(x * x * x)
sx4 = sum(x * x * x * x)
sxy = sum(x * y)
sx2y = sum(x * x * y)
x = [0, 10, 20, 30, 40, 50, 70, 80, 90]
y = [0, 10, 19, 31, 39, 52, 65, 69, 70]
# [A]{c} = {b}
```

```
A = np.array([[n, sx, sx2], [sx, sx2, sx3], [sx2, sx3, sx4]])
b = np.transpose([sy, sxy, sx2y])
c = np.linalg.lstsq(A, b, rcond=None)[0]  # resolução de sistemas
yy = sum(np.power((y-my), 2))             # rcond=None: versão
    mais atualizada
yyy = 0
for i in range(0, n):
    yyy = yyy + (y[i] - c[0] - c[1]*x[i] - c[2] * x[i] ** 2) ** 2

r2 = (yy - yyy) / yy
r = sqrt(r2)
V = 30
I = c[0] + c[1] * V + c[2] * np.power(V, 2)
# gráfico
xx = np.linspace(min(x), max(x), 100)
yy = c[0] + c[1] * xx + c[2] * np.power(xx, 2)
ax.plot(xx, yy, label='pol grau 2')
ax.plot(x, y, 'o', label="(x,y)")
ax.set_xlabel('V (V)')
ax.set_ylabel('I (A)')

plt.tight_layout()
plt.legend()
plt.show()
```

Considerando o problema apresentado, as equações lineares simultâneas para a obtenção de um polinômio de segunda ordem são:

$$\begin{bmatrix} 10 & 450 & 28500 \\ 450 & 28500 & 2025000 \\ 28500 & 2025000 & 153330000 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 417 \\ 25740 \\ 1790400 \end{bmatrix}.$$

Resolvendo o sistema por Gauss, chega-se a

$$P_2(x) = -2,0388 + 1,2782x - 0,0050x^2$$

com r = 99,70%. Para 60 V, I = 56,5593 A. A linha contínua na Figura 7.2 é este resultado.

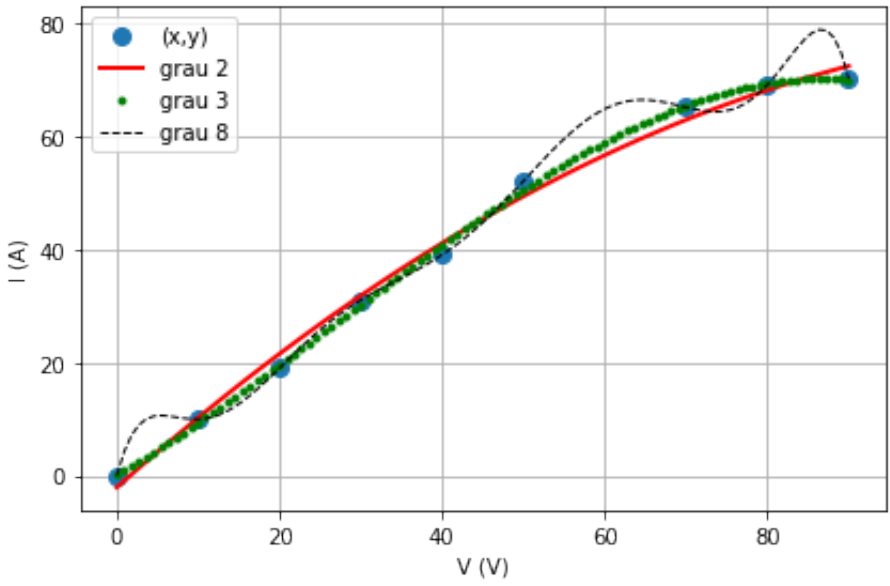


Figura 7.2: Ajuste de curvas com polyfit

A biblioteca Numpy possui funções para o ajuste de curvas utilizando o método dos mínimos quadrados. Primeiramente, `polyfit(x,y,n)` recebe como input os arrays de coordenadas de dados experimentais - `x` e `y` - e a ordem do polinômio a ser encaixado - `n`. Como saída, é retornado um array `c` com os coeficientes do polinômio. Em seguida, a função `polyval(c,xx)` recebe o array de coeficientes `c` e um array `xx`, que representa o intervalo de valores no eixo das abcissas inseridos no polinômio $P(x)$. Como saída, `polyval(c,xx)` retorna um array `yy` de ordenadas que, junto dos valores `xx`, compõe os pontos plotados do ajuste de curva. A seguir, é apresentado o mesmo algoritmo para polinômio do 2° grau utilizando as funções descritas.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.array([0, 10, 20, 30, 40, 50, 70, 80, 90])
5 y = np.array([0, 10, 19, 31, 39, 52, 65, 69, 70])
6 # POLYFIT
7 n = 2
```

```
8  c = np.polyfit(x,y,deg = n)  # ajusta por mínimos quadrados um
       polinômio de n grau
9  xx = np.linspace(x[0],x[len(x)-1],100)  # numero de pontos para
       polinômio de grau maior
10 yy = np.polyval(c,xx)
11 plt.grid(True)
12 plt.plot(x,y,'o',markersize = 8)
13 plt.plot(xx,yy,'k-',linewidth = 1)
14 plt.xlabel('V (V)')
15 plt.ylabel('I (A)')
16 plt.show()
```

A possibilidade do uso de funções internas da NumPy diminui significamente as linhas de código, dando maior praticidade ao estudo e à modelagem de problemas. O exemplo acima mostra que o mesmo código teve seu tamanho reduzido em mais de 50%.

Na Figura 7.2, a linha pontilhada ilustra a aplicação de `polyfit` para `n = 3`. Aqui, para 60 V, $I = 58{,}8775$ A.

Em razão de uma possível melhor representação, é natural o desejo de trabalhar com polinômios de grau superior. Entretanto, distorções podem surgir na forma da curva devido a problemas de condicionamento das matrizes (eq. 7.10). A curva tracejada é a representação para `n = 8`. Esses problemas serão tratados na seção 7.2, referente à interpolação.

7.2 Ajuste de curvas por interpolação

Trata-se de uma família de métodos que permite construir um novo conjunto de dados a partir de um conjunto de poucos dados precisos e confiáveis previamente conhecidos. A forma mais geral para um polinômio pode ser escrita como mostra a eq. (7.7). Para um conjunto de dados $\{(x_i, y_i)\}_{i=1}^{m}$ existe um, e somente um, polinômio (de grau $n = m - 1$) que passa por todos os pontos [1]. Por exemplo, para ligar dois pontos, tem-se uma única reta possível (interpolação linear). Para ligar três pontos não alinhados, tem-se uma parábola (interpolação de segundo grau ou quadrática).

Matricialmente, a eq. (7.7) pode ser organizada da seguinte maneira:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix} \tag{7.11}$$

Matrizes com coeficientes dessa forma são chamados de matrizes de Vandermonde. Elas são frequentemente matrizes mal condicionadas e, quando superiores a 3x3, seu número de condicionamento pode ser grande, causando erros significativos no cálculo dos coeficientes a_i. Existem algoritmos numericamente estáveis que exploram a estrutura da matriz de Vandermonde. Tais métodos consistem em primeiro construir um polinômio de interpolação de Newton ou Lagrange e depois convertê-lo para a forma canônica, conforme eq. (7.7). Quando se tem muitos dados, outra alternativa de evitar o mal condicionamento é realizar a interpolação por partes, também conhecida como $splines$, técnica apresentada ainda neste capítulo.

O problema

A Tabela 40 da NBR 5410:2004 apresenta os fatores de correção por temperaturas ambiente para linhas não-subterrâneas considerando dois tipos de compostos isolantes. Parte desta tabela é:

Temperatura (°C)	PVC	EPR ou XLPE
20	1,12	1,08
25	1,06	1,04
35	0,94	0,96

São poucos pontos, mas confiáveis. Nas próximas seções, será determinado, por meio de diferentes métodos, os fatores de correção para uma temperatura de 30°C.

7.2.1 Polinômios Interpoladores de Newton

Para um conjunto de dados $\{(x_i, y_i)\}_{i=1}^{m}$, o polinômio interpolador de Newton (de grau $n = m - 1$) pode ser expresso da seguinte maneira:

$$f_n(x) = a_1 + a_2(x - x_1) + \cdots + a_m(x - x_1)(x - x_2)\cdots(x - x_n), \quad (7.12)$$

sendo $f_n(x) = y_n$. As funções a correspondem a diferenças divididas finitas:

$$\begin{aligned} a_1 &= f(x_1) \\ a_2 &= f[x_2, x_1] \\ a_3 &= f[x_3, x_2, x_1] \\ &\vdots \\ a_m &= f[x_m, x_{m-1}, \cdots, x_2, x_1] \end{aligned} \quad (7.13)$$

e

$$f[x_2, x_1] = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \quad (7.14)$$

$$f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1} \quad \text{até}$$

$$f[x_m, x_{m-1}, \cdots, x_2, x_1] =$$

$$\frac{f[x_m, x_{m-1}, \cdots, x_2] - f[x_{m-1}, x_{m-2}, \cdots, x_2, x_1]}{x_m - x_1}$$

O método de polinômios interpoladores de Newton pode ser representado por uma função através de um pseudocógico:

```
FUNCTION Interpolação_Newton(x, y, xx)
    n = size of x
    b = ZERO_MATRIX(n, n)

    FOR i = 0 TO n-1 DO
        b[i, 0] = y[i]
    END FOR

    FOR j = 1 TO n-1 DO
        FOR i = 0 TO n-j DO
            b[i, j] = (b[i+1, j-1] - b[i, j-1]) / (x[i+j] - x[i])
        END FOR
    END FOR

    xt = 1
    yint = b[0, 0]

    FOR j = 0 TO n-2 DO
        xt = xt * (xx - x[j])
        yint = yint + b[0, j+1] * xt
    END FOR

    RETURN yint
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
import numpy as np
import matplotlib.pyplot as plt

x = ([20, 25, 35])
y = ([1.12, 1.06, 0.94])  # PVC
# y = np.transpose([1.08, 1.04, 0.96])  # EPR
xx = 30
n = len(x)
b = np.zeros((n, n))

# atribui as variaveis dependentes à primeira coluna de b
b[:n, 0] = y[:]
for j in range(1, n):
    for i in range(0, n - j + 1):
        if i + j != n:
            b[i, j] = (b[i + 1, j - 1] - b[i, j - 1]) / (x[i + j]
    - x[i])

# usa as diferenças divididas finitas para interpolar
```

```
xt = 1
yint = b[0, 0]
for j in range(0, n - 1):
    xt = xt*(xx - x[j])
    yint = yint+b[0, j + 1] * xt
print(f'yint = {yint}')  # Valor da interpolação para temp de 30
    graus

# Comparação com funções nativas!
P = np.polyfit(x, y, 2)  # ajusta por mínimos quadrados um polinô
    mio de n grau
s = np.polyval(P, xx)

# gráfico
xp = np.linspace(min(x), max(x), 100)
ss = np.polyval(P, xp)
plt.plot(x, y, 'bo')
plt.plot(xx, yint, 'ro')  # plotagem de yint
# plt.plot(xx, s,'ro') # usando polyfit
plt.plot(xp, ss, 'k-')
plt.grid(True)
plt.show()
```

Para o problema apresentado, considerando o material PVC, tem-se:

$$f_2(x) = a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2)$$

x_i	$y_i = f(x_i)$	Primeira ddf	Segunda ddf
20	$1,12$	$f[x_2, x_1] = -0,012$	$f[x_3, x_2, x_1] = 0$
25	$1,06$	$f[x_3, x_2] = -0,012$	
35	$0,94$		

$$f_2(x) = 1,12 + (-0,012)(x - 20) + 0(x - 20)(x - 25)$$

$$f_2(x) = 1,12 + (-0,012)(x - 20) \qquad f_2(30) = 1,12 + (-0,012)(30 - 20)$$

Como se tem três pares de dados, o polinômio interpolador de Newton tem grau 2. Para 30°C, o fator de correção considerando o PVC é igual a 1. Para EPR ou XLPE, o resultado também é 1.

7.2.2 Polinômios Interpoladores de Lagrange

Para um conjunto de dados $\{(x_i, y_i)\}_{i=1}^{m}$, de forma geral, o polinômio interpolador de Lagrange (de grau $n = m - 1$) pode ser expresso da seguinte maneira:

$$f_n(x) = \sum_{i=1}^{m} L_i(x) f(x_i) \quad \text{sendo} \quad L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{m} \frac{x - x_j}{x_i - x_j} \tag{7.15}$$

Onde $\prod$ indica "produto de".

O método de polinômios interpoladores de Lagrange pode ser representado por uma função através de um pseudocógico:

```
FUNCTION Interpolação_Lagrange(x, y, xx)
    n = size of x  // número de amostras

    IF size of y != n THEN
        PRINT "x e y devem ter o mesmo tamanho"
        RETURN
    END IF

    s = 0
    FOR i = 0 TO n-1 DO
        product = y[i]
        FOR j = 0 TO n-1 DO
            IF i != j THEN
                product = product * (xx - x[j]) / (x[i] - x[j])
            END IF
        END FOR
        s = s + product
    END FOR

    PRINT "s = ", s

    RETURN s
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
import matplotlib.pyplot as plt
import numpy as np

x = ([20, 25, 35])
y = ([1.12, 1.06, 0.94])  # PVC
# y = ([1.08, 1.04, 0.96])  # EPR
xx = 30
n = len(x)  # número de amostras
if len(y) != n:
    print('x e y devem ter o mesmo tamanho')
s = 0
for i in range(0, n):
    produto = y[i]
    for j in range(0, n):
        if i != j:
            produto = produto * (xx - x[j]) / (x[i] - x[j])
    s = s + produto
print(f's = {s}')

# gráfico
plt.grid(True)
plt.plot(x, y, 'bs')
plt.plot(xx, s, 'ro')
plt.show()
```

Para o problema apresentado, considerando o material PVC, tem-se:

$$f_2(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}f(x_1) + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}f(x_2) + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}f(x_3)$$

$$f_2(30) = \frac{(30-25)(30-35)}{(20-25)(20-35)}1,12 + \frac{(30-20)(30-35)}{(25-20)(25-35)}1,06 + \frac{(30-20)(30-25)}{(35-20)(35-25)}0,94$$

Como se tem três pares de dados, o polinômio interpolador de Lagrange tem grau 2. Para 30°C, o fator de correção considerando o PVC é igual a 1. Para EPR ou XLPE, o resultado também é 1.

Newton e Lagrange produzem resultados numericamente satisfatórios e similares. Diferentemente dos métodos de regressão, aqui não se obtém um polinômio, mas sim um único valor interpolado. Do ponto de vista de esforço computacional, Lagrange necessita de um maior número de operações matemáticas (soma, subtração, multiplicação e divisão) do que Newton.

7.3 Interpolação por splines ou por partes

A interpolação polinomial pode apresentar resultados indesejáveis quando feita para um número elevado de pontos usando-se polinômios de grau relativamente alto, conforme discutido na seção 7.1.2. Esses resultados indesejáveis são, em geral, grandes flutuações sobre o intervalo total de interpolação.

Uma alternativa é interpolar em grupos de poucos pontos, obtendo-se polinômios de grau menor, e impor condições para que a função de aproximação seja contínua e tenha derivadas contínuas até uma certa ordem. Essa técnica é conhecida como aproximação polinomial por partes ou $splines$.

A Figura 7.3 ilustra o processo de interpolação por partes. Os coeficientes da função $spline$ $(f(x_i))$ são calculados para cada intervalo de dados, por exemplo $[x_1 : x_3]$. O número de pontos (x_i) usados para cada função $spline$ definem a sua ordem.

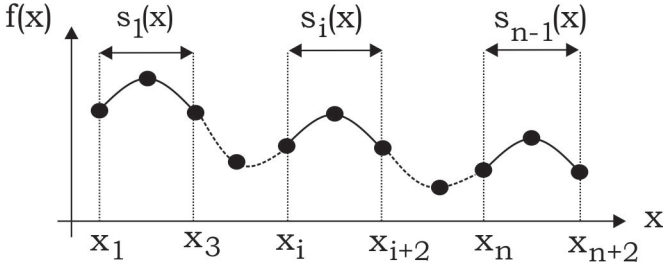


Figura 7.3: Interpolação por parte – splines

O problema

A tensão aplicada sobre um circuito eletrônico tem invertida sua polaridade de tempos em tempos. Entretanto, na inversão, ocorre um transitório durante o tempo de acomodação. Os valores medidos são:

T (s)	-2	-1	1	2	3	4
V (V)	-10	-10	10,5	9,75	10	10

Determine o polinômio que bem represente tal comportamento. Qual o valor da tensão no tempo igual a 1,5 segundos?

Interpolação por splines cúbica

Uma interpolação por $splines$ de primeira ordem implica obter retas entre cada par de pontos, passando obrigatoriamente por eles. Uma $spline$ de segunda ordem busca equações quadráticas entre cada par de pontos. Além de passar pelos pontos, tem-se um polinômio com a garantia da existência de sua derivada de primeira ordem. A $spline$ de terceira ordem ou cúbica produz um polinômio diferençável em primeira e segunda ordem.

Os resultados da interpolação por $splines$ cúbica são diferentes da interpolação cúbica, uma vez que por $splines$ existe a obrigatoriedade de se passar por todos os pontos.

Ressalta-se que $splines$ de ordem superior tendem a ter a mesma instabilidade de polinômios de ordem elevada: oscilações devido ao mau condicionamento. Assim, quando existe um grande conjunto de dados para ser interpolado, sugere-se sempre limitar o uso na ordem cúbica e aumentar o número de partes [1].

A $spline$ cúbica é um polinômio de terceiro grau com coeficientes de expansão b, c e d, representado por:

$$f_i + b_i h_i + c_i (h_i)^2 + d_i (h_i)^3 = f_{i+1} \quad \text{onde} \quad h_i = x_{i+1} - x_i \ . \tag{7.16}$$

Para n pontos dados $(i = 1, 2, 3. \ . \ . \ , n)$, existem $n-1$ intervalos e $4(n-1)$ coeficientes desconhecidos a calcular.

Assim, a primeira condição é atendida: o $spline$ deve passar por todos os pontos. Consequentemente, $4(n-1)$ condições são necessárias para calcular estes coeficientes. São elas:

1. A spline passa pelo primeiro e último ponto do intervalo, dando origem a $2(n-1)$ equações:

   (i) $s_i(x_i) = f_i \rightarrow a_i = f_i$

   (ii) $s_i(x_{i+1}) = f_i \rightarrow s_i(x_{i+1}) = a_i + b_i(h_i) + c_i(h_i)^2 + d_i(h_i)^3 = f_i$

2. A primeira derivada deve ser contínua em cada ponto do intervalo:

   (iii) $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \rightarrow b_i + 2c_i(h_i) + 3d_i(h_i)^2 = b_{i+1}$

3. A segunda derivada deve ser contínua em cada ponto do intervalo:

   (iv) $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \rightarrow 2c_i + 6d_i(h_i) = 2c_{i+1}$

Assim, tem-se $4n-6$ equações, mas são necessárias $4(n-1)$. Existem diversas maneiras para se obter as equações faltantes:

- $Natural\ end\ conditions$ – natural: assumir que a segunda derivada nos primeiro e último pontos é zero. A função é uma reta nos extremos;
- $Clamped\ end\ conditions$ – amarrada: assumir que a primeira derivada nos primeiro e último pontos é conhecida, no caso um valor imposto;
- "$Not\text{-}a\text{-}knot$" $end\ conditions$ – sem um nó: forçar a continuidade da terceira derivada no segundo e nos penúltimos pontos. Isto resulta em se ter os dois primeiros intervalos com a mesma equação, bem como os dois últimos.

Para as $natural\ end\ conditions$, tem-se:

$$\text{(v) } s_1''(x_1) = 0 \rightarrow c_1 \qquad \text{(vi) } s_{n-1}''(x_n) = 0 \rightarrow c_n$$

Por transformações algébricas, pode-se deixar as equações apenas com o coeficiente c:

$$d_i = \frac{c_{i+1} - c_i}{3h_i} \quad e \quad b_i = \frac{f_{i+1-f_i}}{h_i} - \frac{h_i}{3}(2c_i - c_{i+1}). \tag{7.17}$$

Assim, pode-se traduzir o equacionamento para uma forma matricial:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ h_i & 2(h_1+h_2) & h_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{Bmatrix}$$

$$= \begin{Bmatrix} 0 \\ 3(f[x_3, x_2] - f[x_2, x_1]) \\ \vdots \\ 3(f[x_n, x_{n-1}] - f[x_{n-1}, x_{n-2}]) \\ 0 \end{Bmatrix} \tag{7.18}$$

Lembrando que $f[x_i, x_j] = \frac{f_i - f_j}{x_i - x_j}$.

Uma vez calculados os coeficientes c_i, volta-se na eq. (7.17) e determina-se d_i e b_i. Isto posto, tem-se um polinômio para cada par de dados, conforme eq. (7.18).

Para o problema apresentado na seção 7.3, tem-se:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & 0 \\ 0 & 0 & 0 & h_4 & 2(h_4+h_5) & h_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{Bmatrix}$$

$$= \left\{ \begin{array}{c} 0 \\ 3(f[x_3, x_2] - f[x_2, x_1]) \\ 3(f[x_4, x_3] - f[x_3, x_2]) \\ 3(f[x_5, x_4] - f[x_4, x_3]) \\ 3(f[x_6, x_5] - f[x_5, x_4]) \\ 0 \end{array} \right\}$$

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 6 & 2 & 0 & 0 & 0 \\ 0 & 2 & 6 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \left\{ \begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{array} \right\} = \left\{ \begin{array}{c} 0 \\ 30,75 \\ -33 \\ 3 \\ -0,75 \\ 0 \end{array} \right\}$$

Truncando em quatro casas decimais, tem-se:

	1	2	3	4	5	6
[c]	0,0150	0,0580	-0,0653	-0,0106	0	-0,0004
[d]	0,0143	-0,0822	0,0182	0,0035	-0,0001	
[b]	-0,0294	10,2162	-0,7029	0,2571	0,0001	

Para a determinação da tensão no tempo igual a 1,5s, busca-se o polinômio do intervalo que contenha o referido tempo. No caso:

$$s_3(x) = f_3 + b_3(x - x_3) + c_3(x - x_3)^2 + d_3(x - x_3)^3$$

$$s_3(1,5) = 10,5 - 0,7029(1,5 - 1) - 0,0653(1,5 - 1)^2 + 0,0182(1,5 - 1)^3$$

$$s_3(1,5) = 10,1345\ V$$

A Figura 7.4 apresenta os resultados. Os círculos são os dados medidos. A curva tracejada foi obtida com o método aqui apresentado por *spline* cúbica. Para tal, faz-se necessária a interpolação de vários tempos de forma a desenhar a curva.

As curvas pontilhadas foram obtidas pela técnica de mínimos quadrados ($polyfit$) para terceiro e quinto graus. A curva de quinto grau aproxima-se mais do comportamento obtido com a $spline$ cúbica.

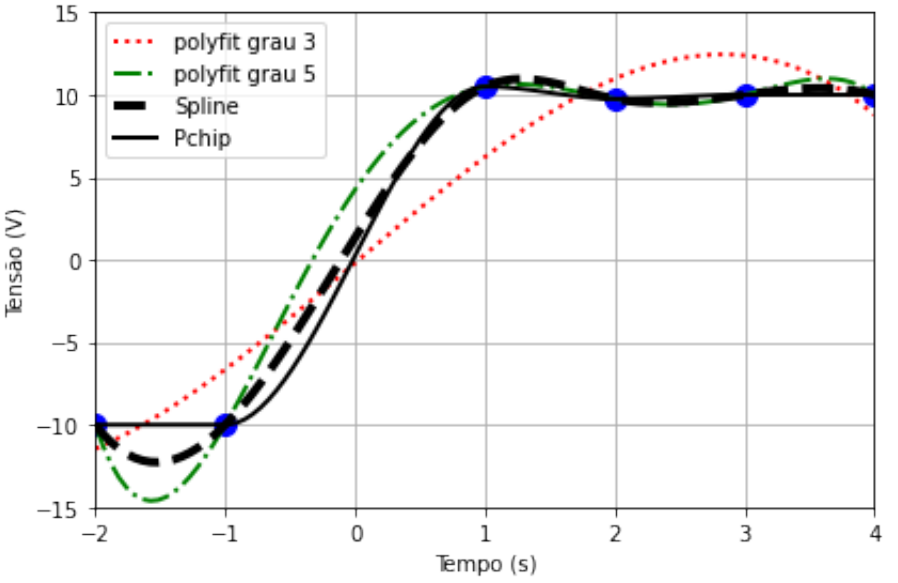


Figura 7.4: Gráfico obtido pela técnica de mínimos quadrados

A biblioteca SciPy possui as funções nativas `scipy.splrep()` e `scipy.splev()` - para interpolação por splines, que funcionam em conjunto com a mesma dinâmica da `np.polyfit()`: `splrep()` recebe os pontos e retorna os coeficientes do polinômio enquanto que `splev()` recebe os coeficientes e os valores de x, retornando os valores de $P(x)$. Além disso, a biblioteca também possui a função `scipy.PchipInterpolator()` - para interpolação $pchip$: com uma dinâmica diferente por ter uma abordagem de programação orientada a objeto. A função `scipy.PchipInterpolator()`, após receber os pontos, em vez de retornar coeficientes do polinômio, transforma a variável a qual é designada em uma função matemática polinomial que recebe os valores de x e retorna os valores de $P(x)$ correspondentes. A curva em linha contínua foi obtida com o método $pchip$ – interpolação cúbica hermitiana por partes - através dessa função da SciPy.

Em comparação com a *spline* cúbica, a interpolação cúbica hermitiana por partes não garante a continuidade na derivada de segunda ordem. Isso pode ser um problema em funções com mudanças abruptas.

Por outro lado, a implementação da *pchip* leva em consideração o conceito de que os valores interpolados não podem ultrapassar os pontos dados (*shape preserving*).

Voltando ao exercício proposto, percebe-se que a *pchip* leva vantagem sobre a spline no trecho antes da inversão da polaridade. Nesse trecho não existem valores menores que -10 V, conforme mostrado pela spline. Para o segundo trecho, após a inversão da tensão, a spline representa melhor o transitório durante o tempo de acomodação.

Com a melhoria do conhecimento sobre o problema em estudo, pode-se diagnosticar que a melhor estratégia a ser adotada seja a separação dos dados em grupos pequenos, podendo-se aplicar a técnica que melhor se adapta ao comportamento do problema.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import scipy.interpolate as sp
4
5  def Spline(x_interp, x_pontos, fx_pontos):
6      tck = sp.splrep(x_pontos, fx_pontos)
7      return sp.splev(x_interp, tck)
8
9  def Pchip(x_interp, x_pontos, fx_pontos):
10     pchip = sp.PchipInterpolator(x_pontos, fx_pontos)
11     return pchip(x_interp)
12
13 tempo = [-2, -1, 1, 2, 3, 4]
14 tensao = [-10, -10, 10.5, 9.75, 10, 10]
15 tt = 1.5
16
17 print(f'Interpolação Spline: y = {Spline(tt,tempo,tensao):.4f}')
18 print(f'Interpolação Pchip: y = {Pchip(tt,tempo,tensao):.4f}')
19
20 # Visualização
21 plt.grid(True)
22 plt.plot(tempo, tensao, 'o', markersize=10, color='b')
23 plt.xlabel('Tempo (s)')
```

```
plt.ylabel('Tensão (V)')

# Comparação com funções nativas
P = np.polyfit(tempo, tensao, 3)  # ajusta por mínimos quadrados um polinômio de grau 3
xp = np.linspace(min(tempo), max(tempo), 100)
ss = np.polyval(P, xp)
plt.plot(xp, ss, 'r:', linewidth=2, label='polyfit grau 3')

P = np.polyfit(tempo, tensao, 5)  # ajusta por mínimos quadrados um polinômio de grau 5
ss = np.polyval(P, xp)
plt.plot(xp,  ss, 'g-.', linewidth=2, label='polyfit grau 5')

yy = Spline(x_interp=xp, x_pontos=tempo, fx_pontos=tensao)
plt.plot(xp, yy, 'k--', linewidth=4, label='Spline')
yy = Pchip(x_interp=xp, x_pontos=tempo, fx_pontos=tensao)
plt.plot(xp, yy, 'k', linewidth=2, label='Pchip')
plt.xlim(-2, 4)
plt.ylim(-15, 15)
plt.tight_layout()

h = np.zeros(len(tempo)-1)
p = np.zeros(len(tempo)-1)
for i in range(0, (len(tempo) - 1)):
    h[i] = tempo[i + 1] - tempo[i]
    p[i] = (tensao[i + 1] - tensao[i]) / (tempo[i + 1] - tempo[i])

A = np.array([[1, 0, 0, 0, 0, 0],
              [h[0], 2*(h[0]+h[1]), h[1], 0, 0, 0],
              [0, h[1], 2*(h[1]+h[2]), h[2], 0, 0],
              [0, 0, h[2], 2*(h[2]+h[3]), h[3], 0],
              [0,  0, 0, h[3], 2*(h[3]+h[4]), h[4]],
              [0, 0, 0, 0, 0, 1]])

print(f'A =\n {A}')

pp = np.array([0, 3 * (p[1] - p[0]), 3 * (p[2] - p[1]), 3 * (p[3] - p[2]), 3 * (p[4] - p[3]), 0])
pp = np.transpose([pp])

c = np.linalg.lstsq(pp, A, rcond=None)[0][0]
b = np.zeros(len(tempo) - 1)
d = np.zeros(len(tempo) - 1)
print(f'pp =\n {pp}')
print(f'c =\n {c}')
for i in range(0, (len(tempo) - 1)):
    d[i] = (c[i + 1] - c[i]) / 3 * h[i]
print(f'd =\n {d}')
```

```
70 for i in range(0, (len(tempo) - 1)):
71     b[i] = ((tensao[i + 1] - tensao[i]) / h[i]) - (h[i] / 3) * (2
       * (c[i]) + c[i + 1])
72 print(f'b =\n {b}')
73 plt.legend()
74 plt.show()
```

7.4 Ajuste de curvas com funções senoidais

Em várias áreas da engenharia, principalmente em processamento de sinais, uma série temporal é uma coleção de observações feitas sequencialmente ao longo do tempo.

As funções matemáticas que melhor representam repetições temporais são o seno e cosseno. De forma geral, tem-se:

$$f(t) = a_0 + a_1 cos(\omega t + \theta) + b_1 sen(\omega t + \theta) + \cdots + a_m cos(m\omega t + \theta) + b_m sen(m\omega t + \theta) \tag{7.19}$$

onde a_0 determina a altura média acima ou abaixo da abscissa, os coeficientes de expansão $a_{1...m}$ e $b_{1...m}$ especificam as alturas máximas das oscilações, ω é a frequência com que os ciclos ocorrem e θ parametriza a extensão pela qual as curvas estão deslocadas horizontalmente.

Para N dados igualmente espaçados, os coeficientes $a_{1...m}$ e $b_{1...m}$ podem ser calculados por:

$$a_0 = \frac{\sum f(t)}{N} \tag{7.20}$$

$$\left.\begin{aligned} a_k &= \frac{2}{N}\sum f(t) cos(k\omega t) \\ b_k &= \frac{2}{N}\sum f(t) sen(k\omega t) \end{aligned}\right\} k = 1, 2, .., m.$$

Assim, se obedecido o critério de $N > 2m + 1$, tem-se um ajuste de dados semelhante à regressão – ajuste por mínimos quadrados de uma curva senoidal.

Se $2m+1$ é igual ao número de pontos dados (N), essa abordagem é chamada de série de Fourier contínua e será tratada mais à frente.

O problema

Mediu-se, sem muita precisão, a saída de uma fonte de tensão senoidal com frequência de 60Hz:

t(ms)	0	2,1	4,2	6,2	8,3	10,4	12,5	14,6	16,7
V	0	220	311	220	0	-220	-311	-220	0

Determine o polinômio que bem represente tal comportamento.

Aqui, $N = 9$ e $\sum f(t) = 0$. Assim, $A_0 = 0$.

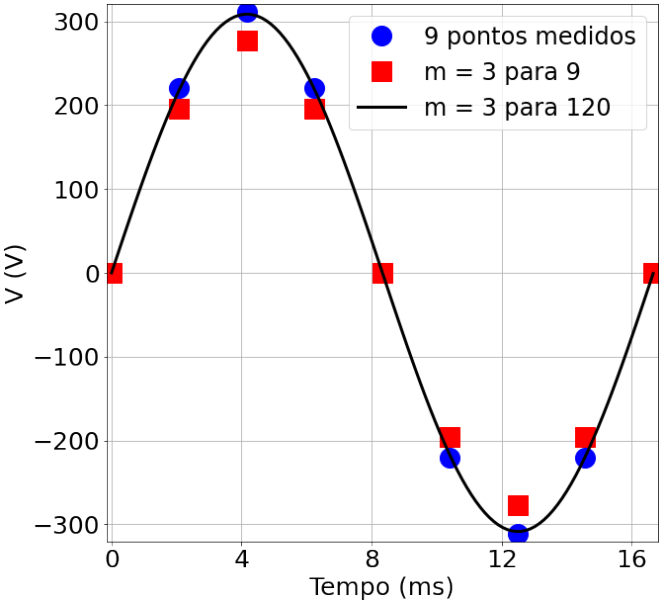
Para m = 1:

$$V(t) = 0 + 0,3475cos(\omega t) + 276,5440sen(\omega t)$$

Para m = 3:

$$\begin{aligned} V(t) = 0 + 0,3475cos(\omega t) + 276,5440sen(\omega t) - 0,0004cos(2\omega t) \\ -0,1738sen(2\omega t) - 0,0003cos(3\omega t) - 0,0739sen(3\omega t) \end{aligned}$$

A figura apresenta os 9 pontos medidos e os 9 pontos calculados com m igual a 3 coeficientes de expansão.



Com m = 1 ou 3, os valores calculados são próximos e apresentam um erro considerável quanto à amplitude do sinal. Lembrando que m = 3 é o limite imposto por $N > 2m + 1$.

Uma maneira de melhorar a precisão ao avaliar um mesmo período é aumentar o número de amostras do sinal. A figura apresenta o resultado para $m = 3$ e 120 amostras igualmente espaçadas de $1/(2 \times 60)$. O erro aqui é menor que 1%:

$$V(t) = 0 - 0,0062cos(\omega t) + 308,5316sen(\omega t) - 0,0062cos(2\omega t) \\ -0,0002sen(2\omega t) - 0,0062cos(3\omega t) - 0,0003sen(3\omega t)$$

O método de ajuste de curvas com funções senoidais pode ser representado por uma função através de um pseudocógico:

```
FUNCTION Ajuste_Curva_Senoidal(t, y, m)
    n = size of t  // número de amostras
    w = 2 * pi * 60  // frequência angular

    sy = soma de todos os elementos de y
    A0 = sy / n

    INICIALIZAR vetores A[0..m] e B[0..m] com zeros
    INICIALIZAR matrizes f[0..m-1][0..n-1] e ff[0..n-1][0..n-1]
    com zeros

    // Cálculo dos coeficientes A[k] e B[k]
    FOR k = 0 TO m DO
        soma_ycos = 0
        soma_ysin = 0
        FOR i = 0 TO n-1 DO
            soma_ycos = soma_ycos + (y[i] * cos(k * w * t[i]))
            soma_ysin = soma_ysin + (y[i] * sin(k * w * t[i]))
        END FOR
        A[k] = (2 / n) * soma_ycos
        B[k] = (2 / n) * soma_ysin
    END FOR

    // Construção das componentes f[k][i]
    FOR k = 0 TO m-1 DO
        FOR i = 0 TO n-1 DO
            f[k][i] = A[k] * cos(k * w * t[i]) + B[k] * sin(k * w
    * t[i])
        END FOR
    END FOR

    // Soma das componentes para formar a curva final
    FOR i = 0 TO n-1 DO
        ff[i] = A0 + soma de todos os f[:, i]
    END FOR

    // Eliminar A[0] e B[0], se necessário para uso posterior
    REMOVER A[0]
    REMOVER B[0]

    RETURN ff
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
from numpy import array, linspace, zeros
from math import pi, cos, sin
import matplotlib.pyplot as plt

def yfunc(t):
    return 0 + 311 * sin(w * t + 0)

T = 1 / 60
t = linspace(0, T, 9)
w = 2 * pi * 60
n = len(t)
m = 3
y = array([0, 220, 311, 220, 0, -220, -311, -220, 0])
sy = sum(y)
A0 = sy / n
A = zeros(m + 1)
B = zeros(m + 1)
f = zeros((m, n))
ff = zeros((n, n))
ycos = zeros(n)
ysin = zeros(n)
for k in range(0, m + 1):
    for i in range(0, n):
        ycos[i] = (y[i] * cos(k * w * t[i]))
        ysin[i] = (y[i] * sin(k * w * t[i]))
    A[k] = (2 / n) * sum(ycos)
    B[k] = (2 / n) * sum(ysin)
for k in range(0, m):
    for i in range(0, n):
        f[k, i] = A[k] * cos(k * w * t[i]) + B[k] * sin(k * w * t[
    i])
for i in range(0, n):
    ff[i] = A0 + sum(f[:, i])
A = A[1:]
B = B[1:]

Fig, ax = plt.subplots(figsize=(10, 10))

ax.plot(t, y, 'o', markersize=10, color='b', label='9 pontos
    medidos')
ffplot = []
for c in ff:
    ffplot.append(c[0])
ff = ffplot
ax.plot(t, ff, 'rs', label='m = 3 para 9')
T = 1 / 60
t = linspace(0, T, 120)
w = 2 * pi * 60
```

```
47 n = len(t)
48 m = 3
49
50 sy = sum(y)
51 A0 = sy / n
52 A = zeros(m + 1)
53 B = zeros(m + 1)
54 f = zeros((m, n))
55 ff = zeros((n, n))
56 ycos = zeros(n)
57 ysin = zeros(n)
58 for k in range(0, m + 1):
59     for i in range(0, n):
60         ycos[i] = (yfunc(t[i]) * cos(k * w * t[i]))
61         ysin[i] = (yfunc(t[i]) * sin(k * w * t[i]))
62     A[k] = (2 / n) * sum(ycos)
63     B[k] = (2 / n) * sum(ysin)
64 for k in range(0, m):
65     for i in range(0, n):
66         f[k, i] = A[k] * cos(k * w * t[i]) + B[k] * sin(k * w * t[
    i])
67 for i in range(0, n):
68     ff[i] = A0 + sum(f[:, i])
69 A = A[1:]
70 B = B[1:]
71 ffplot = []
72 for c in ff:
73     ffplot.append(c[0])
74 ff = ffplot
75 ax.plot(t, ff, 'k', label='m = 3 para 120')
76 ax.legend()
77 ax.grid(True)
78 ax.set_xlim(min(t) - 0.00015, max(t) + 0.00015)  # Ajuste de
    visualização do gráfico
79 ax.set_ylim(min(ff) - 12, max(ff) + 12)  # Ajuste de visualização
    do gráfico
80 ax.set_ylabel('V (V)')
81 ax.set_xlabel('Tempo (s)')
82 plt.show()
```

7.5 Série de Fourier

Série de Fourier é uma representação trigonométrica com senos e cossenos usada para representar funções periódicas complexas. Quando essas funções são infinitas, usa-se série de Fourier contínua. Quando essas funções não se repetem com periodicidade, ou se repetem, mas sem intervalos de tempos

regulares, utiliza-se a transformada de Fourier discreta (TFD) [3]. Exemplos de problemas sem periodicidade ou regularidade são os transitórios e descargas elétricas.

Série de Fourier Contínua

De forma muito semelhante ao ajuste de curvas com funções senoidais, partindo da eq. (7.20), a série de Fourier pode ser escrita como:

$$f(t) = a_0 + \sum_{k=1}^{\infty} [a_k cos(k\omega t) + b_k sen(k\omega t)] \tag{7.21}$$

onde a frequência angular do primeiro modo ($k = 1$) é chamada de frequência fundamental, e os seus múltiplos constantes ($k = 2, 3, \ldots$) são chamados de harmônicos. Os coeficientes da eq. (7.21) podem ser assim calculados:

$$a_0 = \frac{1}{T} \int_0^T f(t) dt$$

$$a_k = \frac{2}{T} \int_0^T f(t) cos(k\omega t) dt \quad \text{e} \quad b_k = \frac{2}{T} \int_0^T f(t) sen(k\omega t) dt \; . \tag{7.22}$$

Usando a fórmula de Euler ($e^{\pm j\theta} = cos(\theta) \pm j sen(\theta)$) , obtém-se uma forma mais compacta para expressar a série de Fourier:

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega t} \quad \text{onde} \quad c_k = \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{T}{2}} f(t) e^{-jk\omega t} dt \; . \tag{7.23}$$

Integral e Transformada de Fourier

A transição de uma função periódica para uma expressão não periódica pode ser feita permitindo-se que o período tenda ao infinito. Partindo-se da eq. (7.19):

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega \quad \text{e} \quad F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \tag{7.24}$$

$F(\omega)$ é chamada de transformada de Fourier de $f(t)$. E $f(t)$ é a transformada de Fourier inversa de $F(\omega)$. Este par de funções permite transformar para o domínio do tempo e da frequência um sinal não periódico.

Transformada de Fourier Discreta (TFD)

Em geral, na engenharia elétrica, as funções são representadas por um conjunto finito de valores discretos – conforme exemplo do problema proposto no início desta seção sobre a definição de um polinômio que descrevesse o comportamento da tensão senoidal medida. A TFD pode ser escrita como:

$$F_k = \sum_{i=0}^{N-1} f_j e^{jk\omega i} \quad \text{para} \quad k = 0 \quad \text{a} \quad N-1 \tag{7.25}$$

e a transformada inversa de Fourier como

$$f_i = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{jk\omega i} \quad \text{para} \quad i = 0 \quad \text{a} \quad N-1 \text{ onde } \omega = \frac{2\pi}{N}. \tag{7.26}$$

A Transformada de Fourier Discreta pode ser aplicada com uma função da biblioteca Numpy diretamente em Python:

```
import numpy as np

pi = np.pi
T = 1 / 60
t = np.linspace(0, T, 9)
y = np.array([0, 220, 311, 220, 0, -220, -311, -220, 0])
n = len(t)
print(np.fft.fft(y, n-1))  # fast Fourier - transform
# np.fft.ifft(y,n-1)  # Descomentar para usar inverse fast Fourier
    - transform
```

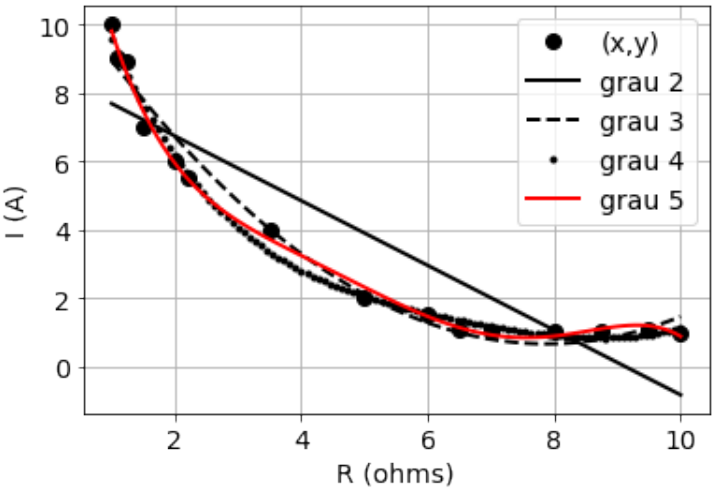
FFT é $fast - Fourier\ transform$, uma variação da eq. (7.22). IFFT é $inverse\ fast - Fourier\ transform$, variação de eq. (7.23).

7.6 Exercícios propostos

1. Uma medição de aterramento chegou aos valores apresentados na tabela. Entendendo que esses valores são pouco precisos, trace a curva que melhor represente o comportamento das medições. Discuta sobre a correção dos dados as curvas traçadas.

R (Ω)	I (A)
1	10
1,1	9
1,25	8,9
1,5	7
2	6
2,2	5,5
3,5	4
5	2
6	1,5
6,5	1,1
7	1,05
8	1
8,75	1
9,5	1,1
10	0,95

Resposta por regressão:
a curva contínua com n = 2;
curva tracejada com n = 3
e curva pontilhada com n = 4.



2. A medição de alguns pontos de uma curva de saturação magnética (B× H) num processo de desmagnetização do ferrite $MnZn_3C_{15}$é[4] :

H(A/m)	-50	-5	5	75
B(mT)	-300	-50	180	350

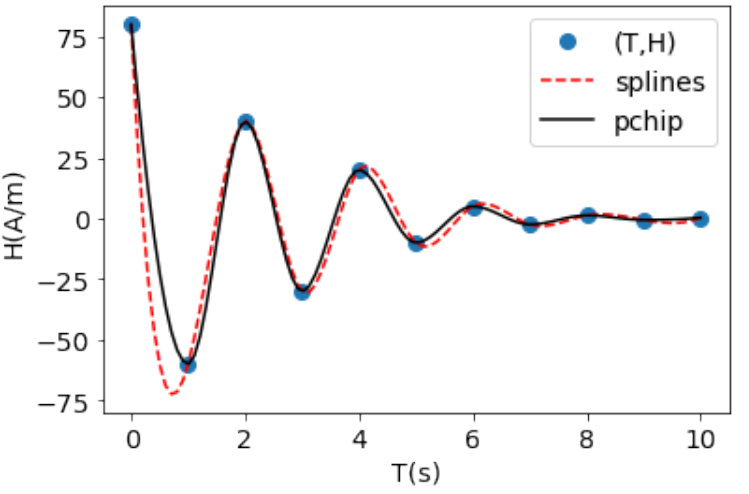
Determine o valor da indução magnética para H = 0 A/m.

Resposta:

por Newton, B = 62,8139 mT;
por Lagrange, B = 62,8139 mT;
por interpolação polinomial, n=2, B=68,5933 mT
n=3, B = 62,8139 mT.

3. Determine o polinômio que melhor represente o processo de desmagnetização do ferrite MnZn3c15, dado o seguinte comportamento:

H(A/m)	T(s)
80	0
-60	1
40	2
-30	3
20	4
-10	5
5	6
-2,5	7
1,25	8
-0,625	9
0,3125	10



Resposta: curva contínua foi obtida por spline e a curva tracejada por $pchip$. Percebe-se aqui que a $pchip$ não extrapola os valores dos dados medidos.

O comportamento deste problema remete às funções de Bessel. Elas são comumente encontradas nos problemas que envolvam o eletromagnetismo e o processamento de sinais, por exemplo.

4. Produza uma onda com frequência de 60Hz ($\omega = 377$ rad/s) e valor máximo de 311, onde a parte negativa da onda é rebatida para o quadrante positivo. A onda tem valor nulo enquanto o ângulo da onda original estiver menor que $\frac{\pi}{12}$ e entre π e $(\pi + \frac{\pi}{12})$.

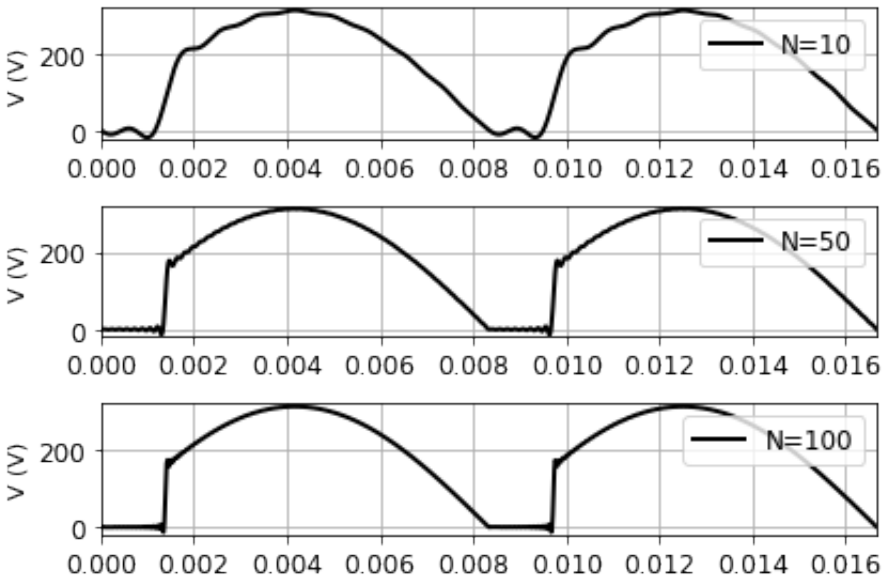
   Resposta:

$$a_0 = \frac{1}{T}\int_{n/(6x377)}^{n/(377)} 311 sen(377t)dt = 184,7260$$

$$a_n = \frac{311}{\pi}\left(\frac{1}{2n-1}\left(cos(\pi(2n-1)) - cos\left(\frac{\pi}{6}(2n-1)\right)\right)\right) + \left(\frac{1}{2n-1}\left(-cos(\pi(2n+1)) + cos\left(\frac{\pi}{6}(2n-1)\right)\right)\right)$$

$$b_n = \frac{311}{\pi}\left(\frac{1}{2n-1}\left(cos(\pi(2n-1)) - sen\left(\frac{\pi}{6}(2n-1)\right)\right)\right) + \left(\frac{1}{2n+1}\left(-sen(\pi(2n+1)) + sen\left(\frac{\pi}{6}(2n-1)\right)\right)\right)$$

$$f(t) = a_0 + \sum_{k=1}^{n}[a_k cos(k\omega t) + b_k sen(k\omega t)]$$



5. Na área de circuitos elétricos, é usual existir comportamentos de correntes como ondas quadradas. Quando determinada a série de Fourier de [6]:

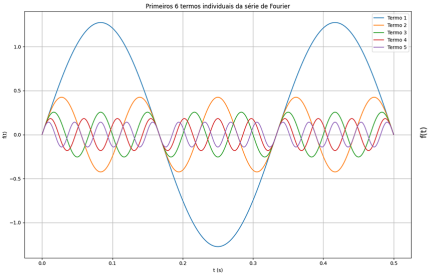
$$f(t) = \begin{cases} A_0, & 0 \leq t < \frac{T}{2} \\ -A_0, & \frac{T}{2} \leq t < T \end{cases}$$

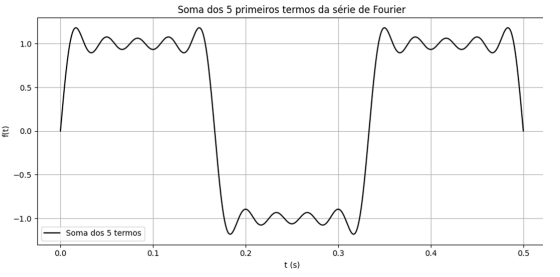
Obtém-se a seguinte série:

$$f(t) = \sum_{n=1}^{\infty} \frac{2A_0}{(2n-1)\pi} \sin\left(\frac{3\pi(2n-1)t}{T}\right)$$

Sendo A0 = 2 e T = 0,5 s, trace os primeiros cinco termos da série de Fourier individualmente e a soma desses termos.

Resposta:





7.7 Referências básicas

[1] REAMAT. **Cálculo Numérico – Um Livro Colaborativo**. UFRGS, 2019. Disponível em: <http://www.ufrgs.br/reamat/CalculoNumerico>. Acesso em: 24 ago. 2025.

[2] DE BOOR, C. **A Practical Guide to Splines**. Springer-Verlag, New York, 1978. ISBN 9780387953663.

[3] LATHI, B. P. **Sinais e Sistemas Lineares**, 2ª ed. São Paulo: Prentice-Hall do Brasil, 2007. ISBN 0195185334.

[4] ALMEIDA, L. A. de; DEEP, G. S.; LIMA, A. M. N.; NEFF, H. Um modelo diferencial para histerese magnética: representação algébrica recursiva. **Controle Automação**, vol. 14, n. 1, 2003.

7.8 Referências complementares – Aplicações

[1] KNOCKAERT, L. Multinomial Lagrange-Bernstein approximants. In: **IEEE Signal Processing Letters**, vol. 13, n. 6, p. 333-336, June 2006. doi: 10.1109/LSP.2006.871719.

[2] ZHANG, Y.; SHANG, C. Combining Newton interpolation and deep learning for image classification. In: **Electronics Letters**, vol. 51, n. 1, p. 40-42, 8 Jan. 2015. doi: 10.1049/el.2014.3223.

[3] SHAKIBA, M.; SUNDARAJAN, E.; ZAVVARI, A.; ISLAM, M. Cubic spline-based tag estimation method in RFID multi-tags identification process. In: **Canadian Journal of Electrical and Computer Engineering**, vol. 36, n. 1, p. 11-17, 2013. doi: 10.1109/CJECE.2013.6544467.

[4] HEIDEMAN, M.; JOHNSON, D.; BURRUS, C. Gauss and the history of the fast Fourier transform. In: **IEEE ASSP Magazine**, vol. 1, n. 4, p. 14-21, 1984. doi: 10.1109/MASSP.1984.1162257.

[5] BARBOSA, D.; MONARO, R. M.; et al. Filtragem adaptativa para a estimação da frequência em sistemas elétricos de potência. In: **SBA Controle Automação**, vol. 19, n. 2, p. 226-234, 2015.

[6] CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**, 7ª ed. McGraw Hill Brasil, 2016. ISBN 9788580555684.

Integração Numérica

No início de qualquer curso de graduação em engenharia, estuda-se cálculo diferencial e integral. A derivada representa a taxa de variação de uma variável dependente com relação a uma independente. A ideia é usar diferenças para quantificar um processo instantâneo.

A integração é o oposto. O objetivo pressupõe a soma de informações instantâneas para fornecer um resultado total ao longo de um intervalo, obtendo-se a área ou o volume de figuras geométricas.

A obtenção de derivadas e integrandos por métodos analíticos em problemas complexos usualmente pressupõe grande esforço, quando possível.

Matematicamente, a integração (definida) é representada por:

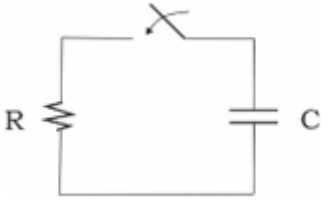
$$Int = \int_a^b f(x)dx, \tag{8.1}$$

que indica a integral da função f(x) em relação à variável independente x, avaliada entre os limites a e b [1].

Neste capítulo primeiro, apresentam-se as fórmulas de integração numérica por Newton-Cotes [2]. Elas são técnicas que utilizam um conjunto de pontos amostrados, sejam reais ou estimados por polinômios de interpolação. Em seguida, discutem-se fórmulas de integração numéricas que utilizam funções diretamente.

O problema

A resposta natural de um circuito RC é dada por:



$v(t) = V_0 e^{-t/\tau},\ t \geq 0$ onde V_0 é a tensão inicial no capacitor e τ a constante de tempo para o circuito, dado por $\tau = RC$. Sendo a corrente obtida por $i(t) = v(t)/R$, pode-se calcular a potência da seguinte forma:

$$p(t) = v(t) \cdot i(t) = \frac{1}{R}\left(V_0 \cdot e^{-t/\tau}\right)^2 = \frac{V_0^2}{R}\left(e^{-2t/\tau}\right) \text{ [W] [3].}$$

.

A energia envolvida pode ser calculada integrando

$$\omega = \int_0^t p(t)\,dt = \frac{V_0^2}{R}\int_0^t (e^{-2t/\tau})\,dt \text{ [J]}$$

Por fim, a potência média é dada por:

$$Pmed = \frac{\omega}{t} = \frac{V_0^2}{R} \cdot \frac{1}{t}\int_0^t (e^{-2t/\tau})\,dt \text{ [W].}$$

Analiticamente, considerando $R = 1\Omega$, $C = 0,001F$, $V_0 = 100V$ e o $\tau = 3RC$, obtém-se o comportamento:

$$\omega = \frac{V_0^2}{R}\int_0^{3\tau} (e^{-2t/\tau})\,dt = 0,0499 \text{ [J].}$$

$$Pmed = \frac{\omega}{3\tau} = 16,6254 \text{ [W].}$$

A energia (ω) é a área sob a curva $P \times t$. O cálculo dessa área é a sua integral. A Figura 8.1 a seguir ilustra o presente exercício.

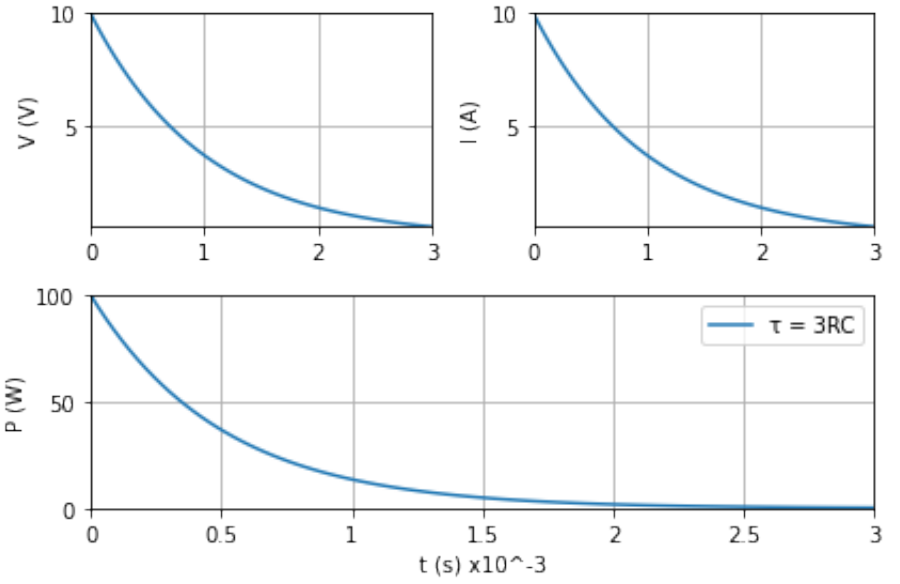


Figura 8.1: Comportamento de um circuito RC no tempo

Certamente, a resolução analítica é possível dada a simplicidade do circuito em análise. Em circuitos mais complexos, pode-se fazer uso de integrais numéricas.

8.1 Fórmulas de integração numérica por Newton-Cotes

As fórmulas de Newton-Cotes, ou regras de Newton-Cotes, são um grupo de fórmulas para integração numérica (também chamadas de quadratura) baseadas na avaliação do integrante em pontos igualmente espaçados. O mais comum é basear na estratégia de substituir uma função complicada ou um conjunto de dados tabulados por um polinômio, como por exemplo:

$$Int = \int_a^b f(x)\,dx \cong \int_a^b f_n(x)\,dx \text{ onde } f_n(x) = a_0 + a_1 x + \cdots + a_n^n\,, \quad (8.2)$$

e o integrar por partes, conforme ilustra a figura 8.2.

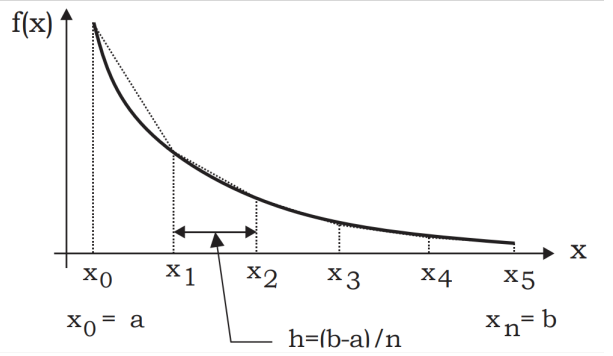


Figura 8.2: Integração por partes

Existem $n+1$ pontos igualmente espaçados $(x_0.x_1, x_2, \cdots, x_n)$. Assim, existem n segmentos de largura $(h = (b-a)/n)$. Se $a = x_0$ e $b = x_n$, a integral total pode ser representada por:

$$I = \int_{x_0}^{x_1} f(x)dx + \cdots + \int_{x_{n-1}}^{x_n} f(x)dx\,,$$

substituindo cada integral pela equação de área de um trapézio, tem-se:

$$Int = h\frac{(f(x_0) - f(x_1))}{2} + \cdots + h\frac{(f(x_{n-1}) - f(x_n))}{2}. \tag{8.3}$$

Agrupando-se os termos, pode-se obter a forma geral da Regra do Trapézio Múltipla:

$$Int = (b-a)\frac{f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)}{2n}. \tag{8.4}$$

O método de Integração Numérica por Newton-Cotes pode ser representado por uma função através de um pseudocógico:

```
FUNCTION Integracao_NewtonCotes(ff, a, b, n)
    h = (b - a) / n            // Tamanho de cada subintervalo
    x = a
    s = ff(x)                  // Inicializa soma com ff(a)

    FOR i = 0 TO n - 2 DO
        x = x + h
        s = s + 2 * ff(x)      // Soma os termos internos
    multiplicados por 2
    END FOR

    s = s + ff(b)              // Soma o valor da função no ponto
    final b

    I_aprox = (b - a) * s / (2 * n)  // Fórmula da regra do trapé
    zio

    RETURN I_aprox
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


def P(Vc, Ic):
    return np.multiply(Vc, Ic)


def ff(x):
    return (10 * np.exp(np.divide(-x, 0.001))) ** 2

R = 1
C = 0.001
Vo = 10
TAL = R * C
t = linspace(0, 3 * TAL, 7)
Vc = Vo * np.exp(np.divide(-t, (TAL)))
Ic = (Vo * np.exp(np.divide(-t, (TAL)))) / R
Pmed = Vo ** 2 / (6 * R) * (-np.exp(-6) + 1)
w = C * Vo ** 2 / 2 * (-np.exp(-6) + 1)
# trapézio
```

```
a = round(t[1])
b = t[len(t) - 1]
# limites
n = 150
# número de segmentos
x = a
h = (b - a) / n

Ianalitico = w
s = ff(x)
for i in range(0, n - 1):
    x = x + h
    s = s + 2 * ff(x)
s = s + ff(b)
P = (b - a) * s / (2 * n)
w = P
Pmed = w/(3 * TAL)
Erro = (Ianalitico - P)/Ianalitico
P = Vc * Ic
print('ff(x) = (10 * np.exp(np.divide(-x, 0.001))) ** 2 \n')
print(f'w = {w:.4f}')
print(f'Pmed = {Pmed:.4f}')
print(f'Erro = {Erro:.4f}')

# Visualização

Fig = plt.figure()
plt.style.use('fast')
ax1 = Fig.add_subplot(221)  # Posicionamento dos subplots
ax1.grid(True)  # grade
ax2 = Fig.add_subplot(222)  # Posicionamento dos subplots
ax2.grid(True)  # grade
ax3 = Fig.add_subplot(212)  # Posicionamento dos subplots
ax3.grid(True)  # grade

ax1.set_ylabel('V (V)')
ax2.set_ylabel('I (A)')
ax3.set_ylabel('P (W)')
ax3.set_xlabel('t (s) x10^-3')

ax1.locator_params(axis="y", nbins=2)
ax2.locator_params(axis="y", nbins=2)
ax3.locator_params(axis="y", nbins=2)

Tens = ax1.plot(t, Vc)
Corr = ax2.plot(t, Ic)
Pot = ax3.plot(t, P, label='T = 3RC')

ax1.set_xticklabels(['0', '1', '2', '3'])  # formatação eixo x, V
```

```
ax2.set_xticklabels(['0', '1', '2', '3'])  # formatação eixo x, I
ax3.set_xticklabels(['0', '0.5', '1', '1.5', '2', '2.5', '3'])  #
    formatação eixo x, P

ax1.set_ylim(min(Vc), max(Vc))  # limite do eixo y, V
ax2.set_ylim(min(Ic), max(Ic))  # limite do eixo y, I
ax3.set_ylim(0, 100)  # limite do eixo y, P
ax1.set_xlim(0, 0.003)  # limite do eixo x, V
ax2.set_xlim(0, 0.003)  # limite do eixo x, I
ax3.set_xlim(0, 0.003)  # limite do eixo x, P
plt.legend()
plt.tight_layout()
plt.show()
```

A biblioteca NumPy possui uma função interna para a integração numérica por Newton-Cotes, chamada `numpy.trapz()`.

A tabela a seguir apresenta os resultados do problema proposto pelo método de integração numérica do trapézio. Percebe-se que quanto maior o número de pontos, por tantas divisões (n), maior será a precisão da solução encontrada.

Para se obter mais pontos, pode-se optar pela técnica de ajustes de curvas mais adequada, conforme visto no capítulo anterior.

Como os valores analíticos são conhecidos, o erro pode ser calculado por $E = 100\%(real - estimado)/real$. Outra maneira pode ser a comparação direta entre as soluções obtidas para diferentes n.

n	$h[s]$	w [J]	$Pmed$ [W]	$Erro\,[\%]$
2	0,00150	0,0827	27,5513	65,72
10	0,00030	0,0514	17,1212	2,9822
50	0,00006	0,0499	16,6453	0,1200

Existem muitos outros métodos para realizar integração numérica. A tabela 8.1 apresenta os mais conhecidos. Para todos os citados, o passo é dado por $h = (b-a)/n$. O erro pode ser calculado utilizando a informação da derivada de ordem superior do ponto qualquer (ξ) entre a e b.

Tabela 8.1. Métodos para a Integração Numérica [1].

Fórmula	Truncamento
Trapézio (n = 2)	
$(b-a)\frac{f(x_0)+f(x_1)}{2}$	$-(1/12)h^3f"(\xi)$
1/3 Simpson (n = 3)	
$(b-a)\frac{f(x_0)+4f(x_1)+f(x_2)}{6}$	$-(1/90)h^5f^{(4)}(\xi)$
3/8 Simpson (n = 4)	
$(b-a)\frac{f(x_0)+3f(x_1)+3f(x_2)+f(x_3)}{8}$	$-(3/80)h^5f^{(4)}(\xi)$
Boole	
$(b-a)\frac{7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)7f(x_4)}{90}$	$-(8/945)h^7f^{(6)}(\xi)$

8.2 Fórmulas de integração de funções

As técnicas apresentadas na seção anterior utilizam um conjunto de pontos amostrados, sejam reais (medidos) ou estimados por polinômios de interpolação. Certamente adequados, tais métodos são custosos e podem acrescentar erros devidos à interpolação. Existindo uma função confiável, é possível gerar tantos valores quanto forem necessários para se alcançar a precisão desejada.

Existem muitos métodos de integração numérica a partir de funções como, por exemplo, a integração de Romberg, que utiliza a extrapolação de Richardson; e as fórmulas de Gauss-Legendre e Gauss-Hermite [1][2].

Aqui será apresentada a técnica chamada de Quadratura Adaptativa [2].

O diferencial desta proposta é a possibilidade de ajuste automático do intervalo de integração (h) de acordo com variações abruptas da função em estudo. Em regiões com gradiente pequeno, passos largos; já em regiões com gradiente grande, passos refinados. Esse procedimento confere maior precisão e velocidade ao cálculo [4].

Partindo da regra 1/3 de Simpson, tem-se uma primeira estimativa:

$$Int(h_1) = \frac{h}{6}[f(a) + 4f(c) + f(b)] \tag{8.5}$$

onde $h = b - a$ e $c = (a + b)/2$.

Uma estimativa mais refinada pode ser obtida pela metade do tamanho do passo, por exemplo com aplicação da regra 1/3 de Simpson com n=4:

$$Int(h_2) = \frac{h}{12}[f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)] \tag{8.6}$$

onde $d = (a + c)/2$ e $e = (c + b)/2$.

De forma a melhorar ainda mais a estimativa, com resultado equivalente à regra de Boole, pode-se utilizar:

$$Int = Int(h_2) + \frac{1}{15}[Int(h_1) + Int(h_2)]. \tag{8.7}$$

O método de Integração Adaptativa pode ser representado por uma função através de um pseudocógico:

```
1  // Função principal que realiza a integração adaptativa:
2  FUNCTION quadsetp(f, a, b, tol, fa, fc, fb)
3      h = b - a
4      c = (a + b) / 2
5      fd = f((a + c) / 2)
6      fe = f((c + b) / 2)
7
8      // Regra de Simpson com 3 pontos (q1)
9      q1 = (h / 6) * (fa + 4 * fc + fb)
10
11     // Regra de Simpson com 5 pontos (q2)     subdividida
12     q2 = (h / 12) * (fa + 4 * fd + 2 * fc + 4 * fe + fb)
```

```
13
14      // Verifica se a diferença entre q1 e q2 está dentro da tolerâ
        ncia
15      IF abs(q2 - q1) <= tol THEN
16          q = q2 + (q2 - q1) / 15      // Correção de erro estimado
17      ELSE
18          // Aplica recursivamente em [a, c] e [c, b]
19          qa = quadsetp(f, a, c, tol, fa, fd, fc)
20          qb = quadsetp(f, c, b, tol, fc, fe, fb)
21          q = qa + qb
22      END IF
23
24      RETURN q
25  END FUNCTION
```

Adaptado de [4]

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
1  import numpy as np
2
3  def feval(Nomefuncao, *argumentos):
4      return eval(Nomefuncao)(*argumentos)
5
6  def f(x):
7      return (10 * np.exp(np.divide(-x, 0.001))) ** 2
8
9  def quadsetp(f, a, b, tol, fa, fc, fb):
10     h = b - a
11     c = (a + b) / 2
12     fd = feval('f', (a + c) / 2)
13     fe = feval('f', (c + b) / 2)
14     q1 = h / 6 * (fa + 4 * fc + fb)
15     q2 = h / 12 * (fa + 4 * fd + 2 * fc + 4 * fe + fb)
16     if abs(q2 - q1) <= tol:
17         q = q2 + (q2 - q1)/15
18     else:
19         qa = quadsetp(f, a, c, tol, fa, fd, fc)
20         qb = quadsetp(f, c, b, tol, fc, fe, fb)
21         q = qa + qb
22     return q
23
24 a = 0
25 b = 0.003
26 c = (a+b) / 2
27 tol = 1 * np.exp(-6)
28 fa = feval('f', a)
29 fb = feval('f', b)
```

```
30 fc = feval('f', c)
31 w = quadsetp(f, a, b, tol, fa, fc, fb)
32 print(f'w = {w}')
```

Adaptado de [4]

O Scipy possui uma função interna para a integração numérica por funções, chamada `scipy.integrate.quad()`.

Para o circuito elétrico RC proposto, com tolerância de 10^{-6}, chega-se a $w = 0,0499$ J em apenas 3 ações recursivas.

Integrais múltiplas são bem comuns. Por exemplo, uma integral dupla estuda uma função bidimensional:

$$\int_{y_1}^{y_2} \left(\int_{x_1}^{x_2} f(x,y)\, dx \right) dy \tag{8.8}$$

Analiticamente, primeiro se integra a função em uma dimensão. Após, esse resultado é integrado na segunda dimensão.

Numericamente se faz o mesmo. A SciPy possui funções internas para integração dupla (`scipy.integrate.dblquad()`) e para integração tripla (`scipy.integrate.tplquad()`).

8.3 Exercícios propostos

1. Calcule numericamente o valor eficaz de $V(t) = 311 sen(377t + 0°)V$.

   Resposta: valor eficaz é o equivalente contínuo que gera o mesmo trabalho que o sinal alternado. Assim, a simples integral do sinal senoidal no intervalo de período reportará $Vmed = 0$. Uma forma de tratar isso é:

$$Vef = \sqrt{\frac{311^2}{2\pi} \int_0^{2\pi} sen^2(377t) dt} = \sqrt{\frac{311^2}{2\pi} Int}$$

Método	Pontos	Int	Vef
Trapésio	4	3,1416	219,9102
Quad	-	3,1416	219,9102

2. O engenheiro foi à bancada do laboratório e obteve os dados de tensão e corrente. Assim, ele pode calcular e conhecer a potência instantânea. Aplique um método de integração numérica e calcule a potência média no período entre [1 a 2] segundos.

t(s)	0	0,50	1,00	1,50	2,00	2,50	3,00
V(V)	10	6,10	3,70	2,25	1,42	0,78	0,50
I(a)	10	6,05	3,65	2,25	1,40	0,80	0,50
P(W)	100	36,90	13,50	5,06	1,99	0,62	0,25

Resposta:

$$P(t) = 0,5582t^6 - 6,6480t^5 + 34,2689t^4 - 100,5233t^3 + 181,5979t^2 - 195,7537t + 100$$

Por trapézio, com n = 90, Pmed = 5,9263.
Por trapézio, com n = 100, Pmed = 5,9262.
Por quadratura adaptativa, com 1 ação recursiva, Pmed = 5,9262.

3. Em um estudo hidrológico, as profundidades (em metros) de um rio foram medidas em intervalos regulares ao longo de uma seção transversal. A área dessa seção pode ser obtida por integração numérica da função profundidade $H(x)$ ao longo do eixo x, conforme [4]:

$$A_c = \int_0^x H(x)\, dx$$

Os dados coletados estão organizados na tabela a seguir:

x (m)	0	3	6	9	12	15	18	21	24
$H(x)$ (m)	0	2,2	3,1	2,9	2,5	2,8	2,1	1,5	0

Utilize um método de integração numérica (como a regra do trapézio ou quadratura adaptativa) para estimar a área da seção transversal do rio.

Resposta: 51.3 m²

4. A função erro, comumente usada em estatística e física, é definida pela integral abaixo [4]:

$$\text{erf}(a) = \frac{2}{\sqrt{\pi}} \int_0^a e^{-x^2}\, dx$$

Essa função não possui uma expressão analítica exata. Use a fórmula da **quadratura de Gauss com dois pontos** para calcular uma aproximação de $\text{erf}(0{,}8)$.

Considere que o valor correto de $\text{erf}(0{,}8)$ é aproximadamente $0{,}742101$.

Resolva utilizando métodos de integração numérica apropriados, como a regra do trapézio ou quadratura adaptativa.

5. Em um processo de transporte de fluido por tubulação, a massa de um poluente que passa por um ponto da tubulação ao longo do tempo pode ser calculada integrando-se o produto entre a vazão $Q(t)$ e a concentração $c(t)$ [4]:

$$M = \int_{t_1}^{t_2} Q(t)c(t)\, dt$$

Considere que a vazão e a concentração variam no tempo segundo as expressões:

$$Q(t) = 3 + 2\cos^2(0{,}5t), \quad c(t) = 2e^{-0{,}2t} + 3e^{0{,}25t}$$

Calcule a massa total transportada entre os instantes $t_1 = 3$ min e $t_2 = 10$ min.

A integração deve ser realizada utilizando métodos numéricos como a regra do trapézio composta ou a quadratura adaptativa, com uma precisão de até $0{,}1\%$.

Resposta: 473,9343 unidades

8.4 Referências básicas

[1] DAVIS, P. J.; RABINOWITZ, P. **Methods of Numerical Integration (Computer Science Applied Mathematics)**, 2ª ed. Academic Press, 2014. ISBN 9781483237961.

[2] NILSSON, J. W.; RIEDEL, S. A. **Circuitos Elétricos**, 10ª ed. Pearson, 2016. ISBN 9788543004785.

[3] GANDER, W.; GAUTSCHI, W. Adaptive Quadrature-Revisited. In: **BIT Numerical Mathematics**, vol. 40, n. 1, p. 84-101, 2000. doi: 10.1023/A:1022318402393.

[4] BISHOP, D. **feval similar to Matlab**. BYTES, 2004. Disponível em: <https://bytes.com/topic/python/answers/29829-feval-similar-matlab>. Acesso em: 24 ago. 2025.

8.5 Referências complementares - Aplicações

[1] SAVAGE, J. S.; PETERSON, A. F. Quadrature rules for numerical integration over triangles and tetrahedra. In: **IEEE Antennas and Propagation Magazine**, vol. 38, n. 3, p. 100-102, 1996. doi: 10.1109/74.511963.

[2] VOLSKIY, V.; et al. Numerical integration of Sommerfeld integrals based on singularity extraction techniques and double exponential-type quadrature formulas. In: **European Conference on Antennas and Propagation (EUCAP)**, p. 3215-3218, 2012. doi: 10.1109/EuCAP.2012.6205950.

[3] KARVONEN, T.; SÄRKKÄ, S. Classical quadrature rules via Gaussian processes. In: **International Workshop on Machine Learning for Signal Processing (MLSP)**, p. 1-6, 2012. doi: 10.1109/MLSP.2017.816819.

[4] CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**, 7ª ed. McGraw Hill Brasil, 2016. ISBN 9788580555684.

Derivação Numérica

A derivação é usual na engenharia. Na grande maioria dos casos, o entendimento de seu comportamento pressupõe a distinção da mudança de grandezas físicas no tempo e no espaço. Por exemplo, o vetor densidade de corrente elétrica (J) pode ser equacionado como a proporcionalidade da condutividade elétrica (σ) do meio condutor e a variação da tensão elétrica (dV/dt), desta forma [1]:

$$J = -\sigma \frac{dV}{dt}. \tag{9.1}$$

Matematicamente, a derivada representa a taxa de variação de uma variável dependente com relação a uma variável independente. Aqui será apresentada uma formulação de diferença finita centrada e noções de gradiente. A resolução de equações diferenciais ordinárias será apresentada em capítulo a parte.

O problema

A lei de Faraday determina que a tensão em um indutor (V) é dada por:

$$V_L = L\frac{di}{dt},$$

onde L é a indutância (H), i é a corrente (A) e t é o tempo (s) [1].

Determine a tensão em função do tempo a partir dos valores medidos da corrente para um $L = 0{,}1$H.

t(s)	0	0,10	0,20	0,30	0,40	0,50
i(A)	0	1,00	3,00	5,00	8,50	20,00

Resposta: VL(V) | 2,00 | 2,00 | 2,75 | 7,50 | 5,75 | 0,00

9.1 Fórmulas de derivação

As expansões de série de Taylor são comumente utilizadas como fórmulas de diferença finita. Aqui, é apresentada a fórmula de diferença dividida centrada de Taylor, entretanto, destaca-se que existem ainda fórmulas progressivas e regressivas ao ponto em estudo [2][3].

Assim, quanto mais termos da expansão, mais acurado será a solução.

Primeira Derivada — Erro

$$f'(x_i) = (f(x_{i+1}) - f(x_{i-1}))/2h \qquad 0(h^2)$$

$$f'(x_i) = (-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2}))/12h \qquad 0(h^4)$$

Segunda Derivada

$$f"(x_i) = (f(x_{i+1}) - 2f(x_i) + f(x_{i-1}))/h^2 \qquad 0(h^2)$$

$$f"(x_i) = (-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) \qquad 0(h^4)$$

$$-f(x_{i-2}))/12h$$

Terceira Derivada

$$f'''(x_i) = (f(x_{i+2}) - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2}))/2h^3 \qquad 0(h^2)$$

$$f'''(x_i) = (-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1})$$

$$-8f(x_{i-2}) + f(x_{i-3}))/8h^3 \qquad 0(h^4)$$

h é o passo de cálculo.

Quando o passo de cálculo é desigualmente espaçado, pode-se ajustar um

polinômio de Lagrange de segundo grau para três pontos adjacentes. A diferenciação do polinômio resulta em:

$$f'(x) = f(x_1)\frac{(2x - x_2 - x_3)}{(x_1 - x_2)(x_1 - x_3)} + f(x_2)\frac{(2x - x_1 - x_3)}{(x_2 - x_1)(x_2 - x_3)} + f(x_3)\frac{(2x - x_1 - x_2)}{(x_3 - x_1)(x_3 - x_2)} \tag{9.2}$$

As vantagens de se usar Lagrange ao invés da expansão de Taylor, muito embora mais custosa computacionalmente, é que se pode obter uma estimativa da derivada em qualquer lugar do intervalo determinado pelos três pontos, com precisão e para passos de cálculo iguais ou não.

Uma outra dificuldade de qualquer método de derivação numérica é a amplificação de erros nos dados. Em engenharia, particularmente em dados de medidas em laboratório, o erro na medição é comum.

A Figura 9.1 ilustra essa preocupação: (a) dados sem erros, (b) a derivação numérica resultante da curva (a), (c) dados ligeiramente modificados, e (d) a derivação resultante da curva (c) manifestando um aumento da variabilidade.

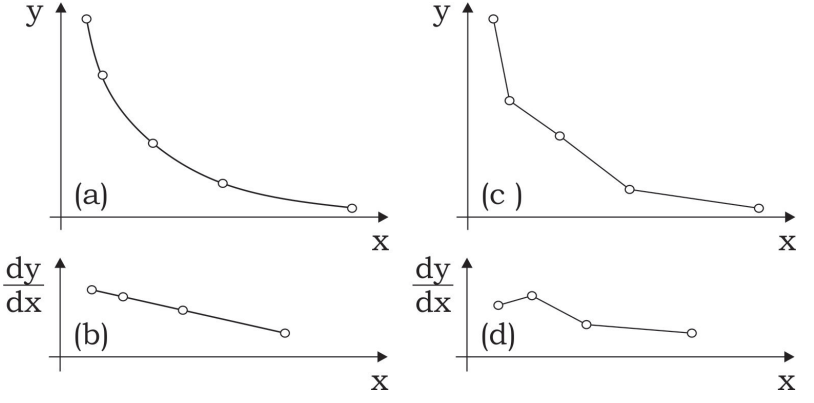


Figura 9.1: Preocupação com a precisão dos dados para derivação.

O método de Derivação Numérica pode ser aplicado diretamente em Python com auxílio da função "diff"do Numpy:

```
from numpy import divide, diff
import matplotlib.pyplot as plt

t = [0, 0.10, 0.20, 0.30, 0.40, 0.50]
I = [0, 1.00, 3.00, 5.00, 8.50, 20.00]
d = divide(diff(I), diff(t))
L = 0.1
V = L * d
tt = [0.05, 0.15, 0.25, .35, .45]

print(f'd = {d}\n')
print(f'V = {V}\n')

# Visualização
fig, ax = plt.subplots()
Corr = ax.plot(t, I, linewidth=1, marker='d', label='Corrente')
ax2 = ax.twinx()  # cria nova escala no lado oposto do gráfico
    eixo y
Tens = ax2.plot(tt, V, linewidth=1, marker='s', label='Tensão',
    linestyle='--', color='k')
lns = Tens + Corr
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=0)
ax.set_xlabel('Tempo (s)')
ax.set_ylabel('Corrente (A)')
ax2.set_ylabel('Tensão (V)')
ax2.yaxis.grid(False)
plt.tight_layout()
plt.show()
```

Para o problema do indutor apresentado e sendo a fórmula de derivação centrada, obtêm-se os resultados expostos na tabela. Aqui, acontece perda de informação. O método utilizado perde o início e o fim dos dados.

V	0		2		2		3,50		11,5	
I		1		3		5		8,50		20
T	0,05	0,10	0,15	0,20	0,25	0,30	0,35	0,40	0,45	0,50

Usando a equação (9.2), para t = 0,15s, I = 2 A.

A Figura 9.2 mostra o comportamento da corrente e da tensão no tempo.

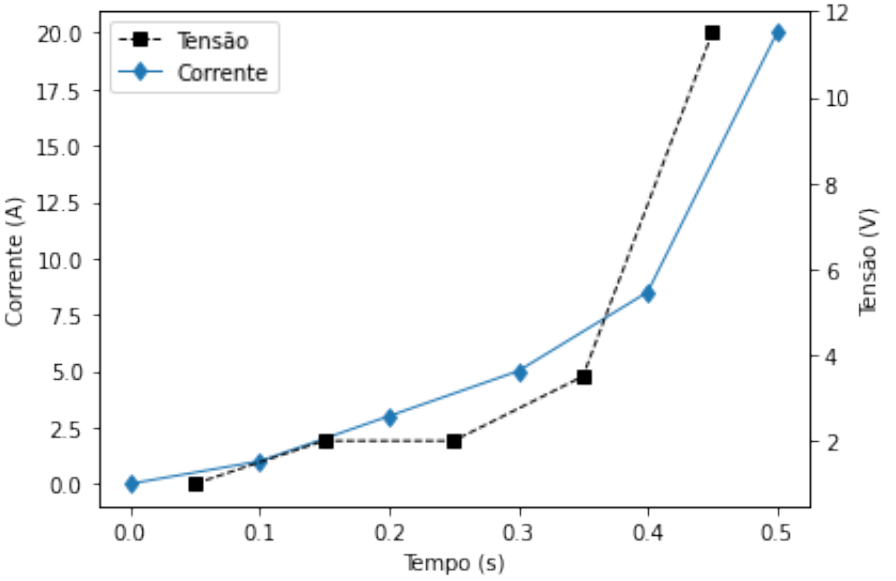


Figura 9.2: Gráfico do comportamento da tensão e da corrente

9.2 Gradiente

O gradiente é um vetor que indica o sentido e a direção na qual obtém- se o maior incremento de uma função. Assim, o módulo do vetor gradiente indica a taxa de variação do fenômeno em estudo.

O campo vetorial e o operador gradiente possuem diversas aplicações em engenharia. Por exemplo, a partir do gradiente do potencial elétrico determina-se o campo elétrico; e a partir do gradiente da energia potencial determina-se o campo de força associado [4].

A Figura 9.3 e o código a seguir ilustram o fenômeno elétrico existente entre duas cargas elétricas estáticas de valores contrários. Nesta plotagem foram utilizadas as funções `plt.quiver()`, para ilustrar os vetores, e `plt.contour()` para as superfícies equipotenciais do campo elétrico, ambas da Matplotlib. A função `plt.quiver(x, y, u, v)` recebe 4 arrays que representam coordenadas: os arrays `x` e `y` representam as coordenadas dos pontos iniciais das setas e, os

arrays `u` e `v` representam as coordenadas dos pontos finais de cada seta. Como saída, ocorre a plotagem das setas no gráfico. A função `plt.contour(x, y, z)` recebe arrays com as coordenadas nos três eixos(x, y, z) e plota as curvas de nível para a representação bidimensional da estrutura tridimensional descrita pelos arrays. Nesse caso, as curvas representam superfícies equipotenciais no entorno das cargas. Como argumento opcional nesta função, a variável $niveis$ entrega ao parâmetro `levels` quais níveis plotar.

```
import matplotlib.pyplot as plt
import numpy as np

# resolução da imagem
x_range = np.arange(-2, 2, 0.2)  # intervalo e quantidade de setas
y_range = np.arange(-2, 2, 0.2)  # intervalo e quantidade de setas
x_mesh, y_mesh = np.meshgrid(x_range, y_range)
# linhas de campo
E = np.multiply(x_mesh, np.exp(-(np.power(x_mesh, 2)) - np.power(
    y_mesh, 2)))
dy, dx = np.gradient(E)
plt.figure(figsize=(10, 10))  # Tamanho
plt.quiver(x_mesh, y_mesh, dx, dy, pivot="middle")  # Setas
plt.axis("scaled")  # Ajuste automático de escala
# superfícies equipotenciais geradas pelas duas cargas
xp = np.arange(-2, 2, 0.1)
yp = np.arange(-2, 2, 0.1)
x_mesh, y_mesh = np.meshgrid(xp, yp)
E = np.multiply(x_mesh, np.exp(-(np.power(x_mesh, 2)) - np.power(
    y_mesh, 2)))
d_o = 4
if (len(E) % 2) != 0:
    l = int((len(E) + 1) / 2)
else:
    l = int((len(E)) / 2)
niveis = np.linspace(E[l][l - d_o], -E[l][l - d_o], 6)  # niveis
    intersecções do plano
eqp = plt.contour(x_mesh, y_mesh, E, levels=niveis)  # linhas de
    contorno
plt.clabel(eqp)
# posicionamento das duas cargas
# carga elétrica negativa
carga_x1 = 0.70
carga_y1 = 0
# carga elétrica positiva
carga_x2 = -0.70
carga_y2 = 0
plt.plot(carga_x1, carga_y1, color='blue', marker='o', linestyle='
```

```
    dashed', markersize=7)
plt.plot(carga_x2, carga_y2, color='red', marker='o', linestyle='
    dashed', markersize=7)
plt.title('Campo Elétrico')
# plt.savefig("Campo_elétrico.png")
plt.show()
```

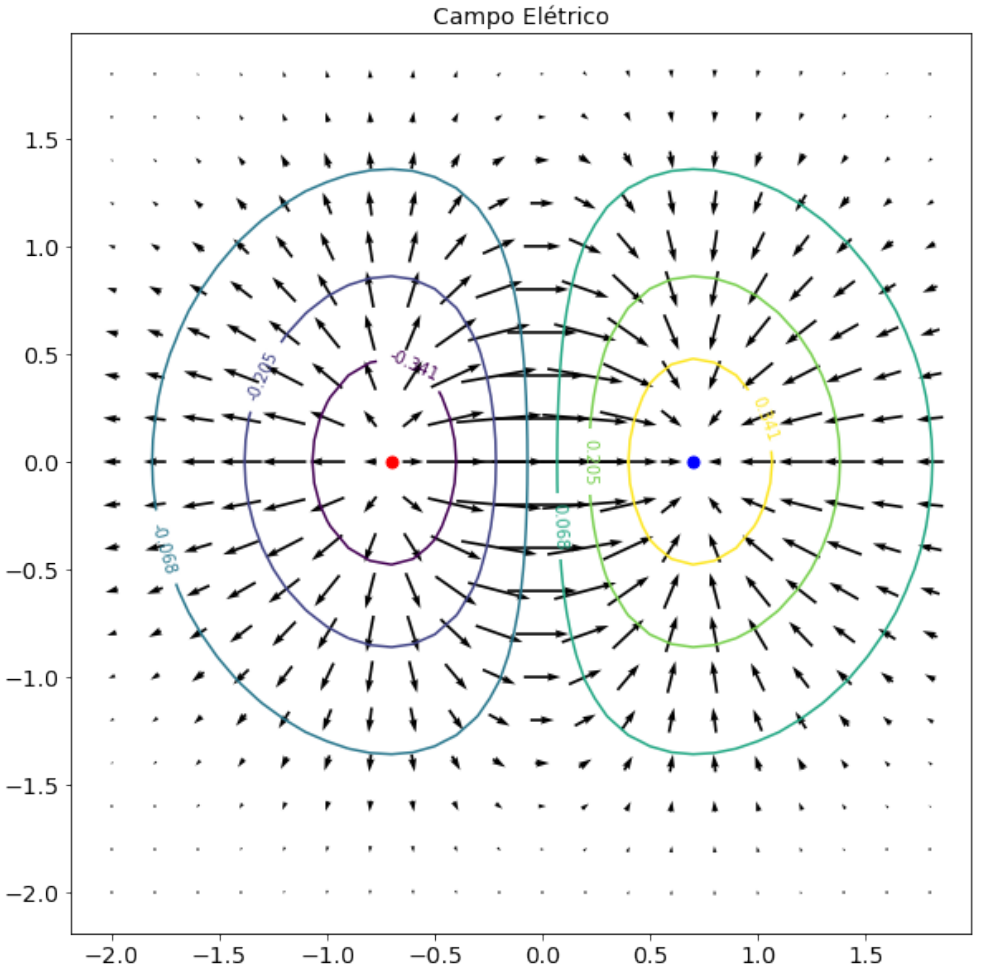


Figura 9.3: Gráfico do campo elétrico

9.3 Exercícios propostos

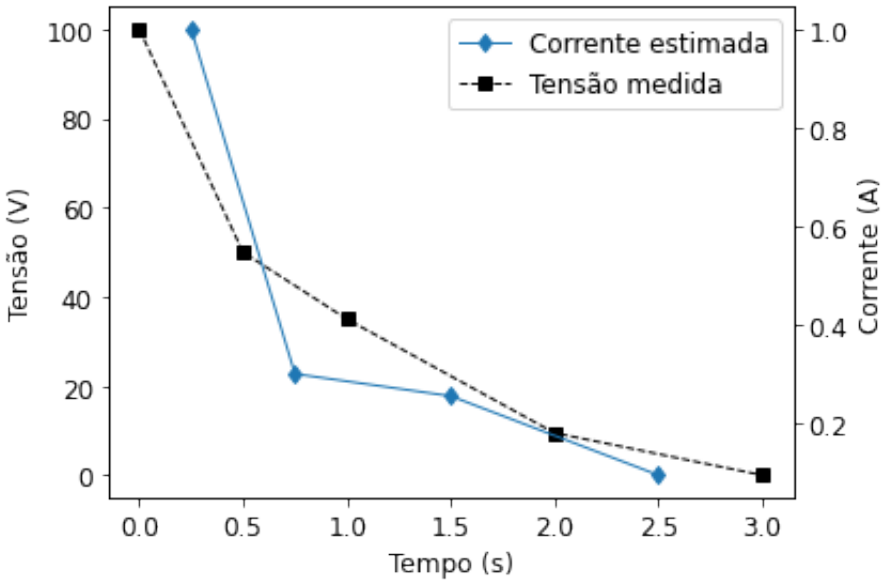
1. A corrente em um capacitor (I_c) é dada por [1]:

$$I_c = C\frac{dV}{dt},$$

onde C é a capacitância (F), V é a tensão (V) e t é o tempo (s). Determine a corrente em função do tempo a partir dos valores medidos da tensão para $C = 10$mF.

t(s)	0	0,50	1	2	3
V(V)	100	50	35	9,50	0,05

Resposta:



2. Refaça o exercício do campo elétrico proposto na seção 9.2, entretanto com as duas cargas sendo negativas.

3. A distância percorrida por um foguete em função do tempo foi analisada e obteram-se os seguinte dados [4]:

t (s)	0	30	60	90	120	150
y (km)	0	38	62	84	105	122

Com métodos de derivação numérica, obtenha as estimativas da **velocidade** e da **aceleração** em cada instante.

Resposta:

	t (s)	y (km)	v (m/s)	a (m/s²)
0	0	0	1266.67	NaN
1	30	38	1033.33	-15.56
2	60	62	766.67	-2.22
3	90	84	716.67	-1.11
4	120	105	633.33	-4.44
5	150	122	566.67	NaN

4. O ar se movimenta paralelamente a uma superfície plana, e sua velocidade v (em m/s) foi medida em diferentes distâncias y (em metros) em relação à superfície. Com base nesses dados, calcule a tensão de cisalhamento τ (em N/m²) junto à superfície, considerando a fórmula [4]:

$$\tau = \mu \frac{dv}{dy}$$

Considere que a viscosidade dinâmica do ar é $\mu = 2 \times 10^{-5}$ N $\cdot$ s/m^2.

Os valores experimentais são:

y (m)	0	0,004	0,008	0,014	0,020	0,026
v (m/s)	0	0,782	0,998	2,519	3,840	5,009

Resposta: 3,91e-03 N/m²

5. Diversas reações químicas obedecem à equação [4]:

$$\frac{dc}{dt} = -kc^n$$

em que c representa a concentração de uma substância, t é o tempo, k é a constante de velocidade da reação e n indica a ordem da reação. A partir dessa equação, é possível aplicar logaritmo e reescrevê-la como:

$$\log\left(-\frac{dc}{dt}\right) = \log k + n \log c$$

Utilize essa forma linearizada e os dados experimentais abaixo para encontrar os valores aproximados de k e n por meio de regressão linear:

t (s)	0	5	10	15	20	25	30
c	2,24	1,85	1,42	1,02	0,86	0,74	0,58

Resposta: n = 0,9194 ; k = 0,044742 1/s

9.4 Referências básicas

[1] NILSSON, J. W.; RIEDEL, S. A. **Circuitos Elétricos**, 10ª ed. Pearson, 2016. ISBN 9788543004785.

[2] FORNBERG, B. Numerical Differentiation of Analytic Functions. **ACM Transactions on Mathematical Software (TOMS)**, 1981.

[3] ZHANG, Y. **Advanced Differential Quadrature Methods**. CRC Press, 2009. ISBN 9781420082487.

[4] BASTOS, J. P. A.; IDA, N. **Electromagnetics and Calculation of Fields**, 2ª ed. Springer Verlag, 2012. ISBN 1461268605.

9.5 Referências complementares – Aplicações

[1] CAPIZZI, G.; COCO, S.; LAUDANI, A. Gaussian basis representation FIR filtering for the numerical differentiation of Laplacian FE solutions. In: **IEEE Transactions on Magnetics**, vol. 38, n. 2, p. 349-352, 2012. doi: 10.1109/20.996094.

[2] COCO, S.; LAUDANI, A. Numerical differentiation of Laplacian 3-D FE solutions by using regular polyhedra quadrature of Poisson integrals. In: **IEEE Transactions on Magnetics**, vol. 37, n. 5, p. 3104-3107, 2001. doi: 10.1109/20.952553.

[3] HOSSEINI, M. S.; PLATANIOTIS, K. N. Derivative Kernels: Numerics and Applications. In: **IEEE Transactions on Image Processing**, vol. 26, n. 10, p. 4596-4611, 2017. doi: 10.1109/TIP.2017.2713950.

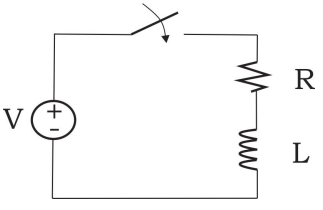
[4] CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**, 7ª ed. McGraw Hill Brasil, 2016. ISBN 9788580555684.

Equações Diferenciais Ordinárias

Em vez de descrever os estados de sistemas físicos diretamente, as leis da física usualmente são dadas em termos de variações temporais e espaciais. Esse tipo de equacionamento, quando composto por uma função desconhecida e suas derivadas, é chamado de equação diferencial [1].

Quando a função envolve apenas uma variável independente, a equação é conhecida como equação diferencial ordinária (EDO). Com duas ou mais variáveis independentes, como tempo, temperatura e espaço, chama-se equação diferencial parcial (EDP).

No circuito elétrico ilustrado, a quantidade que está sendo derivada (I) é chamada de variável dependente. A quantidade em relação à qual I é derivada é denominada variável independente (t) [2].



$$-V + RI(t) + L\frac{dI}{dt} = 0 \qquad \text{Física}$$

$$\frac{dI}{dt} = \frac{1}{L}(V - RI) \qquad \text{EDO}$$

$$I_{i+1} = I_i + \frac{1}{L}(V - RI)\Delta t \qquad \text{Solução numérica}$$

Além da quantidade de independentes, o comportamento das variáveis independentes também é relevante em qualquer análise.

Um problema é chamado de "problema de valor inicial" quando todas as condições são especificadas para o mesmo valor inicial da variável independente. Caso as condições sejam especificadas para diversos valores diferentes da variável independente, chama-se de "problema de valor de

contorno" [1]. A Figura 10.1 ilustra os dois casos.

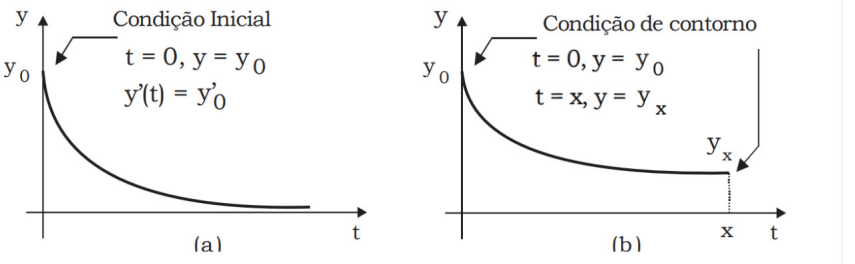


Figura 10.1: EDO: condição inicial e de contorno.

Este capítulo fica restrito ao estudo de EDO, caracterizado como problema de valor inicial.

Para o circuito RL proposto, por análise de malha tem-se:

$$-V + R \cdot I(t) + L\frac{dI}{dt} \quad \therefore \quad \frac{dI}{dt} = \frac{1}{L}(V - R \cdot I(t))$$

onde V é a tensão na fonte, R é a resistência (Ω) e L é a indutância (H) [2]. Determine o comportamento de $I(t)$.

Os métodos mais comuns resolvem EDO da forma:

$$\frac{dx}{dt} = f(x,t) \tag{10.1}$$

e de forma geral, eles possuem a seguinte construção:

$$x_{i+1} = x_i + \varnothing h \tag{10.2}$$

onde $\varnothing$ é chamado de função incremento e é utilizado para extrapolar de x_i para x_{i+1} em uma distância h (passo de cálculo).

Esses métodos são chamados de passo único, isso porque o valor da função incremento é baseado na informação de um único ponto i.

Existem vários métodos de passo único. Aqui será apresentado o Runge-Kutta de quarta ordem (RK4) [3]. Ele foi escolhido pois apresenta uma ótima relação acuracidade versus custo computacional.

10.1 Método Runge-Kutta de 4º ordem

RK4 utiliza o conceito de aproximação proposto por (10.2). A 4° ordem é dada pelos quatro coeficientes utilizados (k_1, k_2, k_3 e k_4), estimados da seguinte forma:

$$x_{i+1} = x_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

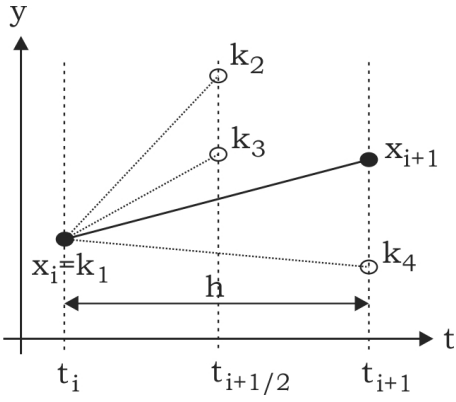


Figura 10.2: Runge-Kutta

$$k_1 = f(t_i, x_i)$$

$$k_2 = f(t_i + \frac{1}{2}h, x_i + \frac{1}{2}k_1 h)$$

$$k_3 = f(t_i + \frac{1}{2}h, x_i + \frac{1}{2}k_2 h)$$

$$k_4 = f(t_i + h, x_i + k_3 h)$$

O método Runge-Kutta de 4º ordem pode ser representado através do seguinte pseudocógico:

```
1  // Loop de integração com o método de Runge-Kutta de 4   ordem
2  WHILE t[it] < tf DO
3
4      // Cálculo de k1
5      k1 = (1 / L) * (V - R * i[it])
6
7      // Cálculo de k2 com valores intermediários
8      tx = t[it] + (1 / 2) * h
9      ix = i[it] + (1 / 2) * k1 * h
10     k2 = (1 / L) * (V - R * ix)
```

```
11
12      // Cálculo de k3 com valores intermediários
13      tx = t[it] + (1 / 2) * h
14      ix = i[it] + (1 / 2) * k2 * h
15      k3 = (1 / L) * (V - R * ix)
16
17      // Cálculo de k4 com valores finais
18      tx = t[it] + h
19      ix = i[it] + k3 * h
20      k4 = (1 / L) * (V - R * ix)
21
22      // Atualização da solução i(t) usando média ponderada dos k's
23      i[it + 1] = i[it] + (1 / 6) * (k1 + 2 * k2 + 2 * k3 + k4) * h
24
25      // Atualização do tempo
26      t[it + 1] = t[it] + h
27
28      // Avança o índice
29      it = it + 1
30
31  END WHILE
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  V = 10
5  R = 1
6  L = 0.001
7  tf = 0.006
8  h = tf/100
9  it = 0
10 r = tf/h
11 i = np.zeros(1+int(r))
12 t = np.zeros(1+int(r))
13
14 while t[it] < tf:
15     k1 = (1 / L) * (V - R * i[it])
16     tx = t[it] + (1 / 2) * h
17     ix = i[it] + (1 / 2) * k1 * h
18     k2 = (1 / L) * (V - R * ix)
19     tx = t[it] + (1 / 2) * h
20     ix = i[it] + (1 / 2) * k2 * h
21     k3 = (1/L) * (V - R * ix)
```

```
22      tx = t[it] + h
23      ix = i[it] + k3 * h
24      k4 = (1 / L) * (V - R * ix)
25      i[it + 1] = i[it] + (1 / 6) * (k1 + 2 * k2 + 2 * k3 + k4) * h
26      t[it + 1] = t[it] + h
27      it += 1
28
29  # Visualização
30
31  plt.plot(t, i, linewidth=2)
32  plt.grid(True)
33  plt.xlabel('Tempo (s)')
34  plt.ylabel('Corrente (A)')
35  plt.xlim(0, 0.006)
36  plt.ylim(0, 10)
37  plt.show()
```

Para o problema do circuito RL: V = 10V, R = 1Ω, L = 0,001 H, L sem energia inicial, e tempo final = 6 ms.

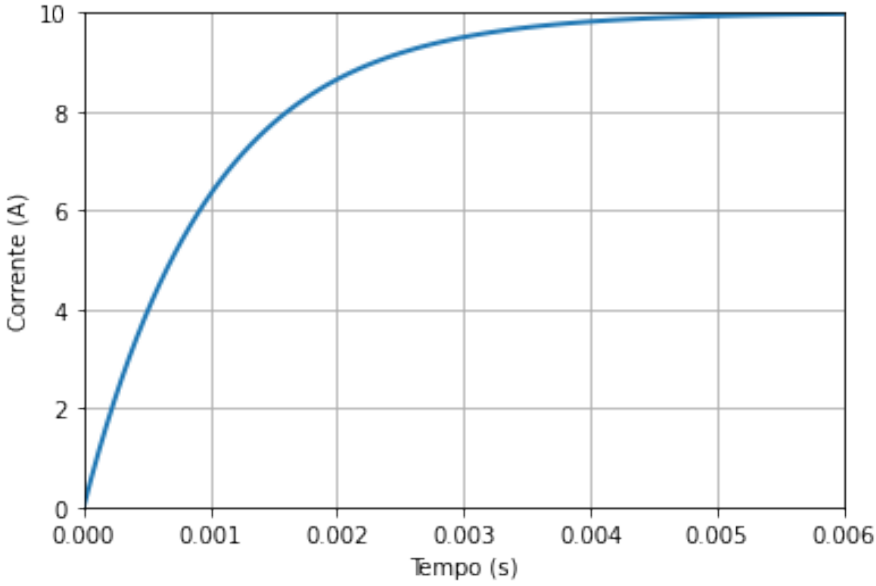
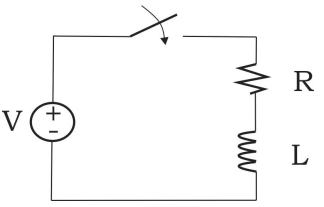


Figura 10.3: Comportamento da corrente no circuito RL

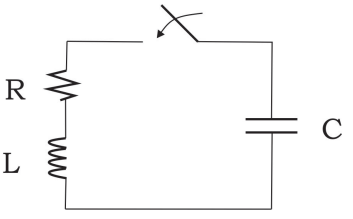
Por RK4 com passo de cálculo h = 0.006/100, obtém-se o comportamento da corrente ilustrado na Figura 10.3.

10.2 Exercícios propostos



1. Para um circuito elétrico proposto, onde V = 100V, R = 10Ω, L = 1 H, L sem energia inicial. Qual o valor da corrente para o tempo de 1s?

Resp.: 10 A.



2. Para um circuito elétrico proposto, onde R = 10Ω, L = 1 H, L sem energia inicial, C = 1μF e com tensão inicial de 10V. (a) em qual tempo a corrente no circuito chega a zero (±0,0005A)? (b) qual o valor da tensão no capacitor neste tempo?

Resp.: (a) 8 s. (b) 0 V

3. Considere o seguinte problema de valor inicial, onde deseja-se encontrar a solução analítica da equação diferencial no intervalo de $x = 1$ até $x = 3$: [3]

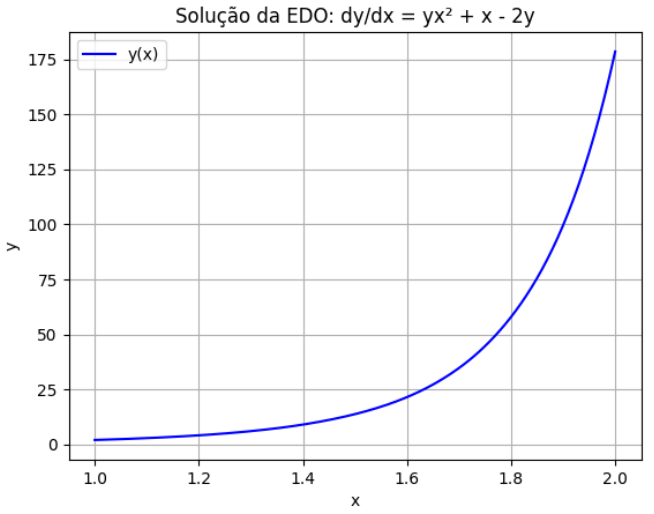
$$\frac{dy}{dx} = yx^2 + x - 2y$$

com a condição inicial definida por:

$$y(0) = 2$$

Determine a solução analítica da equação diferencial e, em seguida, apresente o gráfico da função resultante.

Resposta:



4. A partir da equação diferencial resolvida analiticamente no Exercício 3, agora aplique o método de Runge-Kutta clássico de quarta ordem para obter uma solução numérica. Utilize um passo de integração $h = 0{,}25$ no intervalo de $x = 0$ até $x = 2$. [3]

$$\frac{dy}{dx} = yx^2 + x - 2y, \quad y(0) = 2$$

Resposta:
x = 0,0 ; y = 2,000000
x = 0,2 ; y = 3,351324
x = 0,5 ; y = 5,849970
x = 0,8 ; y = 10,854916
x = 1,0 ; y = 21,996393
x = 1,2 ; y = 50,181796
x = 1,5 ; y = 133,035092
x = 1,8 ; y = 422,814637
x = 2,0 ; y = 1658,982131

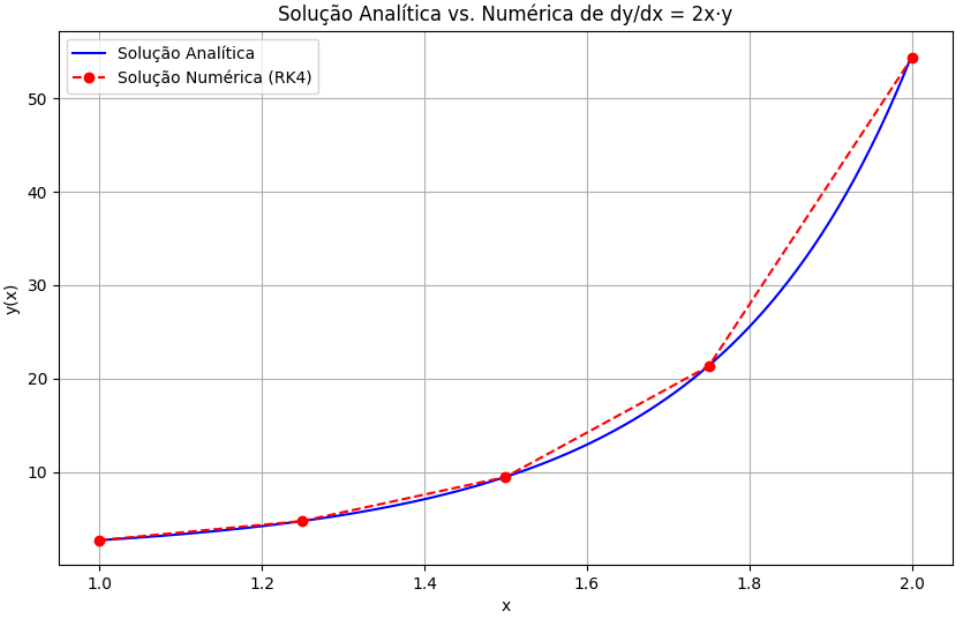
5. Considere uma nova equação diferencial e repita os procedimentos analítico e numérico aplicados nos Exercícios 3 e 4. Resolva o problema de valor inicial

a seguir no intervalo de $x = 1$ até $x = 2$: [3]

$$\frac{dy}{dx} = 2xy, \quad y(0) = e$$

Aplique tanto a resolução analítica quanto o método de Runge-Kutta de quarta ordem com passo $h = 0,25$.

Resposta:
Resultados Numéricos
x = 1,0 ; y = 2,718282
x = 1,25 ; y = 4,769188
x = 1,5 ; y = 9,477417
x = 1,75 ; y = 21,324428
x = 2,0 ; y = 54,298940



10.3 Referências básicas

[1] BOYCE, W. E.; DIPRIMA, R. C. **Elementary Differential Equations and Boundary Values Problems**, 5ª ed. John Wiley Sons, 1992. ISBN 0471509981.

[2] NILSSON, J. W.; RIEDEL, S. A. **Circuitos Elétricos**, 8ª ed. Pearson, 2008. ISBN 9788576051596.

10.4 Referências complementares – Aplicações

[1] RINK, R. A.; STREIFER, W. Application of Digital Computers to Solve Analytically a Class of Second-Order Non-linear Ordinary Differential Equations. In: **IEEE Transactions on Computers**, vol. C-20, n. 8, p. 901-910, 1971. doi: 10.1109/T-C.1971.223368.

[2] HUANG, C.; VAHID, F.; GIVARGIS, T. A Custom FPGA Processor for Physical Model Ordinary Differential Equation Solving. In: **IEEE Embedded Systems Letters**, vol. 3, n. 4, p. 113-116, 2011. doi: 10.1109/LES.2011.2170152.

[3] CHAPRA, S. C.; CANALE, R. P. **Métodos Numéricos para Engenharia**, 7ª ed. McGraw Hill Brasil, 2016. ISBN 9788580555684.

Equações Integrais

Uma equação integral é uma função desconhecida ($f(\Phi)$) sob um sinal de integração. Exemplos na engenharia elétrica de equações integrais são Fourier, Laplace e Maxwell.

Equações integrais lineares são usualmente classificadas em dois tipos [1]:

$$\text{(1) Fredholm:} \quad f(x) = a(x)\Phi(x) - \lambda \int_a^b K(x,t)\Phi(t)dt \tag{11.1}$$

onde λ é um escalar usualmente igual a 1. Funções $K(x,t)$ e $f(x)$ e os limites a e b são conhecidos, enquanto $\phi(x)$ é desconhecido; e

$$\text{(2) Volterra:} \quad f(x) = a(x)\Phi(x) - \lambda \int_a^x K(x,t)\Phi(t)dt \tag{11.2}$$

com destaque para o limite superior de integração. Se $f(x) = 0$, as equações (11.1) e (11.2) se tornam homogêneas.

Uma equação integral não linear acontece quando ϕ aparece com ordem $n > 1$ dentro da integral, por exemplo:

$$f(x) = a(x)\Phi(x) - \lambda \int_a^x K(x,t)\Phi^2(t)dt \tag{11.3}$$

A maioria das equações diferenciais ordinárias podem ser expressas como equações integrais, mas o inverso nem sempre é possível [1]. Isso é muito devido ao fato de que as condições de contorno são incorporadas nas equações integrais, enquanto nas equações diferenciais elas são imposições externas e complementares. Por exemplo, considerando uma equação diferencial ordinária de segunda ordem:

$$\frac{\partial^2 \Phi}{\partial x^2} = F(x, \Phi) \, , a \leq x \leq b \tag{11.4}$$

integrando,

$$\frac{\partial \Phi}{\partial x} = \int_a^x F(x, \Phi(t)) dt + c_1 \tag{11.5}$$

onde $c_1 = \phi'(a)$. Integrando por partes,

$$\Phi(x) = c_2 + c_1 x + \int_a^x (x - t) F(x, \Phi(t)) dt \tag{11.6}$$

onde $c_2 = \Phi(a) - \Phi'(a)$. Então,

$$\Phi(x) = \Phi(a) + (x - a)\Phi'^{(a)} + \int_a^x (x - t) F(x, \Phi) dt \, . \tag{11.7}$$

A equação (11.7) representa a equação (11.4) com suas condições de contorno. Aqui foi demonstrado para uma dimensão (x). Equações integrais envolvendo funções desconhecidas em mais dimensões podem ser encontradas em literatura dedicada ao tema [1][2].

Uma forma mais sistemática de obtenção de uma equação integral a partir de uma equação diferencial ordinária ou parcial é a construção de uma função auxiliar conhecida como $Green's function$ [1][2]. A função de Green é uma série de expansão, portanto, ela lineariza problemas não lineares. Isto pode ser um atrativo ou uma dificuldade, dependendo do problema a ser resolvido.

Entre as técnicas numéricas para resolução de equações integrais, destacam-se o método dos elementos de contorno (BEM) e o método dos momentos (MoM). O MoM tem por fundamento a função de Green e será aqui detalhado.

Para a engenharia elétrica, em particular para o eletromagnetismo, as vantagens de formular um problema em termos de equações integrais reside no fato de que essa formulação incorpora a identidade de Sommerfeld, a qual rege a propagação de ondas [3].

11.1 Método dos momentos

Seja a função de Green [2][4]:

$$L(f) = g \tag{11.8}$$

onde L é um operador qualquer conhecido, g é uma fonte ou excitação conhecida e f é o campo eletromagnético ou resposta.

A função desconhecida f é expandida em uma combinação linear de N funções, no domínio do operador L:

$$f = \alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n = \sum_{n=1}^{N} \alpha_n f_n \tag{11.9}$$

onde a_n são constantes desconhecidas e f_n é conhecida como função de base (ou de expansão). Substituindo (11.10) em (11.9), tem-se:

$$L = \left(\sum_{n=1}^{N} \alpha_n f_n \right) = g \;\; \therefore \;\; \sum_{n=1}^{N} (\alpha_n L(f_n)) = g \; . \tag{11.10}$$

Assumindo um produto escalar ajustável para $\langle f|g \rangle$, a solução do problema indicado pode ser determinada. Para isso definem-se funções de peso (ou de teste) da forma $w_1, w_2, \ldots, w_m$ no domínio de L, e faz-se o produto escalar da última equação para cada w_m. Assim:

$$\sum_{n=1}^{N} \alpha_n \langle w_m | L(f_n) \rangle = \langle w_m | g \rangle \;\; m = 1, 2, 3, \ldots, N \tag{11.11}$$

Tal equação transportada para a forma matricial gera:

$$[I_{mn}][\alpha_n] = [g_m]$$

$$[I_{mn}] = \begin{bmatrix} \langle w_1|L(f_1)\rangle & \cdots & \langle w_1|L(f_n)\rangle \\ \vdots & \ddots & \vdots \\ \langle w_m|L(f_1)\rangle & \cdots & \langle w_m|L(f_n)\rangle \end{bmatrix} \tag{11.12}$$

Se a matriz $[I_{mn}]$ é não singular, então $[I_{mn}]^{-1}$ existe, assim α_n é dado por:

$$[\alpha_n] = [I_{mn}]^{-1}[g_m] \tag{11.13}$$

com o valor de α_n encontrado determina-se o valor de f :

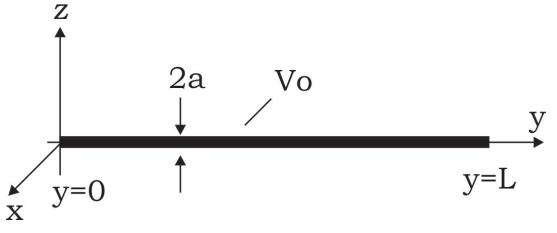
$$f = [f_n][\alpha_n] = [f_n]{I_{mn}}^{-1}[g_m] \tag{11.14}$$

A precisão da solução f depende das escolhas do tipo das funções de base e de peso, f_n e w_m respectivamente.

De forma breve, para se ter soluções mais exatas, pode-se assumir um número maior de funções de base e de peso. Um caso particular conhecido como Método de Galerkin é quando $f_n = w_m$ [2][3]. A função f_n deve ser linearmente independente e escolhida de tal maneira que os produtos escalares $\langle w_m|g\rangle$ sejam relativamente independentes das propriedades de g.

O problema

Considere um fio fino condutor de raio a e comprimento $L(L >> a)$ localizado no espaço livre [4].



Estando o fio num potencial Vo, deseja-se determinar a densidade de carga do fio.

Da equação de Poisson [3], tem-se:

$$V_0 = \int_0^L \frac{\rho_L dl}{4\pi\varepsilon_0 R}$$

Para um ponto fixo y_k no fio:

$$V_0 = \frac{1}{4\pi\varepsilon_0}\int_0^L \frac{\rho_L(Y)dy}{|y_k - y|} \ ,$$

se Δy é pequeno, pode-se considerar a seguinte aproximação:

$$\int_0^L f(y)dy = f(y_1)\Delta y + \cdots + f(y_N)\Delta y = \sum_{k=1}^{N} f(y_k)\Delta y$$

ou seja,

$$4\pi\varepsilon_0 V_0 = \frac{\rho_1\Delta}{|y_k - y_1|} + \frac{\rho_2\Delta}{|y_k - y_2|} + \cdots + \frac{\rho_N\Delta}{|y_k - y_N|} \quad \text{e} \quad \Delta = L/N$$

Transformando em matriz:

$$[B] = [A][\rho]$$

$$[B] = 4\pi\varepsilon_0 V_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad [A] = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \cdots & A_{NN} \end{bmatrix}$$

$$[\rho] = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_N \end{bmatrix} \qquad A_{mn} = \frac{\Delta}{|y_m - y_n|} \ \ m \neq n$$

Aqui, tem-se singularidades na diagonal principal. De forma mais rigorosa e considerando o fio condutor perfeito, portanto a densidade superficial de carga

aparece somente na superfície, e também desprezando-se a espessura do fio, chega-se a:

$$V(centro) = \frac{1}{4\pi\varepsilon_0}\int_0^{2\pi}\int_{-\Delta/2}^{\Delta/2}\frac{\rho_s a d\phi dy}{[a^2+y^2]^{\frac{1}{2}}} = \frac{2\pi a\rho_s}{4\pi\varepsilon_0} ln\left\{\frac{\Delta/2+[(\Delta/2)^2+a^2]^{1/2}}{-\Delta/2+[(\Delta/2)^2+a^2]^{1/2}}\right\}$$

Sendo $\Delta >> a$:

$$V(centro) = \frac{2\pi a\rho_s}{4\pi\varepsilon_0} 2ln\left\{\frac{\Delta}{a}\right\} \qquad \rho_L = 2\pi a\rho_s \qquad A_{nn} = 2ln\left\{\frac{\Delta}{a}\right\}$$

$$\begin{bmatrix} 2ln\left(\frac{\Delta}{a}\right) & \cdots & \frac{\Delta}{|y_1-y_N|} \\ \vdots & \ddots & \vdots \\ \frac{\Delta}{|y_N-y_1|} & \cdots & 2ln\left(\frac{\Delta}{a}\right) \end{bmatrix} \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_N \end{bmatrix} = 4\pi\varepsilon_0 V_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

De forma análoga, a carga Q pode ser calculada por:

$$Q = \int \rho_L dl \qquad Q = \sum_{k=1}^{N} \rho_k \Delta \tag{11.15}$$

O pseudocódigo a seguir resolve numericamente a equação (11.15) usando MoM.

```
// Função que calcula a carga total usando o Método dos Momentos
FUNCTION Metodo_dos_Momentos(Er, Eo, Vo, L, N)
    Delta = L / N

    // Vetor de posições dos centros dos segmentos
    FOR i = 0 TO N - 1 DO
        Y[i] = Delta * (i + 0.5)
    END FOR

    // Construção da matriz de influência A
    FOR i = 0 TO N - 1 DO
        FOR j = 0 TO N - 1 DO
            IF i      j THEN
```

```
                A[i][j] = Delta / ABS(Y[i] - Y[j])
            ELSE
                A[i][j] = 0.0
            END IF
        END FOR
    END FOR

    // Construção do vetor B com os potenciais
    FOR i = 0 TO N - 1 DO
        B[i] = 4 *     * Eo * Er * Vo
    END FOR

    // Solução do sistema linear: A * Rho = B
    C = INVERSA(A)
    Rho = MULTIPLICAR(C, B)

    // Cálculo da carga total Q
    Soma = 0
    FOR i = 0 TO N - 1 DO
        Soma = Soma + Rho[i]
    END FOR

    Q = Soma * Delta

    RETURN Q
END FUNCTION
```

Com a estrutura da função compreendida, pode-se aplicar a solução diretamente em Python:

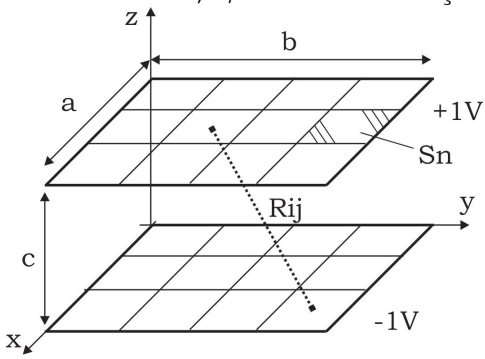
```
import numpy as np

Er = 1
Eo = 8.8541 * (10 ** -12)
Vo = 1
AA = 0.001
L = 1
N = 1000
Delta = L / N
I = np.arange(1, N + 1, 1)
Y = Delta * (I - 0.5)
A = np.zeros((1000, 1000))
for i in range(0, N):
    for j in range(0, N):
        if i != j:
            A[i][j] = Delta / abs(Y[i] - Y[j])
        else:
            A[i][j] = 0.0
```

```
19
20 B = 4 * np.pi * Eo * Er * Vo * np.ones((N, 1))
21 C = np.linalg.inv(A)
22 Rho = np.matmul(C, B)
23 Sum = 0
24 for I in range(0, N):
25     Sum = Sum + Rho[I]
26 I += 1
27 Q = Sum * Delta
28
29 print(f'Q = \n{Q}')
```

A precisão do resultado obtido depende da discretização da geometria (N). Para N = 10, Q = 8.5358e-12. Para N = 1000, Q = 8.6528e-12.

11.2 Exercício proposto

Determine a capacitância de um capacitor de placas paralelas. Seja a = b = c = 1, $\varepsilon_r = 1$ e a diferença de potencial das placas de 2V [4].



$$q = \int \rho_s ds \quad C = Q/V = Q/2$$

$$V_i = \sum_{j=1}^{2n} \rho_j A_{ij}$$

$$A_{ij} = \frac{1}{4\pi\varepsilon_0} \int_{\Delta Si}^{j} \frac{dS}{R_{ij}}$$

$$R_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,2n} \\ \vdots & \ddots & \vdots \\ A_{2n,1} & \cdots & A_{2n,2n} \end{bmatrix} \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{2n} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ -1 \end{bmatrix}$$

Para $i \neq j$

$$A_{ij} = \frac{\Delta S_i}{4\pi\varepsilon_0 R_{ij}} = \frac{(\Delta l)^2}{4\pi\varepsilon_0 R_{ij}}$$

Para $i = j$

$$A_{ij} = \frac{\Delta l}{\pi\varepsilon_0} ln(1+\sqrt{2}) = \frac{\Delta l}{\pi\varepsilon_0}(0,8814)$$

Resposta:

1. N = 9 , C = 26,5155 pF
   N = 25 , C = 27,7353 pF
   N = 100 , C = 28,6959 pF

2. Considere um fio condutor fino de comprimento $L = 1$ m e raio desprezível, submetido a um potencial constante $V_0 = 1$ V. Usando o Método dos Momentos com $N = 5$ segmentos uniformes, determine a densidade de carga linear ρ_L em cada segmento, utilizando aproximação ponto-médio e matriz de influência com:

$$A_{mn} = \begin{cases} \frac{\Delta}{|y_m - y_n|}, & m \neq n \\ \frac{\Delta}{a} \ln\left(\frac{2\Delta}{a}\right), & m = n \end{cases}$$

onde $a = 1$ mm. Monte a matriz $[A]$, o vetor $[B]$ e resolva para o vetor $[\rho]$.

Resposta:
$\text{rho}_1 = 1,271518e+00$
$rho_2 = 2,223187e-01$
$rho_3 = 3,719261e-01$
$rho_4 = 2,190491e-01$
$rho_5 = 1,694096e-01$

3. Dada a equação integral:

$$\int_0^1 \frac{f(y)}{x-y} dy = g(x), \quad 0 < x < 1$$

considere $g(x) = 1$ e resolva numericamente utilizando o método dos momentos com:

- Funções de base constantes por partes.
- $N = 4$ subdivisões.
- Pontos de colimação nos centros dos subintervalos.

Determine a matriz $[A]$, vetor $[g]$ e o vetor de coeficientes $[\alpha]$.

Resposta: **Vetor de coeficientes [$[\alpha]$]:** $\begin{pmatrix} 1,281791e+00 \\ -9,675501e-01 \\ 9,675501e-01 \\ -1,281791e+00 \end{pmatrix}$

4. Para o problema:

$$L(f) = f''(x) = x, \quad 0 < x < 1$$

com condições de contorno $f(0) = f(1) = 0$, resolva utilizando o método dos momentos com:

- Três funções de base: $f_n(x) = \sin(n\pi x)$ para $n = 1, 2, 3$.

Monte a matriz $[I]$, vetor $[g]$ e encontre as constantes α_n para aproximar $f(x)$.

Resposta: **Constantes alpha ($\alpha_1, \alpha_2, \alpha_3$):** $\begin{pmatrix} \alpha_1 = -6,450307e-02 \\ \alpha_2 = 8,062884e-03 \\ \alpha_3 = -2,389003e-03 \end{pmatrix}$

5. Refaça o exercício 2 utilizando agora:

- (a) $N = 3$
- (b) $N = 7$

Compare os resultados obtidos para ρ_L com os de $N = 5$. Discuta como a escolha de N afeta a precisão da aproximação no método dos momentos.

Resposta: Resultados da Densidade de Carga Linear (rho$_L$)$paradiferentesN$:

Para N = 3:

- $\rho_1 = 6,274682e-01$
- $\rho_2 = 2,621564e-01$
- $\rho_3 = 1,956739e-01$

Para N = 5:

- $\rho_1 = 1,271518e+00$
- $\rho_2 = 2,223187e-01$
- $\rho_3 = 3,719261e-01$
- $\rho_4 = 2,190491e-01$
- $\rho_5 = 1,694096e-01$

Para N = 7:

- $\rho_1 = 1,899701e+00$
- $\rho_2 = 1,279313e-02$
- $\rho_3 = 6,249373e-01$
- $\rho_4 = 2,192354e-01$
- $\rho_5 = 3,018945e-01$
- $\rho_6 = 1,804219e-01$
- $\rho_7 = 1,487179e-01$

11.3 Referências básicas

[1] RAY, S. S.; SAHU, P. K. **Novel Methods for Solving Linear and Nonlinear Integral Equations**, 1ª ed. Chapman and Hall/CRC, 2018. ISBN 9781138362741.

[2] SADIKU, M. N. O. **Numerical Techniques in Electromagnetics with Matlab**, 3ª ed. CRC Press, 2009. ISBN 9781420063097.

[3] BASTOS, J. P. A.; IDA, N. **Electromagnetics and Calculation of Fields**, 2ª ed. Springer Verlag, 2012. ISBN 1461268605.

[4] HARRINGTON, R. F. **Field Computation by Moment Methods**. Krieger Pub, 1982. ISBN 9780898744651.

11.4 Referências complementares – Aplicações

[1] TU, G.; LI, Y.; XIANG, J.; MA, J. Distributed Power System Stabiliser for Multimachine Power Systems. In: **IET Generation, Transmission Distribution**, vol. 13, n. 5, 2018. doi: 10.1049/iet-gtd.2018.6415.

[2] XIAO, L.; et al. An Efficient Hybrid Method of Iterative MoM-PO and Equivalent Dipole-Moment for Scattering From Electrically Large Objects. In: **IEEE Antennas and Wireless Propagation Letters**, vol. 16, p. 1723-1726, 2017. doi: 10.1109/LAWP.2017.2669910.

Continuação dos Estudos

Este é um livro de cálculo numérico aplicado à engenharia elétrica e as suas técnicas numéricas. Partindo de um problema de circuitos elétricos ou eletromagnetismo, por exemplo, o leitor é capaz de identificar a natureza matemática do problema. Assim o sendo, ele pode escolher a ferramenta numérica mais apropriada para a sua resolução.

Este é o primeiro passo em computação científica, a qual desenvolve modelos matemáticos e técnicas de soluções numéricas para analisar e resolver problemas científicos e de engenharia.

Computação científica é uma área em franco desenvolvimento, devido a dois motivos principais: (i) a crescente capacidade de processamento computacional; e (ii) a crescente quantidade de dados disponíveis (sensoriamento cada vez mais barato e a sua integração pela internet).

O desafio sempre será obter a modelagem matemática mais próxima possível da realidade, conforme discutido inúmeras vezes neste livro. Junta-se a isso agora, a grande quantidade de dados a serem interpretados. Transformar dados em informação útil para uma tomada de decisão inteligente é o estado da arte hoje.

Nesse contexto, sugere-se ao leitor continuar seus estudos por:

- **Pesquisa operacional** - é o uso de modelos matemáticos, estatística e algoritmos para ajudar na tomada de decisões. Muito utilizada dentro da administração e da engenharia de produção, a pesquisa operacional envolve técnicas como: teoria da dualidade e análise de sensibilidade, teoria de jogos, processo de decisão de Markov e teoria de filas [1][2].

- **Otimização** - é um processo para buscar a melhor solução de um determinado problema. Para modelagens lineares, usa-se a programação

linear, como, por exemplo, o método simplex [1]-[3]. Para equacionamentos não lineares, pode-se utilizar a programação não linear, na forma clássica com métodos baseados em derivada [1][4] ou com abordagens estocásticas (métodos baseados em probabilidade) [5][6]. Destaque aqui para uma nova família de ferramentas estocásticas chamadas de computação evolutiva ou bioinspirada, como os algoritmos genéticos, colônia de formigas, enxame de partículas ou sistemas imunológicos artificiais [7].

- **Inteligência artificial ou aprendizado de máquina** (machine learning) - são algoritmos construídos de forma a reconhecer padrões, aprender e agir a partir de dados, sem necessitar de instrução humana explícita [8]-[10]. De forma geral, o aprendizado pode ser supervisionado ou não supervisionado. Técnicas para aprendizado supervisionado buscam responder um objetivo, ou seja, há uma variável explícita a ser respondida (K-nearest neighbors e support vector machine são exemplos de técnicas). Métodos não supervisionados buscam identificar grupos ou padrões a partir dos dados, sem um objetivo específico a ser alcançado (mapas de Kohonen é a estratégia mais difundida). As chamadas redes neurais artificiais (RNA) podem ser implementadas com as duas orientações. RNA são modelos computacionais inspirados no sistema nervoso central.

O Grupo de Pesquisa em Computação Científica para Engenharia (PECCE), vinculado ao Instituto Federal de Santa Catarina (IFSC), do qual os autores deste livro são integrantes, desenvolve aplicações para a indústria em todos os temas aqui citados. Entre em contato.

Bons estudos!

12.1 Referências

[1] HILLER, F. S.; LEIBERMAN, G. J. **Introdução à Pesquisa Operacional**, 9ª ed. McGraw Hill, 2017. ISBN 9788580551181.

[2] ARENALES, M.; ARMENTANO, V. **Pesquisa Operacional para Cursos de Engenharia**. Elsevier, 2015. ISBN 9788535271614.

[3] MEIRELES, M.; TIOSSI, L. L. **Programação Linear - o método Simplex**, eBook Kindle, KaizenTools, 2016.

[4] EISELT, H. A.; SANDBLOM, C. L. **Nonlinear Optimization: Methods and Applications**. Springer, 2019. ISBN 9783030194611.

[5] MANDAL, J. K.; MUKHOPADHYAY, S.; DUTTA, P. **Multi-Objective Optimization: Evolutionary to Hybrid Framework**. Springer, 2019. ISBN 9789811346392.

[6] DATTA, R.; DEB, K. **Evolutionary Constrained Optimization**. Springer, 2016. ISBN 9788132235057.

[7] Laboratório de Bioinformática e Computação Bio-Inspirada, Universidade Estadual de Campinas, [online]. Disponível em: <http://www.lbic.fee.unicamp.br/homepage/indexpor.htm>. Acesso em: 24 ago. 2025.

[8] HUANG, G.; HUANG, G. B.; SONG, S.; YOU, K. Trends in extreme learning machines: A review. **Neural Networks**, vol. 61, p. 32-48, 2015. doi: 10.1016/j.neunet.2014.10.001.

[9] RASCHKA, V. M. S. **Python Machine Learning: Machine Learning and Deep Learning with Python**, 1ª ed. Packt Publishing, 2017.

[10] BAYOUDH, K. **From Machine Learning to Deep Learning**, 1ª ed., 2017, eBook. ISBN 9781387465606.

Índice Remissivo

INSTITUTO
FEDERAL
Santa Catarina

Câmpus
Florianópolis