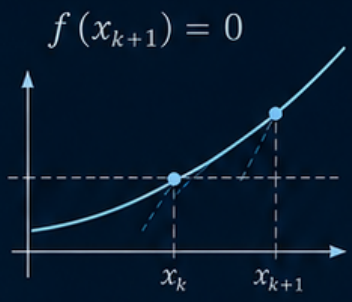
ROSANA MASSAHUD

Métodos Numéricos Computacionais:

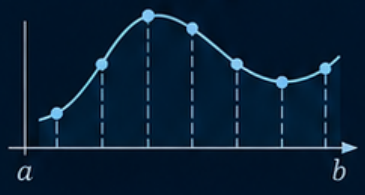
com implementações em Python



$$Ax = b$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} w_i\, f(x_i)$$



$$y'(t) = g(t, y), \quad y(t_0) = y_0$$

```python
import numpy as np

# Método da Bissecção
def bisseccao(f, a, b, tol=1e-8, maxit=100):
    fa, fb = f(a), f(b)
    if fa*fb > 0:
        raise ValueError('Intervalo inválido')

    for k in range(maxit):
        c = (a + b) / 2
        fc = f(c)
        if abs(fc) < tol or (b - a) / 2 < tol:
            return c, k+1
        if fa*fc < 0:
            b, fb = c, fc
        else:
            a, fa = c, fc
    return c, maxit

# Exemplo
f = lambda x: np.cos(x) - x
raiz, it = bisseccao(f, 0, 1)
print(f'Raiz aproximada: {raiz:.10f} em {it}iterações')
```

Raízes de Equações

Sistemas Lineares

Integração Numérica

EDOs e EDOs Numéricas

CEFET-MG

NEPOMUCENO - MG

2026

Rosana Massahud

MÉTODOS NUMÉRICOS COMPUTACIONAIS

Com implementações em Python

Colaboradores:

Jérifer Venceslau Pessin

Raphael Masson

Centro Federal de Educação Tecnológica de Minas Gerais
Nepomuceno - MG
2026

© 2026 Centro Federal de Educação Tecnológica (CEFET - MG).
Campus Nepomuceno.
Todos os direitos reservados.

Direção geral:
Israel Teodoro Mendes

Direção adjunta:
Anelise Lima de Abreu Dessimoni

Coordenação do curso técnico integrado em Eletrotécnica:
Zélia Maria Velloso Missagia

Coordenação do curso técnico integrado em Mecatrônica:
André Luís Marcomini

Coordenação do curso técnico integrado em Redes de Computadores:
Renata Barbosa de Oliveira

Coordenação de Assuntos Acadêmicos:
Cristhian Flamarion Gomes de Carvalho

Coordenação de Desenvolvimento Estudantil:
Ana Flávia Martins da Mata

Revisão Gramatical:
Trindade Monografias & Edições

Projeto Gráfico e Diagramação:
Rosana Massahud

Dados Internacionais de Catalogação na Publicação (CIP)

M414m Massahud, Rosana.
Métodos numéricos computacionais: com implementações em Python [recurso eletrônico] / Rosana Massahud – Nepomuceno: CEFET-MG, 2026.
232 p. : il. ; color.

ISBN 978-65-87888-46-0

1. Matemática computacional. 2. Análise numérica. 3. Python (Linguagem de programação) I. Título. II. Massahud, Rosana.

CDU-519.6:004.438

Simone dos Santos Moura Silva - CRB6-3315

Lista de Figuras

Lista de Algoritmos

Lista de Programas

Sumário

Apresentação

Este e-book nasceu da demanda da disciplina Métodos Numéricos Computacionais, ofertada no curso de Bacharelado em Engenharia Elétrica do CEFET-MG – Campus Nepomuceno. Seu propósito é fornecer um material atualizado, acessível e diretamente aplicável às atividades acadêmicas, reunindo fundamentos teóricos, algoritmos e implementações práticas em um único recurso didático.

A obra abrange conteúdos essenciais da área, incluindo noções de erros, métodos numéricos para sistemas lineares, interpolação, ajuste de curvas, regressão linear múltipla, integração numérica e solução numérica de equações diferenciais ordinárias (EDO's). Para apoiar a aprendizagem prática, cada método é apresentado com seu algoritmo em pseudocódigo e sua implementação em Python 3, linguagem escolhida por sua clareza, ampla adoção e importância crescente no contexto científico e tecnológico.

O capítulo inicial traz uma introdução à linguagem Python, oferecendo ao leitor as bases necessárias para compreender e adaptar os códigos desenvolvidos ao longo do livro. Além disso, como complemento ao estudo, disponibilizamos uma calculadora online contendo todos os métodos apresentados, permitindo a experimentação imediata dos algoritmos por meio de uma interface simples e intuitiva. A calculadora pode ser acessada em:

https://calculadora-mnc.streamlit.app/

Espera-se que este material contribua para a formação dos estudantes, ampliando sua autonomia na resolução de problemas numéricos e fortalecendo a relação entre teoria e prática – pilares fundamentais na engenharia contemporânea.

Capítulo 1

Softwares para Métodos Numéricos

1.1 Introdução

A utilização de ferramentas computacionais é fundamental no estudo e na aplicação dos métodos numéricos. Embora o foco deste material seja o desenvolvimento e a interpretação dos algoritmos, é igualmente importante dominar um ambiente de programação que permita implementá-los de forma clara, eficiente e reprodutível.

Ao longo deste e-book, adotaremos a linguagem Python 3 como base para os exemplos, atividades e exercícios práticos. Python destaca-se pela simplicidade de sua sintaxe, pela ampla disponibilidade de bibliotecas científicas e pela facilidade de integração com diferentes plataformas. Além disso, seu uso tem crescido significativamente em ambientes acadêmicos e profissionais, tornando-o uma escolha natural para disciplinas e projetos que envolvem cálculos numéricos, visualização de dados e simulação computacional.

Este capítulo apresenta os softwares e ferramentas necessários para o desenvolvimento dos métodos numéricos ao longo do curso. Começamos pela instalação e configuração do Python 3, seguidas de uma introdução aos principais conceitos da linguagem, visando preparar o leitor para os capítulos subsequentes, que tratam da implementação prática dos métodos estudados.

1.2 Instalação e Configuração do Python 3

Para acompanhar os exemplos desta apostila, é necessário ter o Python 3 instalado no computador. A instalação é simples e pode ser feita em qualquer sistema operacional (Windows, Linux ou macOS). A seguir, apresentamos um guia básico para instalar e configurar

o ambiente de desenvolvimento.

Instalação do Python 3

1. Acesse o site oficial Acesse o endereço oficial da linguagem Python:

   https://www.python.org/downloads

   Sempre utilize versões da série 3.x, que é a versão suportada pela comunidade e pela maioria das bibliotecas científicas.

2. Download e instalação

   - No Windows: baixe o instalador e marque a opção "Add Python to PATH" antes de confirmar.
   - No macOS: baixe o instalador correspondente ao seu sistema.
   - No Linux: em geral, o Python já vem instalado; caso contrário, ele pode ser obtido pelo gerenciador de pacotes da distribuição.

3. Verificando a instalação

   Após instalar, abra o terminal (ou Prompt de Comando no Windows) e digite:

   ```
   python --version
   ```

   ou

   ```
   python3 --version
   ```

   O sistema deverá retornar o número da versão instalada, por exemplo: Python 3.12.0.

Gerenciador de Pacotes: pip

O Python acompanha um gerenciador de pacotes chamado pip, usado para instalar bibliotecas adicionais. Para verificar seu funcionamento, utilize:

```
pip --version
```

Durante o estudo dos métodos numéricos, utilizaremos algumas bibliotecas específicas (como numpy e matplotlib). Elas podem ser instaladas com:

```
pip install numpy matplotlib
```

Ambientes Virtuais (opcional, mas recomendado)

Ambientes virtuais permitem criar instalações isoladas do Python, garantindo que diferentes projetos usem versões específicas de bibliotecas sem conflitos. Para criar um ambiente virtual, utilize:

```
python -m venv nome_do_ambiente
```

E para ativá-lo:

- Windows:

```
nome_do_ambiente\Scripts\activate
```

- Linux/macOS:

```
source nome_do_ambiente/bin/activate
```

Editor de Código

Embora o Python possa ser executado diretamente no terminal, é recomendado utilizar um editor de código para facilitar o desenvolvimento. Alguns editores recomendados são:

- Visual Studio Code;
- PyCharm;
- Sublime Text;
- Spyder (voltado para ciência e engenharia).

Ao longo desta apostila, você poderá utilizar o editor de sua preferência, desde que seja capaz de executar arquivos .py e visualizar saídas de texto e gráficos.

1.3 Introdução ao Python 3

Python é uma linguagem de programação interpretada de alto nível e que suporta múltiplos paradigmas/estilos de programação, como o imperativo, orientado a objetos e até o funcional. É uma linguagem com tipagem dinâmica e gerenciamento automático de memória.

Linguagem de alto nível:

Uma linguagem de alto nível possui maior proximidade com a linguagem humana do que com a linguagem de máquina (binário). A Figura 1.1 faz um comparativo entre a linguagem de máquina e a de alto nível para um algoritmo simples de soma entre dois números.

Linguagem Interpretada:

Isso significa que a implementação da linguagem Python em cada computador é feita por meio de um processo no qual um dos principais componentes é o interpretador. Existem algumas diferenças entre uma linguagem interpretada e uma linguagem compilada, mas esse não é o tema deste e-book.

A Figura 1.2 mostra um esquema simplificado do funcionamento de linguagens de programação que são interpretadas; observe o papel do interpretador.

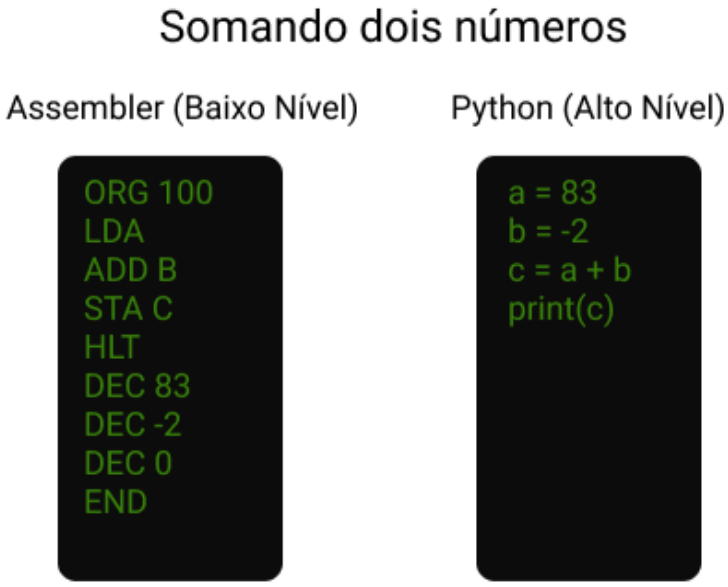


Figura 1.1: Comparação entre linguagem de máquina e linguagem de alto nível.

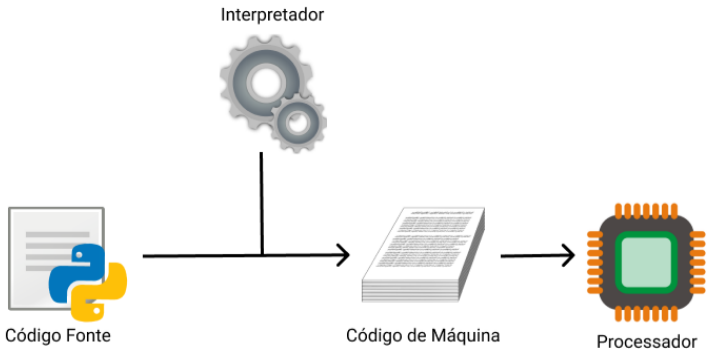


Figura 1.2: Esquema de linguagem interpretada.

Tipagem Dinâmica:

Dentro de uma linguagem de programação, trabalhamos constantemente com tipos de dados, alguns exemplos são: números inteiros (int), sequências de caracteres (str/string/-text) e operadores lógicos (boolean). Ter uma tipagem dinâmica significa que o próprio programa "entende" qual tipo de dados está sendo usado e, portanto, seu tipo não precisa ser previamente declarado. O Python possui tipagem dinâmica.

Gerenciamento Automático de Memória:

O Python constantemente realiza uma manutenção ou "limpeza" da memória não utilizada por meio de mecanismos como o *garbage collector* (coletor de lixo) e a *Reference Counting* (Contagem de Referência). Dessa forma, o programador não tem que se preocupar em fazer um gerenciamento manual de memória.

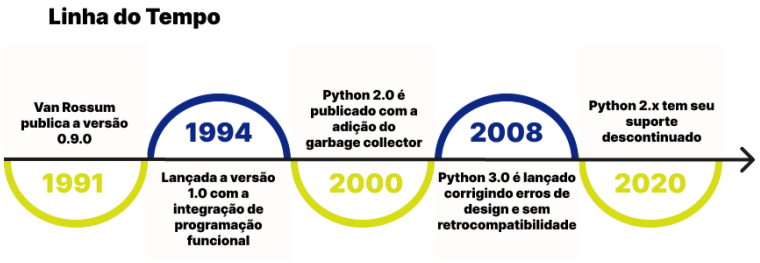


Figura 1.3: Linha do tempo da linguagem Python.

1.4 História

Python foi idealizado pelo programador holandês Guido Van Hossum e sua primeira versão foi lançada em 1991 no Centrum Wiskunde & Informatica - CWI (Instituto Nacional de Pesquisa para Matemática e Ciência da Computação), na Holanda.

Ao final da década de 1980, Guido Van Hossum trabalhava com uma linguagem de programação destinada ao uso de não programadores chamada ABC, amplamente utilizada no sistema operacional Amoeba. Inspirado na fácil sintaxe da linguagem, porém frustrado com o seu design monolítico e outras limitações, Van Rossum decidiu começar um projeto paralelo que levaria ao nascimento do Python.

A primeira versão da linguagem Python (0.9.0), lançada em 1991, incluía os conceitos de classes com herança, funções e os tipos fundamentais de dados. Desde então, a linguagem recebeu várias novas versões. Dessas, as mais famosas são as versões 2 e 3.

A Python Software Foundation (PSF), fundada no ano de 2001, é uma organização sem fins lucrativos que possui a propriedade intelectual relacionada à linguagem Python. Atualmente, a PSF é a responsável pela curadoria e administração geral das versões da linguagem. Desde sua concepção, o Python existe pelo princípio de tornar a programação um ato mais intuitivo e menos travado em sua própria estruturação. O método de desenvolvimento Open Source permite que sua própria comunidade de programadores e usuários ajudem em seu contínuo melhoramento.

Ao considerarmos a sua baixa curva de aprendizagem devido à sintaxe de simples compreensão, o Python acaba sendo uma das linguagens mais populares. Uma característica muito importante para a comunidade Python é a legibilidade do código, algo que reforça a ideia de ser uma linguagem mais próxima do humano e mais democrática.

É normal que a maioria das linguagens possuam atualizações constantes que contribuem para o seu avanço e melhoramento. A maioria dessas atualizações são feitas no funcionamento interno da linguagem e, portanto, não há preocupação por parte do programador, uma vez que não é necessário corrigir seus códigos. Esse não é o caso do embate Python 2 vs Python 3. Por apresentar uma sintaxe diferente das versões 2.x, a versão 3.x se tornou a primeira a ser incompatível com as versões anteriores e, no início de 2020, o suporte às versões 2.x foi descontinuado.

1.4.1 Utilização do Python no mercado

Desenvolvimento Web

A área de Desenvolvimento web engloba todas as atividades usadas para criar websites e aplicativos web-based. Existem duas subcategorias quando falamos em desenvolvimento web: O *Front-end* e o *Back-end*.

A linguagem Python entra nessa história pelos *frameworks* de desenvolvimento web *back-end*, o **Django** e o **Flask**. Esses frameworks são estruturas que facilitam a criação lógica do *back-end*, mapeando URLs, auxiliando a integração com bancos de dados e a criação de APIs.

Data Science

Nessa nova era do Big Data, saber extrair, manusear e analisar dados se tornou mais importante do que nunca e, uma hora ou outra, alguma linguagem de programação teria que se destacar pela sua eficiência nessas atividades.

O Python brilha em Data Science graças à sua sintaxe de fácil compreensão, implementação de interpretação imediata e suporte de múltiplas bibliotecas gráficas e estatísticas criadas constantemente pela comunidade. Sejam análises gráficas ou modelos preditivos feitos com Machine Learning, o Python acaba por ser o favorito dos cientistas de dados.

Podemos destacar algumas bibliotecas de extração, análise e visualização de dados, como a Pandas, Matplotlib e Seaborn. Já na área de Machine Learning, podemos mencionar a Scikit-learn, TensorFlow e Pytorch.

Automação e Scripting

Todos sonham em automatizar tarefas e deixar o computador fazer todo o trabalho repetitivo. Pois bem, o Python também pode fazer isso! Novamente, a fácil sintaxe e a velocidade da linguagem fazem diferença aqui, otimizando as tentativas e testes.

Por meio de bibliotecas como Selenium, PyAutoGUI e BeautifulSoup, podemos realizar automatização de tarefas de rotina: envio de e-mails ou mensagens ou até mesmo utilizar a técnica de Web Scraping para extrair dados de um web site.

1.5 Conceitos básicos

Os conceitos apresentados nesta seção fornecem a base necessária para escrever, entender e adaptar códigos em Python ao longo desta apostila. A maior parte dos algoritmos numéricos exige a manipulação de variáveis, estruturas condicionais, repetição, coleções e arquivos, além de ferramentas de visualização. A seguir, apresentamos esses elementos de forma simples e objetiva.

1.5.1 Variáveis

Variáveis são "caixas" que armazenam valores usados pelo programa. Em Python, não é necessário declarar o tipo antes de usar; basta definir um nome e atribuir um valor:

```
x = 10
nome = "Python"
```

Os nomes de variáveis devem começar com letra ou sublinhado e não podem conter espaços.

1.5.2 Tipagem - Tipos de dados

Python possui tipagem dinâmica, ou seja, o tipo é determinado automaticamente conforme o valor atribuído.

Principais tipos usados nesta apostila:

- int – números inteiros;
- float – números reais;
- str – textos;
- bool – valores lógicos: True ou False.

Exemplos:

```
a = 5          # int
b = 3.14       # float
c = "Métodos"  # str
d = True       # bool
```

1.5.3 Comentários de código

Comentários são trechos ignorados pelo interpretador e servem para documentar o código.

- Comentário de uma linha:

```
# Este é um comentário
```

- Comentário de múltiplas linhas:

```
"""
Comentário
que ocupa
várias linhas
"""
```

1.5.4 Indentação do código

Python utiliza indentação obrigatória para definir blocos de código. Normalmente usa-se 4 espaços por nível.

```
if x > 0:
    print("Positivo")
```

Erros de indentação são comuns e impedem o código de rodar.

1.5.5 Comandos de entrada e saída

Entrada de dados:

```
nome = input("Digite seu nome: ")
```

Saída de dados:

```
print("Olá,", nome)
```

O comando print() aceita textos, variáveis e expressões matemáticas.

1.5.6 Condicionais

São usadas para executar ações diferentes dependendo de uma condição lógica.

```
if x > 0:
    print("Positivo")
elif x == 0:
    print("Zero")
else:
    print("Negativo")
```

1.5.7 *Loops*

Loop for

Usado para percorrer sequências:

```
for i in range(5):
    print(i)
```

Loop while

Usado quando a repetição depende de uma condição:

```
while x < 10:
    x += 1
```

1.5.8 Coleções

Python fornece estruturas para armazenar múltiplos valores.

Listas (*list*)

Listas são mutáveis, indexadas e ordenadas.

```
lista = [1, 2, 3]
lista.append(4)
```

Tuplas (*tuple*)

Tuplas são coleções indexadas, porém, imutáveis.

```
coordenada = (3, 5)
```

Conjuntos (*set*)

Conjuntos não possuem ordem e não aceitam elementos repetidos.

```
s = {1, 2, 2, 3}   # s = {1, 2, 3}
```

Dicionários (*dict*)

Dicionários armazenam pares chave–valor.

```
pessoa = {"nome": "Ana", "idade": 25}
print(pessoa["nome"])
```

1.5.9 Leitura e escrita de arquivos

Python permite manipular arquivos de texto ou CSV com facilidade.

Leitura:

```
with open("dados.txt", "r") as f:
    conteudo = f.read()
```

Escrita:

```
with open("saida.txt", "w") as f:
    f.write("Resultado = 3.14")
```

CSV (simples):

```
import csv


with open("dados.csv") as f:
    leitor = csv.reader(f)
    for linha in leitor:
        print(linha)
```

1.5.10 Gráficos

Visualizações são importantes em métodos numéricos para análise de erros, convergência e resultados.

O pacote matplotlib

A biblioteca padrão é o matplotlib, especialmente o módulo pyplot.

Exemplo básico:

```
import matplotlib.pyplot as plt


x = [0, 1, 2, 3]
y = [0, 1, 4, 9]
```

```
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Exemplo de gráfico")
plt.show()
```

Estilos nos gráficos

O matplotlib possui estilos prontos:

```
plt.style.use("ggplot")
```

Outras formatações comuns:

```
plt.grid(True)
plt.legend()
plt.scatter(x, y)
```

Outros pacotes

Além do matplotlib, podemos citar:

- seaborn – gráficos estatísticos;
- plotly – gráficos interativos;
- pandas.plot – integração com dataframes.

Nesta apostila utilizaremos principalmente matplotlib.

1.6 NumPy

NumPy é a principal biblioteca para cálculos numéricos em Python. Fornece arrays multidimensionais e operações vetorizadas de alta performance.

1.6.1 Primeiros passos

```
import numpy as np
```

```
a = np.array([1, 2, 3])
```

```
b = np.zeros((2, 2))
c = np.linspace(0, 1, 5)
```

Funções úteis:

- np.ones;
- np.eye;
- np.arange.

1.6.2 Verificação de propriedades de um *ndarray*

```
x = np.array([[1, 2, 3],
              [4, 5, 6]])

x.ndim      # número de dimensões
x.shape     # formato (linhas, colunas)
x.size      # quantidade de elementos
x.dtype     # tipo dos dados
```

1.6.3 Indexação e fatiamento (*slicing*)

NumPy segue regras próximas às listas comuns, mas com maior flexibilidade.

```
v = np.array([10, 20, 30, 40])

v[0]      # 10
v[1:3]    # [20, 30]
```

Arrays 2D:

```
A = np.array([[1,2,3],
              [4,5,6]])

A[0, 1]     # elemento da linha 0, coluna 1
```

```
A[:, 2]     # toda a coluna 2

A[1, :]     # toda a linha 1
```

1.6.4 Manipulação de arrays Numpy

Operações comuns:

- Alterar formato

  ```
  B = A.reshape((3, 2))
  ```

- Empilhar arrays

  ```
  np.vstack((a, a))
  np.hstack((a, a))
  ```

- Operações matemáticas vetorizadas

  ```
  x = np.array([1, 2, 3])
  y = np.array([10, 20, 30])

  z = x + y       # soma elemento a elemento
  w = x * 2       # multiplicação escalar
  ```

- Agregações

  ```
  A.sum()     # soma
  A.mean()    # média
  A.max()     # máximo
  A.min()     # mínimo
  ```

Capítulo 2

Noções básicas sobre erros

2.1 Introdução

Estudaremos métodos numéricos para a resolução de problemas que surgem nas mais diversas áreas.

A resolução desses problemas envolve várias fases que podem ser estruturadas conforme mostrado na Figura 2.1:

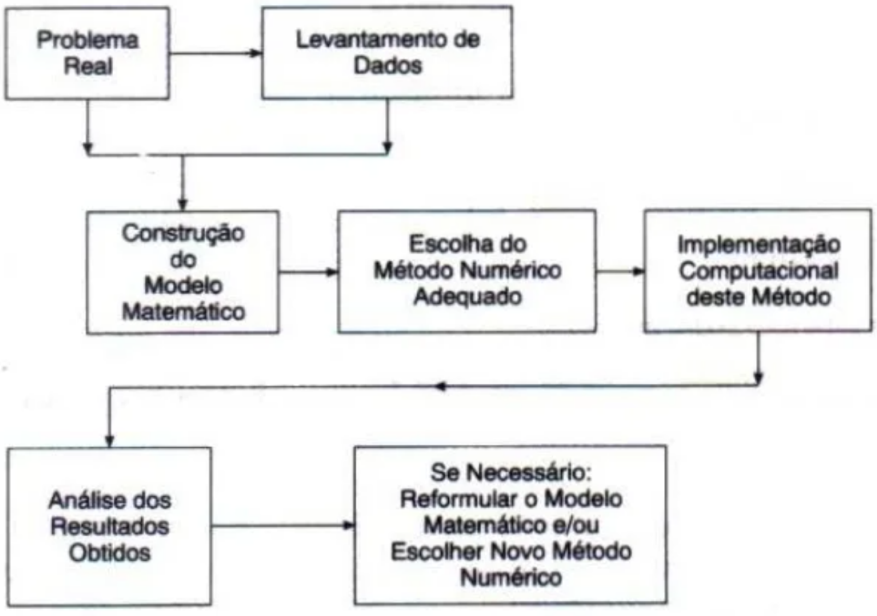


Figura 2.1: Fases na resolução de problemas.

Mesmo que todas as fases de resolução tenham sido realizadas corretamente, não é raro acontecer que os resultados finais estejam distantes do que se esperaria obter.

Os resultados obtidos dependem também:

- da precisão dos dados de entrada;
- da forma como esses dados são representados no computador;
- das operações numéricas efetuadas.

2.2 Representação de Números

Exemplo 1: Calcular a área de uma circunferência de raio 100 m.

Resultados Obtidos

a) A = 31400 m^2 ($\pi = 3.14$)
b) A = 31416 m^2 ($\pi = 3.1416$)
c) A = 31415.92654 m^2 ($\pi = 3.141592654$)

Como justificar as diferenças entre os resultados? É possível obter "exatamente" esta área?

Exemplo 2:

Efetuar o somatório seguinte em uma calculadora e em um computador:

$$S = \sum_{i=1}^{30000} x_i \text{ para } x_i = 0.5 \text{ e para } x_i = 0.11$$

Resultados Obtidos:

i) para $x_i = 0.5$:

na calculadora: $S = 15000$

no computador: $S = 15000$

ii) para $x_i = 0.11$:

na calculadora: $S = 3300$

no computador: $S = 3299.99691$

Como justificar as diferenças entre os resultados obtidos pela calculadora e pelo computador para $x_i = 0.11$? Os erros ocorridos nos dois problemas dependem da representação dos números na máquina utilizada. A representação de um número depende da base escolhida ou disponível na máquina em uso e do número máximo de dígitos na sua representação.

No Exemplo 1, o erro depende exclusivamente da aproximação escolhida para π.

Um número pode ter representação finita em uma base e não-finita em outras bases, como é o caso do Exemplo 2. A base decimal é a que mais empregamos atualmente, enquanto na Antiguidade foram utilizadas outras bases como a base 12 e a base 60. Um computador opera normalmente no sistema **binário**.

O que acontece na interação entre o usuário e o computador?

1. Os dados de entrada são enviados ao computador pelo usuário no sistema decimal.
2. Toda essa informação é convertida para o sistema binário e as operações todas serão efetuadas nesse sistema.
3. Os resultados finais serão convertidos para o sistema decimal e, finalmente, serão transmitidos ao usuário.

Todo esse processo de conversão é uma fonte de ***erros*** *que afetam o resultado final dos cálculos.*

2.2.1 Conversão de números nos sistemas decimal e binário

Conversão de números inteiros:

$$(347)_{10} = 3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

$$(10111)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Um número na base β, $(a_j a_{j-1} \ldots a_2 a_1 a_0)_\beta$, $0 \leqslant a_k \leqslant (\beta - 1)$, $k = 1, \ldots, j$, pode ser escrito na forma polinomial:

$$a_j\beta^j + a_{j-1}\beta^{j-1} + \cdots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0$$

Mudança de base binária para a base decimal

Procedimento: multiplicar o dígito binário por uma potência adequada de 2.

Exemplos:

a) $(10110)_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 = (22)_{10}$

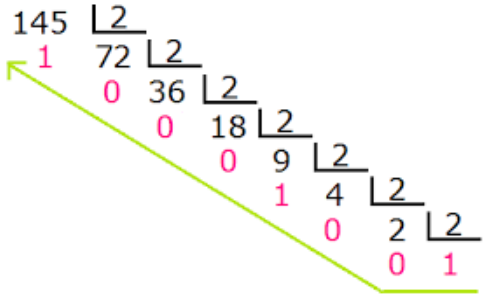
b) $(111)_2 = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = (7)_{10}$

Mudança de base decimal para a base binária

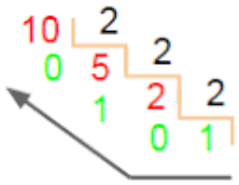
Procedimento: divisões sucessivas.

O procedimento consiste na divisão do número N na base decimal sucessivamente por 2, armazenando, a cada passo, os restos, $r_i i = n-1, n-2, \ldots, 1$ até que o quociente da divisão seja igual a 1. O número binário é constituído do quociente 1 e pelos restos das divisões $r_{n-1}, r_{n-2}, \ldots, r_1$ nessa ordem.

Exemplo 1: O número 145 na base 10 para a base 2,



Exemplo 2: O número 10 na base 10 para a base 2,



Conversão de números fracionários

De base decimal para base binária

Procedimento: multiplicações sucessivas. O procedimento é constituído dos seguintes passos:

a) Multiplicamos o número fracionário por 2.

b) Do resultado do passo a), a parte inteira é o primeiro dígito binário.

c) Do resultado do passo a), a parte fracionária é novamente multiplicada por 2.

d) O processo continua até que a parte fracionária seja nula.

Exemplo 3: O número 0.625 na base 10 para a base 2:

$0.625 \times 2 = 1.25 \longrightarrow 1$

$0.25 \times 2 = 0.5 \longrightarrow 0$

$0.5 \times 2 = 1.0 \longrightarrow 1$

$(0.625)_{10} = (0.101)_2$

Exemplo 4: O número 0.5 na base 10 para a base 2:

$0.5 \times 2 = 1.0 \longrightarrow 1$

$(0.5)_{10} = (0.1)_2$

Exemplo 5: O número 0.11 na base 10 para a base 2:

$0.11 \times 2 = 0.22 \longrightarrow 0$

$0.22 \times 2 = 0.44 \longrightarrow 0$

$0.44 \times 2 = 0.88 \longrightarrow 0$

$0.88 \times 2 = 1.76 \longrightarrow 1$

$0.76 \times 2 = 1.52 \longrightarrow 1$

$0.52 \times 2 = 1.04 \longrightarrow 1$

$0.04 \times 2 = 0.08 \longrightarrow 0$

$0.08 \times 2 = 0.16 \longrightarrow 0$

$0.16 \times 2 = 0.32 \longrightarrow 0$

$0.32 \times 2 = 0.64 \longrightarrow 0$

$0.64 \times 2 = 1.28 \longrightarrow 1$

$0.28 \times 2 = 0.56 \longrightarrow 0$

$0.56 \times 2 = 1.12 \longrightarrow 1$

$0.12 \times 2 = 0.24 \longrightarrow 0$

$0.24 \times 2 = 0.48 \longrightarrow 0$

$0.48 \times 2 = 0.96 \longrightarrow 0$

$0.92 \times 2 = 1.92 \longrightarrow 1$

$(0.11)_{10} = (0.00011100001010001\ldots)_2$

Exemplo 6: O número 22.25 na base 10 para a base 2:

$$(22.25)_{10} = (22)_{10} + (0.25)_{10} = (10110)_2 + (0.01)_2 = (10110.01)_2$$

Exemplo 7: O número 10.2 na base 10 para a base 2:

$$(10.2)_{10} = (10)_{10} + (0.2)_{10} = (1010)_2 + (0.001100110011\ldots)_2 = (1010.001100110011\ldots)_2$$

Conversão de bases numéricas em Python

Vamos escrever códigos em Python para conversão de bases numéricas. Neste texto, trataremos da conversão binário para decimal e vice-versa.

No Programa 2.1 temos funções de conversão de bases numéricas usando recursividade, loops ou funções built-in. A aplicação-exemplo usa todas as funções implementadas para exemplificar o funcionamento.

Programa 2.1: Métodos de conversão de bases numéricas binário-decimal

```
1
2 #Funcao para imprimir em binario,
3 #dada uma entrada em decimal
4
5 def decimal_to_binary(n):
```

```
6       if (n > 1):
7           # recursivamente, divide com o resultado integral
8           # (descarta o restante)
9           decimal_to_binary(n // 2)
10      print(n % 2, end=' ')
11
12
13  # Decimal para binario
14  # usando a funcao bin()
15  def decimal_to_bin(n):
16      return bin(n).replace("0b","")
17
18  # Funcao que imprime o decimal
19  # dado um numero em binario
20  def binary_to_decimal(binary):
21      decimal, i = 0, 0
22      while (binary != 0):
23          dec = binary % 10
24          decimal = decimal + dec * pow(2, i)
25          binary = binary // 10
26          i += 1
27      print(decimal)
28
29  # Conversao de numero binario (string) para decimal
30  # usando a funcao int()
31  def binary_to_dec(n):
32      return int(n, 2)
33
34  # Aplicacao
```

```
35 if __name__ == "__main__":
36     #decimal_to_binary(8)
37     print(decimal_to_bin(8))
38     print("\n")
39     #decimal_to_binary(18)
40     print(decimal_to_bin(18))
41     print("\n")
42     #decimal_to_binary(7)
43     print(decimal_to_bin(7))
44     print("\n")
45 
46     binary_to_decimal(100)
47     binary_to_decimal(101)
48     binary_to_decimal(1001)
49 
50     print(binary_to_dec("100"))
51     print(binary_to_dec("101"))
52     print(binary_to_dec("1001"))
53 
54     # Usando o metodo format()
55 
56     # Decimal (n) para binario
57     n = 4
58     print("{0:b}".format(n))
59 
60     binary = format(n, "b")
61     print(binary)
62     # Usando o metodo int()
63     # Binario para decimal
```

```
64    binary = "100"
65    decimal = int(binary, 2)
66    print(decimal)
```

2.2.2 Aritmética de ponto flutuante

Um computador ou calculadora representa um número real no sistema denominado *aritmética de ponto flutuante*. Nesse sistema, o número r será representado na forma:

$$r = \pm(.d_1 d_2 \dots d_t) \times \beta^e$$

Onde:

β é a base em que a máquina opera;

t é o número de dígitos na mantissa; $0 \leqslant d_j \leqslant (\beta - 1), j = 1, \dots, t, d_1 \neq 0$;

e é o expoente no intervalo $[l, u]$.

Exemplo 8: Em qualquer máquina, apenas um subconjunto dos números reais é representado exatamente e, portanto, a representação de um número real será realizada por meio de truncamento ou de arredondamento.

Considere uma máquina que opera no sistema: $\beta = 10$; $t = 3$; e $\in [-5, 5]$.

Informe:

- Os números representados.
- O menor número representado em valor absoluto.
- O maior número representado em valor absoluto.

Solução: $\beta = 10$; $t = 3$; e $\in [-5, 5]$.

- Números: $0.d_1 d_2 d_3 \times 10^e, 0 \leqslant d_j \leqslant 9, d_1 \neq 0, e \in [-5, 5]$.
- Menor número: $m = 0.100 \times 10^{-5} = 10^{-6}$.

- Maior número: $M = 0.999 \times 10^5 = 99900$.

Considere o conjunto dos números reais $\mathbb{R}$ e o seguinte conjunto.

$G = \{x \in \mathbb{R} \mid m \leqslant |x| \leqslant M\}$.

Dado um número real x, várias situações poderão acontecer:

- **Caso 1)** $x \in G$: Por exemplo: $x = 235.89 = 0.23589 \times 10^3$. Observe que este número possui 5 dígitos na mantissa. Se for usado *truncamento*, x será representado por 0.235×10^3 e, se for usado o *arredondamento*, x será representado por 0.236×10^3.

- **Caso 2)** $|x| < m$: Por exemplo: $x = 0.345 \times 10^-7$. Este número não pode ser representado nesta máquina porque o expoente, e, é menor que -5. A máquina acusa ocorrência de ***underflow***.

- **Caso 3)** $|x| > M$: Por exemplo: $x = 0.875 \times 10^9$. Neste caso, o expoente e é maior que 5 e a máquina acusa a ocorrência de ***overflow***.

Dar a representação dos números a seguir num sistema de aritmética de ponto flutuante de três dígitos para $\beta = 10$, $m = -4$ e $M = 4$.

Tabela 2.1: Exemplo de representação em aritmética de ponto flutuante

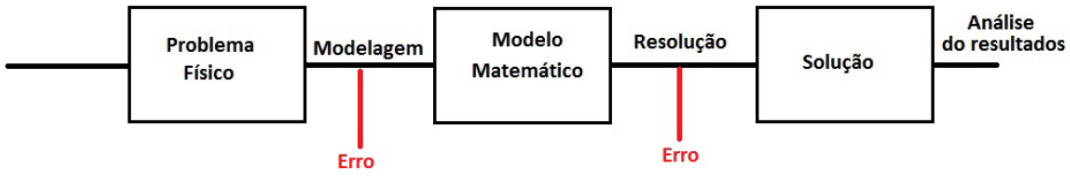
x	Representação obtida por arredondamento	Representação obtida por truncamento
1.25	0.125×10	0.125×10
10.053	0.101×10^2	0.100×10^2
-238.15	-0.238×10^3	-0.238×10^3
2.71828...	0.272×10	0.271×10
0.000007	(expoente menor que -4)	(expoente menor que -4)
718235.82	(expoente maior que 4)	(expoente maior que 4)

2.3 Tipos de erros

2.3.1 Etapas da solução de um problema

Durante as etapas de solução de um problema, surgem erros de várias fontes que podem alterar profundamente os resultados obtidos.

É de importância fundamental conhecer as causas desses erros para minimizar as suas consequências. No processo de solução de um problema físico por meio de aplicação de métodos numéricos, o erro pode aparecer:



2.3.2 Erros na fase de modelagem

Ao se tentar representar um fenômeno físico por meio de um modelo matemático, raramente temos uma descrição correta do fenômeno. Os erros na fase de modelagem são decorrentes de simplificações para a construção do modelo, muitas vezes necessárias para que o fenômeno da natureza que estivermos observando possa ser representado por um modelo matemático e que tenha condições de ser tratado com as ferramentas matemáticas disponíveis.

Os dados de entrada contêm imprecisão inerente: medidas são obtidas de equipamentos específicos ou são dados resultantes de pesquisas.

Exemplo 9: Um engenheiro quer medir a altura de um prédio. Ele tem um cronômetro e uma bolinha de metal e a equação abaixo:

$$s = s_0 + v_0 t + \frac{at}{2}$$

Onde:

- s: posição final;
- s_0: posição inicial;
- v_0: velocidade inicial;
- t: tempo;
- a: aceleração.

Para determinar a altura, o engenheiro sobe então ao topo do edifício, mede o tempo que a bolinha gasta para tocar o solo e encontra 3 segundos. Levando esse valor à equação anterior, temos:

$$s = 0 + 0 \times 3 + \frac{9.8 \times 3^2}{2}$$

$s = 44.1m$

Esse resultado é confiável?

Problema 1: Uso de simplificação no modelo

- Resistência do ar.
- Velocidade do vento.
- Velocidade dos ventos laterais.
- Entre outras.

Problema 2: Precisão das medidas

- Erro na medição da leitura do cronômetro.
- Para uma pequena variação no tempo medido, existe uma grande variação na altura do edifício. Exemplo: t = 3.5 segundos.

$$s = 0 + 0 \times 3.5 + \frac{9.8 \times (3.5)^2}{2} = 60.025m$$
$$\Delta t(\%) = 16.67\%$$
$$\Delta s(\%) = 36.11\%$$

  - Errar 0.5 segundo no tempo significa aumentar a altura do prédio em $15.9m$.

2.3.3 Erros na fase de resolução

Para resolução de modelos matemáticos é necessária a utilização de instrumentos de cálculo. Os erros na fase de resolução são provenientes da utilização de um computador para processarmos os cálculos necessários na obtenção de uma solução para o modelo. Tais erros ocorrem devido ao fato de os equipamentos terem capacidade limitada para armazenar dados numéricos utilizados nas operações elementares de adição, multiplicação, subtração e divisão.

Os erros nessa fase de resolução podem ser classificados em:

- Erros decorrentes do sistema de representação numérica na máquina.
- Conversões entre bases numéricas (Ex.: Num computador, decimal → binário → decimal).
- Erros de **truncamento**.
- Erros de **arredondamento**.

2.3.4 Erros de truncamento

Provenientes da utilização de processos que deveriam ser infinitos ou muito grandes para a determinação de um valor e que, por razões práticas são *truncados*.

Processos infinitos são utilizados na avaliação de funções matemáticas, como seno, exponencial, funções trigonométricas, entre outras.

O erro de truncamento é devido à aproximação de uma fórmula por outra. É sabido que, para avaliar uma função matemática no computador, somente as operações aritméticas e lógicas podem ser requeridas, por serem as operações que ele é capaz de efetuar.

Por exemplo, para avaliar $f(x) = sen(x)$ esta tem que ser aproximada por uma série, tal como,

$$sen(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots, 0 \leq x \leq \frac{\pi}{4}$$

À medida que n aumenta mais o valor da série se aproxima do valor real.

A Tabela 2.2 mostra a diferença entre o valor obtido pela séria de $sen(x)$ e um valor mais exato, para n até $2, 3$ e 4. Quando n aumenta, o erro de truncamento diminui, ficando claro que esses erros são devido aos truncamentos da série.

Tabela 2.2: Efeito de truncamento no cálculo de $sen(x)$

$\sum_{n=0}^{t}(-1)^n \frac{x^{2n+1}}{(2n+1)!} - sen(x)$			
x	$t = 2$	$t = 3$	$t = 4$
0	0	0	0
$\frac{\pi}{16}$	2.4×10^{-6}	2.2×10^{-9}	1.2×10^{-12}
$\frac{\pi}{8}$	7.8×10^{-5}	2.9×10^{-7}	6.1×10^{-10}
$\frac{\pi}{6}$	3.3×10^{-4}	2.1×10^{-6}	8.1×10^{-9}
$\frac{\pi}{4}$	2.5×10^{-3}	3.6×10^{-5}	3.1×10^{-7}

2.3.5 Erros de arredondamento

Como vimos, um número pode ter representação finita em uma base e não finita em outras bases. No Exemplo 2, o número 0.5 tem representação finita no sistema binário e o número 0.11 tem representação não finita. Alguns números decimais não podem ser exatamente representados por um computador. Ele tem que ser convertido para a base 2 e armazenado em um número finito de bits. O **erro de arredondamento** é causado por essa imperfeição na representação de um número. Para analisar as causas e consequências desse tipo de erro precisa-se conhecer como é realizada a conversão entre bases e como os números são armazenados em um computador.

2.3.6 Erros absolutos e relativos

O erro cometido na computação de um resultado pode ser medido de duas maneiras. A primeira é o **erro absoluto** definido como,

$$\text{erro absoluto} = \text{valor real} - \text{valor aproximado}$$

O tamanho do erro absoluto é mais grave quando o valor verdadeiro for pequeno. Por exemplo, 1711.321 ± 0.030 é exato com cinco dígitos significativos, enquanto 0.001 ± 0.030 tem pouco significado.

A outra maneira é o **erro relativo**, definido como:

$$\text{erro relativo} = \frac{\text{valor real} - \text{valor aproximado}}{\text{valor real}}, \text{sendo indefinido para valor real nulo}$$

A vantagem do erro relativo é a independência da magnitude dos valores.

2.3.7 Erros grosseiros

A possibilidade de um computador cometer um erro é muito pequena; no entanto, podem ser cometidos erros:

- na elaboração do algoritmo;

- na sua implementação;
- na digitação de dados.

Executar o programa, cujo resultado seja conhecido, ajuda a remover erros, mas demonstra, apenas, que o programa está correto para aquela massa de dados! A solução seria elaborar uma *prova de correção de programa* que é uma tarefa não trivial.

2.4 Exercícios

1. Converta de decimal para binário :

   a) 89_{10}

   b) 147_{10}

   c) 16.35_{10}

   d) 10.8_{10}

2. Converta de binário para decimal:

   a) 1101.1011_2

   b) 1110.1101_2

   c) 10011.1001_2

   d) 1000.01001_2

3. Escrever os seguintes números reais, que estão todos na base $\beta = 10$, 3 dígitos na mantissa e $e \in [-5, 5]$.

   a) $x_1 = 850$

   b) $x_2 = 5.1722$

   c) $x_3 = 0.00123$

   d) $x_4 = 0.3$

   e) $x_5 = 535391.35$

   f) $x_6 = x_1 + x_2$

g) $x_7 = x1 \times x_2$

Calcule os erros **absoluto** e **relativo** das operações em (f) e (g). Informe se os números serão ou não representados pela máquina.

4. Explique o que é erro de arredondamento e erro de truncamento. Dê exemplos.

5. Usando aritmética num sistema de ponto flutuante de 4 dígitos, base decimal e arredondamento, calcule o valor da seguinte soma:

$$S = \sum_{i=1}^{4}(x_i + y_i), \text{ sendo } x_i = 0.46709 \text{ e } y_i = 3.5678$$

Calcule os erros absoluto e relativo para essa operação.

6. Implemente uma função em Python que receba como entrada o valor real e o valor aproximado e retorne o erro absoluto e o erro relativo da aproximação.

Capítulo 3

Resolução de Sistemas Lineares

3.1 Conceitos Fundamentais de Matrizes

Uma matriz é um conjunto de elementos dispostos em forma retangular. O tamanho ou dimensão de uma matriz é definido pelo número de linhas e colunas. Uma matriz com m linhas e n colunas é dita ser $m \times n$ e se $m = n$, então ele é quadrada de ordem m. Os elementos da matriz são delimitados por colchetes ou parênteses.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Um elemento é referenciado por dois índices, o primeiro indica a linha e o segundo a coluna onde está o elemento.

3.1.1 Alguns tipos de matrizes

Vejamos a seguir alguns tipos de matrizes, seguidas por um esquema exemplo.

Matriz coluna: Uma matriz de tamanho $n \times 1$ é dita uma matriz coluna ou um vetor coluna de tamanho n.

$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{m1} \end{bmatrix}$$

Matriz linha: Uma matriz de tamanho $1 \times m$ é dita uma matriz linha ou um vetor linha de tamanho m.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \end{bmatrix}$$

Matriz nula: Se todos os elementos de uma matriz A forem iguais a zero, então ela é uma matriz nula, ou seja, $a_{ij} = 0, \forall i, j$.

$$\begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

Matriz diagonal: Uma matriz quadrada é dita diagonal se todos os elementos fora da diagonal principal forem nulos, isto é, se $d_{ij} = 0, \forall i \neq j$.

$$\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{mn} \end{bmatrix}$$

Matriz identidade: matriz identidade é uma matriz diagonal com os elementos da diagonal principal iguais a 1, ou seja, $e_{ij} = 1, \forall i = j$ e $e_{ij} = 0, \forall i \neq j$.

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Triangular inferior: Uma matriz triangular inferior é uma matriz com todos os elementos acima da diagonal principal iguais a zero, ou seja, $b_{ij} = 0, \forall i < j$.

$$\begin{bmatrix} b_{11} & 0 & 0 & \dots & 0 \\ b_{21} & b_{22} & 0 & \dots & 0 \\ b31 & b_{32} & b_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \dots & bmm \end{bmatrix}$$

Triangular superior: Uma matriz triangular superior é uma matriz com todos os elementos abaixo da diagonal principal iguais a zero, ou seja, $b_{ij} = 0, \forall i > j$.

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1m} \\ 0 & c_{22} & c_{23} & \dots & c_{2m} \\ 0 & 0 & c_{33} & \dots & c_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & c_{mm} \end{bmatrix}$$

Simétrica: Uma matriz é dita simétrica se houver uma simetria dos elementos em relação à diagonal principal, isto é, $m_{ij} = m_{ji}, \forall i, j$. Deste modo, uma matriz é simétrica se ela for igual à sua transposta, ou seja, $M = M^T$.

Exemplo:

$$M = \begin{bmatrix} 3 & 1 & 0 & 4 \\ 1 & 9 & 5 & 2 \\ 0 & 5 & 8 & 6 \\ 4 & 2 & 6 & 7 \end{bmatrix} \quad e \quad M^T = \begin{bmatrix} 3 & 1 & 0 & 4 \\ 1 & 9 & 5 & 2 \\ 0 & 5 & 8 & 6 \\ 4 & 2 & 6 & 7 \end{bmatrix}$$

3.1.2 Operações com matrizes

Operações envolvendo matrizes são de grande importância na solução de problemas. Algumas delas, mais relevantes para o assunto deste texto, são descritas a seguir.

Transposição: A transposta de uma matriz A, representada por A^T, é obtida trocando-se as linhas pelas colunas, de modo que a linha i torna-se a coluna i e a linha j torna-se a coluna j.

Exemplo: transposição da matriz A.

$$A = \begin{bmatrix} 5 & 2 & 0 \\ 6 & 9 & 1 \\ 3 & 3 & 2 \\ 7 & 0 & 8 \\ 1 & 5 & 6 \end{bmatrix} \quad e \quad A^T = \begin{bmatrix} 5 & 6 & 3 & 7 & 1 \\ 2 & 9 & 4 & 0 & 5 \\ 0 & 1 & 2 & 8 & 6 \end{bmatrix}$$

Adição e subtração: Se A e B forem matrizes de dimensão $n \times m$, então $C = A + B$ também será $n \times m$, tal que $c = a + b$, $\forall\ i, j$. É claro que apenas matrizes de mesmo tamanho podem ser somadas ou subtraídas.

Exemplo:

$$A = \begin{bmatrix} 8 & 6 \\ 1 & 4 \\ 5 & 7 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 2 & 1 \end{bmatrix}, C = A + B = \begin{bmatrix} 9 & 8 \\ 1 & 7 \\ 7 & 8 \end{bmatrix} \; eD = A - B = \begin{bmatrix} 7 & 4 \\ 1 & 1 \\ 3 & 6 \end{bmatrix}$$

Multiplicação: O Produto de uma matriz A de dimensão $n \times m$ por um escalar k resulta em uma matriz $B = kA$ de mesma dimensão $n \times m$, tal que $b_{ij} = ka_{ij}$, $\forall\ i, j$.

Exemplo:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} eB = 2A = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

O produto de uma matriz $A(n \times m)$ por um vetor $B(m \times 1)$ resulta em um vetor $x(n \times 1)$, de forma que

$$x_i = \sum_{j=1}^{m} a_{ij} v_j, i = 1, 2, \ldots, n$$

.

Exemplo

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \rightarrow x = Av = \begin{bmatrix} 5 \\ 11 \\ 17 \end{bmatrix}$$

O produto de uma matriz $A(n \times p)$ por uma matriz $B = (p \times m)$ é uma matriz $C = AB(n \times m)$, tal que

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, i = 1, 2, \ldots, n\ e\ j = 1, 2, \ldots, m$$

.

Exemplo:

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 3 & 5 & 6 \end{bmatrix}, B = \begin{bmatrix} 1 & 6 \\ 4 & 0 \\ 3 & 5 \end{bmatrix} \rightarrow C = AB = \begin{bmatrix} 6 & 12 \\ 41 & 48 \end{bmatrix}$$

Determinante: Uma matriz quadrada A de ordem n tem um número associado denominado determinante, cujo valor pode ser obtido pela fórmula de recorrência

$$det(A) = a_{11}det(M_{11}) - a_{12}det(M_{12}) + \cdots + (-1)^{n+1}a_{1n}det(M_{1n})$$

,

onde M_{ij} é a matriz de ordem $n-1$ resultante da remoção da linha i e coluna j de A e sendo o determinante de uma matriz (1×1) igual a esse único elemento.

$$A = \begin{bmatrix} a_{11} \end{bmatrix} \rightarrow det(A) = a_{11}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \rightarrow det(A) = a_{11}a_{22} - a_{21}a_{12}$$

e

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow$$

$$det(A) = a_{11}(a_{22}a_{33} - a_{32}a_{23}) - a_{12}(a_{21}a_{33} - a_{31}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22})$$

Uma matriz A com $det(A) = 0$ é dita singular e com $det(A) \neq 0$ é não singular.

Traço: O traço de uma matriz quadrada A é a soma dos elementos da sua diagonal principal

$$\text{traço}(A) = \sum_{i=1}^{n} a_{ii}$$

Exemplo: A matriz

$$M = \begin{bmatrix} 5 & -1 & 3 \\ 2 & 3 & 1 \\ 0 & 8 & 9 \end{bmatrix}$$

tem traço$(M) = 5 + 3 + 9 = 17$.

3.2 Noções de autovalores e autovetores

Seja a matriz:

$$A = \begin{bmatrix} 10 & -4 \\ 12 & -4 \end{bmatrix}$$

que apresenta a propriedade

$$\begin{bmatrix} 10 & -4 \\ 12 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \text{ ou seja:}$$

$\mathrm{A}v = \lambda v$

$\mathrm{A}v - \lambda v = 0$

$(\mathrm{A} - \lambda)v = 0$

$(\mathrm{A} - \lambda I_n)v = 0$

É dito, então, que a matriz A possui um autovalor $\lambda = 2$ e um correspondente autovetor $v = [1\ \ 2]^T$. O mesmo também é verdade para $\lambda = 4$ e $v = [2\ \ 3]^T$.

Uma possível interpretação geométrica é mostrada na Figura 3.1

Pela figura temos que:

- u é autovetor de T pois $\exists\ \lambda \in \mathbb{R} / T(u) = \lambda u$.
- v não é autovetor de T pois $\nexists\ \lambda \in \mathbb{R} / T(v) = \lambda v$.

O problema do autovalor consiste em encontrar a solução não trivial (ou não nula) do sistema homogêneo.

$$(A - \lambda I)v = 0$$

Teorema 1: Se M for uma matriz de ordem n, então o sistema homogêneo $My = 0$ tem solução não trivial se, e somente se, M for singular[1].

[1] Uma matriz singular tem determinante nulo.

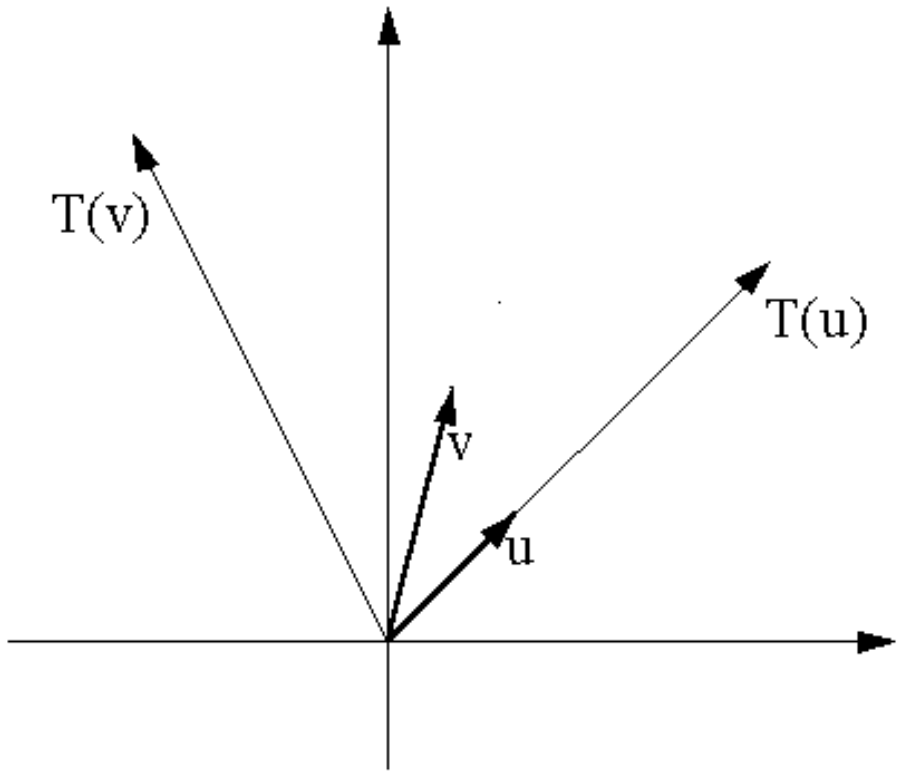


Figura 3.1: Interpretação geométrica de autovalores e autovetores.

Pelo Teorema 1 e sabendo que uma matriz singular tem determinante nulo, então,

$$det(A - \lambda I) = 0$$

Exemplo: Calcule os autovalores e seus correspondentes autovetores para a matriz

$$A = \begin{bmatrix} 10 & -4 \\ 12 & -4 \end{bmatrix}$$

Solução:

$\det(\text{A - } \lambda I)$

$$\det\left(\begin{bmatrix} 10-\lambda & -4 \\ 12 & -4-\lambda \end{bmatrix}\right) = 0, \text{ isto é,}$$

$(10 - \lambda)(-4 - \lambda) - 12(-4) = 0$

$\lambda^2 - 6\lambda + 8 = 0$

$\Delta = 36 - 32 = 4$

$\lambda = \frac{6 \pm 2}{2}$

$\lambda_1 = 2$

$\lambda_2 = 4$

Cálculo dos autovetores:

Para $\lambda = 2$

$$\upsilon_1 = \begin{bmatrix} x \\ y \end{bmatrix}$$

$A\upsilon_1 = \lambda_1\upsilon_1$

$(A - \lambda I)\upsilon = 0$

$$\left(\begin{bmatrix} 10 & -4 \\ 12 & -4 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 8 & -4 \\ 12 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} 8x - 4y = 0 \\ 12x - 6y = 0 \end{cases}$$

$4y = 8x \ (\div 4)$

$y = 2x$, o que é verdade para $x = 1 \rightarrow y = 2$

Logo, um possível autovetor para $\lambda = 2$ é

$$\upsilon_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Para $\lambda = 4$

$$\upsilon_2 = \begin{bmatrix} x \\ y \end{bmatrix}$$

$A\upsilon_2 = \lambda_2\upsilon_2$

$(A - \lambda I)\upsilon_2 = 0$

$$\left(\begin{bmatrix} 10 & -4 \\ 12 & -4 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 6 & -4 \\ 12 & -8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} 6x - 4y = 0 \\ 12x - 8y = 0 \end{cases}$$

$4y = 6x \ (\div 4)$

$y = \frac{6}{4}x$

$x = \frac{2}{3}y$

Fazendo y=3, temos, x = 2.

Logo, um possível autovetor para $\lambda = 4$ é

$$\upsilon_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Polinômio característico: O polinômio $\lambda^2 - 6\lambda + 8 = 0$ é chamado *polinômio característico* de A e seus n zeros λ_i são os autovalores de A.

É fácil notar que o polinômio característico de A pode ser escrito por $\lambda^2 - \text{traço}(A) + det(A)$.

> traço$(A) = \sum_{i=1}^{n} a_{ii}$, ou seja, o traço(A) é a soma dos elementos da diagonal principal de A.

Relações de Girard

- A soma dos elementos da diagonal principal de uma matriz (traço) é igual à soma dos seus autovalores.

$$\text{traço}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i$$

- O determinante de uma matriz é igual ao produto de seus autovalores.

$$det(A) = \prod_{i=1}^{n} \lambda_i$$

O processo de calcular os autovalores por intermédio da determinação dos zeros do polinômio característico, apesar de ser simples, é ineficiente em termos computacionais, e, portanto, é um processo não recomendado.

Cálculo de autovalores e autovetores usando Python

Em Python 3, temos as funções *eig()* e *eigvals()*, ambas da biblioteca NumPy, para cálculos de autovalores e autovetores. Veja o Programa 3.1:

Programa 3.1: Cálculo de autovalores e autovetores de uma matriz

```
1 A = np.array([[10,-4],[12,-4]])
2 print("A:\n",A)
3 resultado_A = np.linalg.eig(A)
4 print("Autovalores e autovetores de A:\n", resultado_A)
5 resultado_A2 = np.linalg.eigvals(A)
6 print(resultado_A2)
```

A função *eig()* retorna uma tupla com dois elementos:

- um array com os autovalores e
- um array com os autovetores.

Já a função *eigvals()* retorna uma lista com os autovalores da matriz.

Exercício resolvido:

Calcular os autovalores da matriz $A = \begin{bmatrix} 2 & 2 \\ 2 & -1 \end{bmatrix}$

Solução:

traço$(A) = 1$

$det(A) = -6$

$\lambda^2 - \lambda - 6 = 0 \rightarrow$ polinômio característico

$$\lambda = \frac{1 \pm \sqrt{(-1)^2 - 4 \times 1 \times (-6)}}{2} \rightarrow \begin{cases} \lambda_1 & = 3 \\ \lambda_2 & = -2 \end{cases}$$

É fácil verificar que:

traço(A) $= 1 = \sum_{i=1}^{2} \lambda_i = 3 + (-2)$

det(A) $= 2$(-1)-2 $\times 2 = -6 = \prod_{i=1}^{2} \lambda_i = 3 \times (-2)$.

3.3 Sistemas Lineares

Conjunto de m equações polinomiais com n variáveis x_i de grau 1.

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n & = & b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n & = & b_3 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n & = & b_m \end{array}$$

Forma matricial:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots\ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}$$

ou simplesmente, $Ax = b$, onde:

- $\boldsymbol{A}$ é a matriz de coeficientes;
- $\boldsymbol{x}$ é o vetor solução;
- $\boldsymbol{b}$ é o vetor de termos independentes.

3.3.1 Classificação dos sistemas

Classificação dos sistemas lineares quanto ao número de soluções

- Uma única solução: $det(A) \neq 0$;
- Infinitas soluções: $det(A) = 0$;
- Sem solução: $det(A) = 0$ e $\nexists\ x$.

3.4 Sistemas triangulares

3.4.1 Sistema triangular inferior

Na matriz de coeficientes, os elementos **acima** da diagonal principal são iguais a 0. A solução se dá por meio de **substituições sucessivas**.

Exemplo:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 \\ 1 & -6 & 8 & 0 \\ -1 & 4 & -3 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 48 \\ 6 \end{bmatrix}$$

Solução:

$2x_1 = 4 \rightarrow \boxed{x_1 = 2}$

$3x_1 + 5x_2 = 1$

$3 \cdot 2 + 5x_2 = 1 \rightarrow \boxed{x_2 = -1}$

$x_1 - 6x_2 + 8x_3 = 48$

$2 + 6 + 8x_3 = 48 \rightarrow \boxed{x_3 = 5}$

$-x_1 + 4x_2 - 3x_3 + 9x_4 = 6$

$-2 - 4 - 15 + 9x_4 = 6 \rightarrow \boxed{x_4 = 3}$

Vetor solução: $x = \begin{bmatrix} 2 & -1 & 5 & 3 \end{bmatrix}^T$

Procedimento substituições sucessivas:

O Algoritmo 1 resolve sistemas lineares do tipo triangular inferior. A implementação desse algoritmo em Python 3 é mostrado no Programa 3.2.

Programa 3.2: Função substituições sucessivas

```
1  import numpy as np
2
3  #funcao para resolucao de sistema triangular inferior
4  #algoritmo substituicoes sucessivas
5  def sucessiva(n, A, b):
6      x = np.zeros(n)
7      x[0] = b[0]/A[0][0]
8      print(x)
9      for i in range(1,n):
10         soma = 0
11         for j in range(i):
12             soma += A[i][j] * x[j]
13         x[i] = (b[i]-soma)/A[i][i]
14     return x
```

3.4.2 Sistema triangular superior

Na matriz de coeficientes, os elementos **abaixo** da diagonal principal são iguais a 0. A solução se dá por meio de **substituições retroativas**

Exemplo:

$$\begin{bmatrix} 5 & -2 & 6 & 1 \\ 0 & 3 & 7 & -4 \\ 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 28 \\ 8 \end{bmatrix}$$

Solução:

$2x_4 = 8 \rightarrow \boxed{x_4 = 4}$

$4x_3 + 5x_4 = 28$

$4x_3 + 5 \cdot 4 = 28 \rightarrow \boxed{x_3 = 2}$

$3x_2 + 7x_3 - 4x_4 = -2$

$3x_2 + 7 \cdot 2 - 4 \cdot 4 = -2 \rightarrow \boxed{x_2 = 0}$

$5x_1 - 2x_2 + 6x_3 + x_4 = 1$

$5x_1 - 2 \cdot 0 + 6 \cdot 2 + 4 = 1 \rightarrow \boxed{x_1 = -3}$

$$\text{Vetor solução: } x = \begin{bmatrix} -3 & 0 & 2 & 4 \end{bmatrix}^T$$

Procedimento substituições retroativas:

O Algoritmo 2 resolve sistemas lineares do tipo triangular superior. A função em Python 3 que implementa esse algoritmo é mostrada no Programa 3.3.

Programa 3.3: Função substituições retroativas

```
1  import numpy as np
2
3  #funcao para resoluca de sistema triangular superior
4  #algoritmo substituicoes retroativas
5  def retroativa(n,A,b):
6      x = np.zeros(n)
7      x[n-1] = b[n-1] / A[n-1][n-1]
8      for i in range(n-2, -1,-1):
9          soma = 0
10         for j in range(i+1, n):
11             soma += A[i][j] * x[j]
12         x[i] = (b[i] - soma) / A[i][i]
13     return x
```

Observação: na linguagem **Python**, a contagem dos elementos em listas, vetores e outras estruturas indexáveis **inicia em zero**.

Isso significa que o **primeiro elemento** de um vetor é acessado pelo índice `0`, o **segundo** pelo índice `1`, e assim por diante.

```
1 x = [10, 20, 30, 40]
2 print(x[0])  # Saida: 10
3 print(x[1])  # Saida: 20
```

Essa convenção é conhecida como *indexação baseada em zero* (*zero-based indexing*) e é adotada em diversas linguagens de programação, como C e Java.

3.5 Métodos para resolução de sistemas lineares

Existem duas classes de métodos para a resolução de sistemas lineares: métodos diretos (ou exatos) e iterativos (ou abertos). Os métodos diretos são aqueles em que a solução exata do sistema é obtida, teoricamente, com um número finito de operações aritméticas. Na prática, os erros de arredondamento devido à aritmética de ponto flutuante interferem no resultado verdadeiro. Os métodos iterativos obtêm a solução exata somente com um número infinito de operações. Em cada passo desses métodos a solução é calculada com um nível de exatidão crescente, o qual é limitado pelo número finito de bytes utilizados para armazenar as variáveis do programa que implementam o método iterativo.

Métodos exatos

- Eliminação de Gauss;
- Decomposição L.U.;
- Decomposição de Cholesky.

Métodos abertos

- Gauss-Jacobi;
- Gauss-Seidel.

3.6 Métodos exatos

3.6.1 Eliminação de Gauss

Consiste em transformar o sistema $Ax = b$ em um sistema equivalente com matriz de coeficientes triangular superior, por meio de transformações elementares.

$Ax = b \xrightarrow{T.E} \tilde{A}x = \tilde{b}$

Teorema: Seja $Ax = b$ um sistema linear. Se:

a) trocarmos duas equações;

b) multiplicarmos uma equação por uma constante não nula;

c) adicionarmos um múltiplo de uma equação a outra equação,

obteremos um sistema equivalente $\tilde{A}x = \tilde{b}$.

O método da Eliminação de Gauss engloba 2 fases:

i) fase da eliminação;

ii) fase da substituição.

Fase da eliminação

i.1) Transformações elementares na matriz aumentada $[A|b]$;

i.2) Para uma matriz (n), este processo terá $(n-1)$ iterações.

Procedimento: *fase da eliminação*

a) Montar a matriz aumentada $[A|b]$;

b) Determinação do pivô: a_{kk};

c) Definir os multiplicadores de linha,

$$m_{ik} = \frac{a_{ik}}{a_{kk}}$$

d) Atualização das linhas,

$$L_i \leftarrow L_i - m_{ik} \times L_{\text{pivô}}$$

Exemplo: Resolver o sistema abaixo usando o método de eliminação de Gauss.

$$\begin{cases} 3x_1 + 2x_2 + 4x_3 &= 1 \\ x_1 + x_2 + 2x_3 &= 2 \\ 4x_1 + 3x_2 - 2x_3 &= 3 \end{cases}$$

Solução:

$$A^{(0)}|b^{(0)} = \left[\begin{array}{ccc|c} 3 & 2 & 4 & 1 \\ 1 & 1 & 2 & 2 \\ 4 & 3 & -2 & 3 \end{array}\right] \text{ pivô} = a_{11} = 3$$

Cálculo dos multiplicadores e atualização das linhas 2 e 3.

Multiplicador da linha 2:

$$mL_2 = \frac{a_{21}}{a_{11}} = \frac{1}{3}$$

Multiplicador da linha 3:

$$mL_3 = \frac{a_{31}}{a_{11}} = \frac{4}{3}$$

Atualização da linha 2:

$\mathrm{L}_2 \longleftarrow = L_2 - mL_2 \times L_1$

$$1 - \tfrac{1}{3} \times 3 = 0$$
$$1 - \tfrac{1}{3} \times 2 = \tfrac{1}{3}$$
$$2 - \tfrac{1}{3} \times 4 = \tfrac{2}{3}$$
$$2 - \tfrac{1}{3} \times 1 = \tfrac{5}{3}$$

Atualização da linha 3:

$\mathrm{L}_3 \longleftarrow = L_3 - mL_3 \times L_1$

$$4 - \tfrac{4}{3} \times 3 = 0$$
$$3 - \tfrac{4}{3} \times 2 = \tfrac{1}{3}$$
$$-2 - \tfrac{4}{3} \times 4 = \tfrac{-22}{3}$$
$$3 - \tfrac{4}{3} \times 1 = \tfrac{5}{3}$$

Matriz aumentada atualizada $A^{(1)}|b^{(1)}$, cálculo do pivô da próxima iteração.

$$A^{(1)}|b^{(1)} = \begin{bmatrix} 3 & 2 & 4 & | & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & | & \frac{5}{3} \\ 0 & \frac{1}{3} & \frac{-22}{3} & | & \frac{5}{3} \end{bmatrix} \text{ pivô} = a_{22} = \frac{1}{3}$$

Cálculo dos multiplicadores e atualização da linha 3.

$mL_3 = \frac{1}{3} \div \frac{1}{3} = 1 \quad L_3 \leftarrow= L_3 - mL_3 \times L_1$

Atualização da linha 3:

$\mathrm{L}_3 \leftarrow= L_3 - mL_3 \times L_1$

$0 - 1 \times 0 = 0$

$\frac{1}{3} - 1 \times \frac{1}{3} = 0$

$\frac{-22}{3} - 1 \times \frac{2}{3} = -8$

$\frac{5}{3} - 1 \times \frac{5}{3} = 0$

Matriz aumentada atualizada $A^{(2)}|b^{(2)}$.

$$A^{(2)}|b^{(2)} = \begin{bmatrix} 3 & 2 & 4 & | & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & | & \frac{5}{3} \\ 0 & 0 & -8 & | & 0 \end{bmatrix}$$

A matriz aumentada $A^{(2)}|b^{(2)}$ é triangular superior e, portanto, a solução do sistema será por substituições retroativas.

$$\begin{cases} 3x_1 + 2x_2 + 4x_3 = 1 \therefore x_1 = -3 \\ \frac{1}{3}x_2 + \frac{2}{3}x_3 = \frac{5}{3} \therefore x_2 = 5 \\ -8x_3 = 0 \therefore x_3 = 0 \end{cases}$$

Vetor solução: $x = \begin{bmatrix} -3 & 5 & 0 \end{bmatrix}^T$

Cálculo do resíduo: O resíduo é calculado para verificarmos a exatidão da solução

encontrada pelo método. Para isso, temos a seguinte fórmula:

Vetor resíduo: $r = b - Ax$

$$r = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 3 & 2 & 1 \\ 1 & 1 & 2 \\ 4 & 3 & -2 \end{bmatrix} \begin{bmatrix} -3 \\ 5 \\ 0 \end{bmatrix} \rightarrow r = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{ (Solução exata)}$$

Implementação do método em Python

O Algoritmo 3 resolve o sistema linear pela Eliminação de Gauss, utilizando as duas fases: eliminação e resolução. O algoritmo foi extraído de RUGGIERO e LOPES, 1996, tem como entradas a matriz de coeficientes a e o vetor de termos independentes b e usa o método *retroativa()*, Programa 3.3, na fase de resolução. A saída do algoritmo é x, a solução do sistema linear.

O Programa 3.4 implementa o método de eliminação de Gauss em Python 3.

Programa 3.4: Função Eliminação de Gauss

```
1  import numpy as np
2
3  def gauss(A, b):
4      """
5      :param A: matriz de coeficientes
6      :param b: matriz de coeficientes independentes
7      :return: solucao do sistema
8      """
9      # algoritmo do livro do Ruggiero
10     # supor que o elemento  que esta na posicao akk
11     # eh diferente de zero no inicio da etapa k
12     n = len(b)
13     A_copy = A.copy()
14     b_copy = b.copy()
```

```
15      for k in range(0,n-1):
16          for i in range(k+1,n):
17              if(A_copy[k][k]==0):
18                  return None
19              m = float(A_copy[i][k]/A_copy[k][k])
20              for j in range(k,n):
21                  A_copy[i][j] = float(A_copy[i][j]
22                      -float(m*A_copy[k][j]))
23              b_copy[i] = float(b_copy[i]) - float(m*b_copy[k])
24              A_copy[i][k] = 0
25      print(A_copy)
26      print(b_copy)
27      #fase da resolucao
28      return retroativa(A_copy,b_copy)
```

3.6.2 Pivotação parcial

O método de Gauss irá falhar quando um pivô for nulo, pois nesse caso não será possível calcular os multiplicadores m_{ij} utilizados na eliminação. Esse problema pode ser contornado pelo uso da estratégia da pivotação parcial, que consiste em escolher como pivô o maior elemento em módulo da coluna, cujos elementos serão eliminados. A pivotação parcial garante que o pivô seja não nulo, exceto quando a matriz for singular. Outra vantagem é que todos os multiplicadores satisfazem $-1 \leq m_{ij} \leq 1$, evitando, assim, que eles sejam muito grandes. Multiplicadores grandes podem ampliar os erros de arredondamento de modo a comprometer a solução do sistema.

A única diferença entre o método de eliminação de Gauss com e sem pivotação parcial está no modo de escolha do elemento pivô.

Exemplo: Resolver o sistema abaixo pelo método de Gauss com pivotação parcial.

$$\begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix}$$

O primeiro elemento pivô escolhido é $a_{31} = 4$ porque ele é o maior elemento em módulo da primeira coluna. Para eliminar os demais elementos da coluna (a_{11} e a_{21}), efetua-se $L_1 \leftarrow L_1 - m_{11}L_3$. O multiplicador da linha 1 $m_{11} = a_{11}/a_{31} = \frac{1}{4} = 0.25$ e o da linha 2 $m_{21} = a_{21}/a_{31}$ e $L_2 \leftarrow L_2 - m_{21}L_3$. Efetuando-se essas 2 operações l-elementares, o sistema equivalente intermediário conterá dois elementos nulos na primeira coluna

$$\begin{bmatrix} 0 & -1,5 & 0,75 \\ 0 & 5 & 1,5 \\ 4 & -6 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3,75 \\ -0,5 \\ 29 \end{bmatrix}$$

Para eliminar o elemento da segunda coluna, escolhe-se o maior elemento em módulo dessa coluna, sem considerar o elemento da linha pivotal L_3. Portanto, o elemento pivô será $a_{22} = 5$. Para eliminar a_{12}, basta $L_1 = L_1 - m_{12}L_2$, $m_{12} = a_{12}/a_{22} = (-1,5/5 = -0,3$. O sistema equivalente é:

$$\begin{bmatrix} 0 & 0 & 1,2 \\ 0 & 5 & 1,5 \\ 4 & -6 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3,6 \\ -0,5 \\ 29 \end{bmatrix}$$

o qual pode ser transformado num sistema triangular superior trocando-se a ordem de duas linhas, no caso, a primeira e a terceira linhas,

$$\begin{bmatrix} 4 & -6 & 5 \\ 0 & -5 & 1,5 \\ 0 & 0 & 1,2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 29 \\ -0,5 \\ 3,6 \end{bmatrix}$$

Este sistema é resolvido pelas substituições retroativas. Solução:

$$x = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix}^T$$

Vetor resíduo: $r = b - Ax$

$$r = \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix} - \begin{bmatrix} -1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{solução exata}$$

3.6.3 Decomposição L.U.

Consiste em fatorar a matriz A dos coeficientes na forma $A = LU$.

Ax = b

(LU)x = b

Multliplicação de matrizes

$A_{mxn} \cdot B_{nxp} = C_{mxp}$

LUx = b fazendo: Ux = y

Ly = b

Logo, teremos 2 sistemas simplificados para resolver:

$$\begin{cases} Ux = y \\ Ly = b \end{cases}$$

Mas quem são as matrizes L e U?

L = Lower Matriz triangular inferior com diagonal unitária e os demais elementos são os multiplicadores das etapas do escalonamento.

U = Upper Matriz triangular superior resultante do escalonamento. Nada mais é que a aplicação do método de Gauss na matriz A.

Teorema: Se o determinante de todos os menores principais[2] da matriz A forem não nulos, então a fatoração $A = LU$ é única.

Exemplo:

$$\begin{cases} 3x_1 + 2x_2 + 4x_3 &= 1 \\ x_1 + x_2 + 2x_3 &= 2 \\ 4x_1 + 3x_2 - 2x_3 &= 3 \end{cases}$$

$$\mathrm{A}^{(0)} = \begin{bmatrix} 3 & 2 & 4 \\ 1 & 1 & 2 \\ 4 & 3 & -2 \end{bmatrix}$$

$$\text{pivô} = a_{11} = 3$$
$$mL_2 = \tfrac{1}{3}$$
$$mL_3 = \tfrac{4}{3}$$
$$L_2 \leftarrow L_2 - mL_2 \cdot L_1$$
$$L_3 \leftarrow L_3 - mL_3 \cdot L_1$$

$$\mathrm{A}^{(1)} = \begin{bmatrix} 3 & 2 & 4 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & \frac{1}{3} & \frac{-22}{3} \end{bmatrix}$$

$$\text{pivô} = a_{22} = \tfrac{1}{3}$$
$$mL_3 = 1$$
$$L_3 \leftarrow L_3 - mL_3 \cdot L_2$$

$$A^{(2)} = \begin{bmatrix} 3 & 2 & 4 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & -8 \end{bmatrix}$$

$\mathrm{A}_{(2)} = U$

$$\begin{cases} Ux = y \\ Ly = b \end{cases}$$

[2]Menores principais:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Os determinantes de cada submatriz (1x1), (2x2) e (3x3) têm que ser $\neq 0$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{4}{3} & 1 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 3 & 2 & 4 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & -8 \end{bmatrix}$$

$\boxed{Ly = b}$

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{4}{3} & 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{aligned} y_1 &= 1 \\ \tfrac{1}{3}y_1 + y_2 &= 2 \\ \tfrac{4}{3}y_1 + y_2 + y_3 &= 3 \end{aligned}$$

$$\text{y} = \begin{bmatrix} 1 \\ \frac{5}{3} \\ 0 \end{bmatrix}$$

$\boxed{Ux = y}$

$$\begin{bmatrix} 3 & 2 & 4 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{5}{3} \\ 0 \end{bmatrix}$$

$$\begin{aligned} 3x_1 + 2x_2 + 4x_3 &= 1 \\ \tfrac{1}{3}x_2 + \tfrac{2}{3}x_3 &= \tfrac{5}{3} \\ -8x_3 &= 0 \end{aligned}$$

$$\text{x} = \begin{bmatrix} -3 \\ 5 \\ 0 \end{bmatrix}$$

Solução: $\text{x} = \begin{bmatrix} -3 & 5 & 0 \end{bmatrix}^T$

Implementação do método em Python

Programa 3.5: Função Decomposição L.U.

```
1  def decomposicao_LU(A):
2      """
3      :param A: matriz de coeficientes
4      :return: matriz decomposta
5      """
6      n = len(A)
7      A_copy = A.copy()
8      for k in range(0, n - 1):
9          for i in range(k + 1, n):
10             m = float(A_copy[i][k] / A_copy[k][k])
11             A_copy[i][k] = m
12             for j in range(k + 1, n):
13                 A_copy[i][j] = float(A_copy[i][j] -
14                     float(m * A_copy[k][j]))
15     return A_copy
```

3.6.4 Decomposição de Cholesky

Se a matriz A for simétrica[3] e definida positiva[4], então A pode ser decomposta na forma $A = L \cdot L^T$.

Teorema: Se A for uma matriz simétrica e definida positiva, então existe uma única matriz triangular inferior L, com elementos diagonais positivos tal que $A = L \cdot L^T$.

[3] **Matriz simétrica:** $a_{ij} = a_{ji}$

[4] **Matriz definida positiva**:

i) os autovalores de A são todos positivos

ii) menores principais são positivos

iii) $v^t Av > 0, \forall v \neq 0$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

Elemento $l_{11}, 1^a linha, 1^a coluna$

$a_{11} = l_{11}^2 \quad \therefore l_{11} = \sqrt{a_{11}} \quad (A)$

Demais elementos da $1^a coluna$

$a_{21} = l_{21} \cdot l_{11} \quad \therefore l_{21} = \frac{a_{21}}{l_{11}} \quad (B)$

$a_{31} = l_{31} \cdot l_{11} \quad \therefore l_{31} = \frac{a_{31}}{l_{11}} \quad (B)$

Demais elementos da $2^a coluna$

$a_{22} = l_{21} \cdot l_{21} + l_{22} \cdot l_{22} \quad \therefore l_{22} = \sqrt{a_{22} - l_{21}^2} \quad (C)$

$a_{32} = l_{31} \cdot l_{21} + l_{32} \cdot l_{22} \quad \therefore l_{32} = \frac{a_{32} - l_{31} \cdot l_{21}}{l_{22}} \quad (D)$

Demais elementos da $3^a coluna$

$a_{33} = l_{31}^2 + l_{32}^2 + l_{33}^2 \quad \therefore l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} \quad (C)$

CHOLESKY	
A) $l_{11} = \sqrt{a_{11}}$	C) $l_{ii} = \sqrt{(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2)}$
B) $l_{i1} = \frac{a_{i1}}{l_{11}}$	D) $l_{ij} = \frac{(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk})}{l_{jj}}$

Temos uma ordem lógica para usar o formulário dado acima e calcular a matriz L. Primeiro calculamos o primeiro elemento da diagonal principal l_{11} (fórmula A). Em seguida, os demais elementos da primeira coluna (fórmula B). Depois, alternamos entre o cálculo do elemento da diagonal principal (fórmula C) e demais elementos da coluna (fórmula D). Sempre coluna por coluna.

Exemplo:

$$\begin{cases} 4x_1 + 12x_2 - 16x_3 &= 1 \\ 12x_1 + 37x_2 - 43x_3 &= 2 \\ -16x_1 - 43x_2 + 98x_3 &= 3 \end{cases}$$

$$A = \begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix} \rightarrow \text{A é simétrica e definida positiva.}$$

Cálculo dos elementos de L:

Elementos da primeira coluna:

$\mathrm{l}_{11} = \sqrt{a_{11}} = \sqrt{4} = 2$

$\mathrm{l}_{21} = \frac{a_{21}}{l_{11}} = \frac{12}{2} = 6$

$\mathrm{l}_{31} = \frac{a_{31}}{l_{11}} = \frac{-16}{2} = -8$

Elementos da segunda coluna:

$\mathrm{l}_{22} = \sqrt{(a_{22} - l_{21}^2)} = 1$

$\mathrm{l}_{32} = \frac{a_{32} - l_{31} \cdot l_{21}}{l_{22}} = \frac{-43 - (-8) \cdot 6}{1} = 5$

Elementos da terceira coluna:

$\mathrm{l}_{33} = \sqrt{a_{33} - (l_{31}^2 + l_{32}^2)}$

$\mathrm{l}_{33} = \sqrt{98 - ((-8)^2 + 5^2)} = 3$

Portanto, nossa matriz L é:

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{bmatrix}, \; L^t = \begin{bmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{bmatrix}$$

Se $Ax = b$ e $A = L \cdot L^T$, então:

Ax = b

$$(\mathrm{L} \cdot L^T)x = b \rightarrow \begin{cases} L^T x &= y \\ Ly &= b \end{cases}$$

$\boxed{\mathrm{Ly = b}}$

$$\begin{bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \rightarrow \begin{cases} 2y_1 = 1 \therefore y_1 = \frac{1}{2} \\ 6y_1 + y_2 = 2 \therefore y_2 = -1 \\ -8y_1 + 5y_2 + 3y_3 = 3 \therefore y_3 = 4 \end{cases} \quad y = \begin{bmatrix} \frac{1}{2} \\ -1 \\ 4 \end{bmatrix}$$

$\boxed{L^t x = y}$

$$\begin{bmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ -1 \\ 4 \end{bmatrix} \rightarrow \begin{cases} 3x_3 = 4 \therefore x_3 = \frac{4}{3} \\ x_2 + 5x_3 = -1 \therefore x_2 = \frac{-23}{3} \\ 2x_1 + 6x_2 - 8x_3 = \frac{1}{2} \therefore x_1 = \frac{343}{12} \end{cases} \quad x = \begin{bmatrix} \frac{343}{12} \\ \frac{-23}{3} \\ \frac{4}{3} \end{bmatrix}$$

Vamos verificar a exatidão da solução encontrada, calculando o vetor resíduo:

Vetor resíduo: $r = b - Ax$

$$r = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix} \cdot \begin{bmatrix} \frac{343}{12} \\ \frac{-23}{3} \\ \frac{4}{3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{Solução exata}$$

Agora, verificaremos a unicidade da solução.

Cálculo do determinante:

$$\mathrm{L} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \qquad \det(\mathrm{L}) = \mathrm{l}_{11} \cdot l_{22} \cdot l_{33} \qquad det(L) = \prod_{i=1}^{n} l_{ii}$$

$$\mathrm{L}^t = \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix} \qquad det(L^t) = \prod_{i=1}^{n} l_{ii}$$

$\mathrm{A} = \mathrm{L} \cdot L^t$

Pela propriedade dos determinantes, temos:

$\det(\mathrm{A}) = \det(\mathrm{L}) \cdot det(L^t)$

$\det(A) = (\prod_{i=1}^{n} l_{ii})^2$

$\det(A) = (2{\cdot}1 \cdot 3)^2 = 36 \neq 0 \rightarrow$ Solução única

Implementação da função em Python

A função decomposicao_cholesky(), do Programa 3.6, retorna a matriz G, pelo método de Choleky, que vimos anteriormente. Na prática, aplicamos a fatoração de Cholesky para verificar se uma determinada matriz A simétrica é definida positiva. Se o algortimo falhar, isto é, se em alguma etapa tivermos $r \leqslant 0$, o processo será interrompido e, consequentemente, a matriz original não é definida positiva; caso contrário, ao final teremos $A = GG^T$ com o fator conforme descrito no Teorema.

Programa 3.6: Função decomposição de Cholesky

```
1  import numpy as np
2
3  def decomposicao_cholesky(A):
4      """
5      :param A: matriz de coeficientes
6      :return: matriz G e booleano se ocorreu Erro ou nao
7      """
8      n = len(A)
9      G = np.zeros((n,n), dtype=float)
10     for k in range(n):
11         soma = 0
12         for j in range(k):
13             soma = soma + (G[k][j])**2
14         r = A[k][k] - soma
15         if(r<=0):
16             print("A matriz nao eh definida positiva")
17             return [], True
18         G[k][k] = r ** (1 / 2)
19         for i in range(k+1, n):
```

```
20              soma = 0
21              for j in range(0,k):
22                  soma = soma + G[i][j]*G[k][j]
23              G[i][k] = (A[i][k]-soma)/G[k][k]
24      return G, False
```

O Programa 3.7 recebe o retorno da função decomposicao_cholesky() e termina o algoritmo do método, verificando se houve ou não falha na decomposição da matriz A.

Programa 3.7: Função método de Cholesky

```
1  import numpy as np
2
3  def cholesky(A,b):
4      """
5      :param A: matriz de coeficientes
6      :param b: matriz de coeficientes independentes
7      :return: vetor solucao do sistema ou erro
8      """
9      G, erro = decomposicao_cholesky(A)
10     if(erro):
11         return erro
12     else:
13         Gt = np.transpose(G)
14         y = np.linalg.solve(G,b)
15         x = np.linalg.solve(Gt,y)
16         return x
```

3.7 Métodos iterativos

Objetivo: Gerar uma sequência $x^{(k)}$ de vetores a partir de uma solução inicial $x^{(0)}$ que deve convergir para a solução do problema.

$$x^{k+1} = Mx^k + c$$

- Método de Jacobi (ou também chamado de Gauss-Jacobi);
- Método de Gauss-Seidel.

3.7.1 Método de Gauss-Jacobi

Seja o sistema linear $Ax = b$. Se decompusermos a matriz A em três matrizes: inferior, diagonal e superior, $A = I + D + S$, temos:

$Ax = b$

$(I + D + S)x = b$

$Dx = -(I + S)x + b$

$x^{(k+1)} = -D^{-1}(I + S)x^{(k)} + D^{-1}b \Rightarrow -D^{-1}(I + S) = M$ e $D^{-1}b = c$

$x^{(k+1)} = Mx^{(k)} + c$

O método de Gauss-Jacobi é basicamente uma recorrência que tenta aproximar x iterativamente. O índice k não vem de manipulação algébrica — ele aparece quando você transforma o sistema fixo $Ax = b$ em um **método iterativo**.

Em outras palavras:

- $x \rightarrow$ solução exata (desconhecida);
- $x^{(k)} \rightarrow$ aproximação da solução na iteração k;
- $x^{(k+1)} \rightarrow$ nova aproximação calculada a partir de $x^{(k)}$.

Mas temos uma forma mais fácil de montar o sistema iterativo, vejamos. Dado um sistema (3x3) $Ax = b$, em que:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{cases}$$

O método de Gauss-Jacobi consiste em isolar x_1 na 1^a equação, isolar x_2 na 2^a equação, e assim sucessivamente, até isolar x_n na n-ésima equação. Então, para nosso sistema exemplo (3x3), temos:

$$
\begin{aligned}
{x_1}^{(k)} &= \frac{1}{a_{11}}(b_1 - a_{12}{x_2}^{(k-1)} - a_{13}{x_3}^{(k-1)}) \\
{x_2}^{(k)} &= \frac{1}{a_{22}}(b_2 - a_{21}{x_1}^{(k-1)} - a_{23}{x_3}^{(k-1)}) \\
{x_3}^{(k)} &= \frac{1}{a_{33}}(b_3 - a_{31}{x_1}^{(k-1)} - a_{32}{x_2}^{(k-1)})
\end{aligned}
$$

Então, essa é a atualização do método de Gauss-Jacobi. Mas podemos aplicar o método de Gauss-Jacobi em qualquer sistema que ele vai convergir?

Condições de convergência

A convergência é garantida se qualquer das condições for satisfeita:

a) Se o raio espectral[5] $\rho(M) < 1$. O cálculo do raio espectral pode ser muito trabalhoso.

b) Se a matriz A for diagonalmente dominante.[6]

E se o sistema não atender aos critérios de convergência? Então, o sistema pode ou não convergir, depende da proximidade que se está da solução real e das condições do espaço de soluções. Vejamos alguns critérios para testarmos a convergência.

Critérios de parada

a) Distância relativa entre a solução atual e a anterior. Dada uma precisão ε, o vetor $x^{(k)}$ será escolhido como $\bar{x}$, ou seja, solução aproximada da solução exata, se $d_r^{(k)} < \varepsilon$.

[5] O raio espectral $\rho(M)$ é o maior autovalor de M, em módulo.

[6] **Matriz diagonalmente dominante:** Podemos verificar 2 critérios.

Critério das linhas:

$|a_{ii}| > \sum_{j=1;j}^{n} |a_{ij}|$, i = 1, 2, . . . , n

Critério das colunas:

$|a_{jj}| > \sum_{i=1;i}^{n} |a_{ij}|$, j = 1, 2, . . . , n

$$d_r^{(k)} = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\left\|x^{(k)}\right\|_\infty} < \varepsilon$$

Se a distância entre duas soluções (de forma relativa) for menor que a precisão, ou seja, se dermos um passo muito pequeno, então é porque já estamos tão perto da solução que já não conseguimos mais dar passos. Este é um critério mais usado, pois é mais fácil calcular.

b) Resíduo. Dada uma precisão ε, se o resíduo $r^{(K)} < \varepsilon$, então o vetor $x^{(k)}$ será escolhido como $\bar{x}$, ou seja, solução aproximada da solução exata.$r^{(k)} = \left\|Ax^{(k)} - b\right\|_\infty < \varepsilon$.

Nesse caso, é necessário mais operações que no critério anterior, já que envolve multiplicação de matriz e vetor.

c) limite de iterações.

Exemplo de resolução de sistema pelo método de Gauss-Jacobi

Resolva o sistema a seguir, com $\varepsilon = 0.05$ e $x^{(0)} = \frac{b_i}{a_{ii}}$.

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 7 \\ x_1 + 5x_2 + x_3 = -8 \\ 2x_1 + 3x_2 + 10x_3 = 6 \end{cases}$$

$$\varepsilon = 0.05$$

$$x^{(0)} = \frac{b_i}{a_{ii}}$$

Solução:

$$\begin{bmatrix} 10 & 2 & 1 \\ 1 & 5 & 1 \\ 2 & 3 & 10 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -8 \\ 6 \end{bmatrix}$$

1º passo) Verificar condições de convergência:

Linhas:

$10 > 2+1$ ✓

$5 > 1 + 1$ ✓

$10 > 2 + 3$ ✓

Colunas:

$10 > 1 + 2$ ✓

$5 = 2 + 3$ ✗

$10 > 1 + 1$ ✓

Logo, o sistema vai convergir independente da condição inicial.

A matriz A é diagonalmente dominante.

2º passo) Isolar x_1 na 1ª equação, x_2 na 2ª equação, x_3 na 3ª equação.

$$\text{Função de iteração} \begin{cases} x_1^{(k)} = \frac{1}{10}(7 - 2x_2^{(k-1)} - x_3^{(k-1)}) \\ x_2^{(k)} = \frac{1}{5}(-8 - x_1^{(k-1)} - x_3^{(k-1)}) \\ x_3^{(k)} = \frac{1}{10}(6 - 2x_1^{(k-1)} - 3x_2^{(k-1)}) \end{cases}$$

$$\mathrm{x}_0 = \begin{bmatrix} 0,7 \\ -1,6 \\ 0,6 \end{bmatrix}$$

$$\mathrm{x}_1^{(1)} = \tfrac{1}{10}(7 - 2(-1,6) - 0,6) = 0,96$$
$$\mathrm{x}_2^{(1)} = \tfrac{1}{5}(-8 - 0,7 - 0,6) = -1,86$$
$$\mathrm{x}_3^{(1)} = \tfrac{1}{10}(6 - 2(0,7) - 3(-1,6)) = 0,94$$

$$\mathrm{x}^{(1)} = \begin{bmatrix} 0,96 \\ -1,86 \\ 0,94 \end{bmatrix}$$

3º passo) Teste de convergência

$$\mathrm{x}_0 = \begin{bmatrix} 0,7 \\ -1,6 \\ 0,6 \end{bmatrix}, \quad x^{(1)} = \begin{bmatrix} 0,96 \\ -1,86 \\ 0,94 \end{bmatrix} \therefore x^{(1)} - x^{(0)} = \begin{bmatrix} 0,26 \\ 0,26 \\ 0,34 \end{bmatrix}$$

$$d^r = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\left\|x^{(k)}\right\|_\infty} = \frac{0,34}{1,86} = 0,1827 > 0,05$$

$$\mathrm{x}_1^{(2)} = \tfrac{1}{10}(7 - 2(-1,86) - 0,94) = 0,978$$
$$\mathrm{x}_2^{(2)} = \tfrac{1}{5}(-8 - 0,96 - 0,94) = -1,98$$
$$\mathrm{x}_3^{(2)} = \tfrac{1}{10}(6 - 2(0,96) - 3(-1,86)) = 0,966$$

$$\mathrm{x}^{(2)} = \begin{bmatrix} 0,978 \\ -1,98 \\ 0,966 \end{bmatrix}$$

Teste de convergência

$$\mathrm{x}_1 = \begin{bmatrix} 0,96 \\ -1,86 \\ 0,94 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 0,978 \\ -1,98 \\ 0,966 \end{bmatrix} \therefore x^{(2)} - x^{(1)} = \begin{bmatrix} 0,018 \\ 0,12 \\ 0,026 \end{bmatrix}$$

$$d^r = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\left\|x^{(k)}\right\|_\infty} = \frac{0,12}{1,98} = 0,0606 > 0,05$$

$$\begin{aligned} \mathrm{x}_1^{(3)} &= \tfrac{1}{10}(7 - 2(-1,98) - 0,966) = 0,9994 \\ \mathrm{x}_2^{(3)} &= \tfrac{1}{5}(-8 - 0,978 - 0,966) = -1,9888 \\ \mathrm{x}_3^{(3)} &= \tfrac{1}{10}(6 - 2(0,978) - 3(-1,98)) = 0,9984 \end{aligned} \qquad \mathrm{x}^{(3)} = \begin{bmatrix} 0,9994 \\ -1,9888 \\ 0,9984 \end{bmatrix}$$

Teste de convergência

$$\mathrm{x}_2 = \begin{bmatrix} 0,978 \\ -1,98 \\ 0,966 \end{bmatrix}, \quad x^{(3)} = \begin{bmatrix} 0,9994 \\ -1,9888 \\ 0,9984 \end{bmatrix} \therefore x^{(3)} - x^{(2)} = \begin{bmatrix} 0,0214 \\ 0,0088 \\ 0,0324 \end{bmatrix}$$

$$d^r = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\left\|x^{(k)}\right\|_\infty} = \frac{0,0324}{1,9888} = 0,01629 < 0,05$$

Portanto, $d^r < \varepsilon$ ✓

Solução: $x^{(3)} = \begin{bmatrix} 0,9994 \\ -1,9888 \\ 0,9984 \end{bmatrix} \rightarrow$ Solução aproximada do sistema

Implementação do método em Python

A técnica iterativa de Jacobi para resolução de sistemas lineares é mostrada no Algoritmo 4, BURDEN et al. (2005). O algoritmo recebe como entrada a matriz de coeficientes A, o vetor de coeficientes independentes b, a tolerância, *eps* e o número máximo de iterações, *maxiter*. O Programa 3.8, implementa esse algoritmo em Python.

Programa 3.8: Técnica iterativa de Jacobi

```
1  import numpy as np
2
3  def jacobi(A, b, maxiter, eps):
4      """
5      :param A: matriz de coeficientes
6      :param b: vetor de coeficientes independentes
7      :param maxiter: numero maximo de iteracoes
8      :param eps: tolerancia
9      :return: vetor solucao (x) e no. de iteracoes (it)
10     """
11     n = len(A)
12     d = np.divide(b, np.diag(A))
13     x_old = d
14     it = 1
15     x = np.zeros(3, dtype=float)
16     while(it<=maxiter):
17         for i in range(n):
18             s = 0
19             for j in range(n):
20                 if(i!=j):
21                     s = s + A[i][j]*x_old[j]
22             s=-s+b[i]
23             x[i] = s/A[i][i]
24         dif=float(np.divide(np.max(np.abs(np.subtract(
25                                 x,x_old))), np.max(np.abs(x))))
26         if(dif<=eps):
27             return x, it
28         x_old = x.copy()
```

```
29          it+=1
30      return x, it
```

3.7.2 Método de Gauss-Seidel

O método de Gauss-Seidel é uma modificação do método de Gauss-Jacobi de forma a utilizar a componente mais atualizada disponível. Busca acelerar a convergência do método de Jacobi. Sempre que disponível utilizaremos a versão mais atualizada da componente de x. Então, a diferença desse método com o anterior, é que no de Gauss-Seidel vamos trabalhar com uma mescla de duas soluções: a mais atualizada e a anterior. Como montar a função de iteração para um sistema exemplo (3×3):

$$Ax = b$$

$$(I + D + S)x = b$$

$$Dx = b - (I + S)x$$

$$Dx = b - Ix - Sx$$

$$x^{(k+1)} = D^{-1}b - D^{-1}Ix^{(k+1)} - D^{-1}Sx^{(k)}$$

Mas temos uma forma mais fácil de montar o sistema iterativo, vejamos. Dado um sistema (3x3) $Ax = b$, em que:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{cases}$$

Vamos isolar x_1 na 1^a equação, isolar x_2 na 2^a equação, e assim sucessivamente, até isolar x_n na n-ésima equação. Então, para nosso sistema exemplo (3x3), temos,

$\mathrm{x}_1^{(k)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2{}^{(k-1)} - a_{13}x_3{}^{(k-1)}) \rightarrow$ idêntico a Jacobi

$\mathrm{x}_2^{(k)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1{}^{(k)} - a_{23}x_3{}^{(k-1)})) \rightarrow$ mais atualizada (azul) e a anterior (vermelho)

$\mathrm{x}_3^{(k)} = \frac{1}{a_{33}}(b_3 - a_{31}x_1{}^{(k)} - a_{32}x_2{}^{(k)}) \rightarrow$ usa componentes mais atualizadas (azul)

Essa modificação tem por objetivo acelerar a convergência para a solução em relação ao método de Jacobi.

As condições de convergência são as mesmas observadas para o método de Jacobi. A convergência é garantida se qualquer das condições for satisfeita:

a) Se o raio espectral $\rho(M) < 1$;

b) Se a matriz A for diagonalmente dominante;

c) e mais um critério adicional que é o **Critério de Sassenfeld**.

Critério de Sassenfeld

$$\beta_1 = \frac{|a_{12}| + |a_{13}| + \cdots + |a_{1n}|}{a_{11}}$$

$$\beta_j = \frac{|a_{j1}|\beta_1 + |a_{j2}|\beta_2 + \cdots + |a_{jj-1}|\beta_{j-1} + |a_{jj+1}| + \cdots + |a_{jn}|}{a_{jj}}$$

Seja $\beta = max_{1 \leq j \leq n}\{\beta_j\}$ Se $\beta < 1$ o método converge

Além disso, os critérios de parada são os mesmos que já definimos para o método de Gauss-Jacobi.

Exemplo de resolução de sistema pelo método de Gauss-Seidel

Resolva o sistema a seguir, com $\varepsilon = 0.05$ e $x^{(0)} = \frac{b_i}{a_{ii}}$.

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 7 & \varepsilon = 0.05 \\ x_1 + 5x_2 + x_3 = -8 & \mathrm{x}^{(0)} = \frac{b_i}{a_{ii}} \\ 2x_1 + 3x_2 + 10x_3 = 6 & \end{cases}$$

Solução:

1º passo) Verificar convergência pelo critério de Sassenfeld.

$$\beta_1 = \frac{|a_{12}| + |a_{13}|}{|a_{11}|} = \frac{2+1}{10} = 0.3$$

$$\beta_2 = \frac{|a_{21}|\,\beta_1 + |a_{23}|}{|a_{22}|} = \frac{1 \cdot 0.3 + 1}{5} = 0.26$$

$$\beta_3 = \frac{|a_{31}|\,\beta_1 + |a_{32}|\,\beta_2}{|a_{33}|} = \frac{2 \cdot 0.3 + 3 \cdot 0.26}{10} = 0.138$$

$\beta = 0.3 < 1 \rightarrow$ ok! Atende ao critério de Sassenfeld.

2º passo) Montar a função de iteração.

$$\begin{cases} x_1^{(k)} = \frac{1}{10}(7 - 2x_2^{(k-1)} - x_3^{(k-1)}) \\ x_2^{(k)} = \frac{1}{5}(-8 - x_1^{(k)} - x_3^{(k-1)}) \\ x_3^{(k)} = \frac{1}{10}(6 - 2x_1^{(k)} - 3x_2^{(k)}) \end{cases} \qquad \mathrm{x}^{(0)} = \begin{bmatrix} 0,7 \\ -1,6 \\ 0,6 \end{bmatrix}$$

3º passo) Calcular a atualização e testar a convergência.

$$\mathrm{x}_1^{(1)} = \frac{1}{10}(7 - 2(-1,6) - 0,6) = 0,96$$
$$\mathrm{x}_2^{(1)} = \frac{1}{5}(-8 - 0,96 - 0,6) = -1,912 \qquad \mathrm{x}^{(1)} = \begin{bmatrix} 0,96 \\ -1,912 \\ 0,9816 \end{bmatrix}$$
$$\mathrm{x}_3^{(1)} = \frac{1}{10}(6 - 2(0,96) - 3(-1,912)) = 0,9816$$

Teste de convergência

$$\mathrm{x}_0 = \begin{bmatrix} 0,7 \\ -1,6 \\ 0,6 \end{bmatrix}, \quad x^{(1)} = \begin{bmatrix} 0,96 \\ -1,912 \\ 0,9816 \end{bmatrix} \therefore x^{(1)} - x^{(0)} = \begin{bmatrix} 0,26 \\ 0,312 \\ 0,3816 \end{bmatrix}$$

$$d^r = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\left\|x^{(k)}\right\|_\infty} = \frac{0,3816}{1,912} = 0,1996 > 0,05$$

Calcular a próxima atualização e testar a convergência.

$$\mathrm{x}_1^{(2)} = \frac{1}{10}(7 - 2(-1,912) - 0,9816) = 0,98424$$
$$\mathrm{x}_2^{(2)} = \frac{1}{5}(-8 - 0,98424 - 0,9816) = -1,993168 \qquad \mathrm{x}^{(2)} = \begin{bmatrix} 0,98424 \\ -1,993168 \\ 1,0011024 \end{bmatrix}$$
$$\mathrm{x}_3^{(2)} = \frac{1}{10}(6 - 2(0,98424) - 3(-1,993168)) = 1,0011024$$

Teste de convergência

$$\mathrm{x}_1 = \begin{bmatrix} 0,96 \\ -1,912 \\ 0,9816 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 0,98424 \\ -1,993168 \\ 1,0011024 \end{bmatrix} \therefore x^{(2)} - x^{(1)} = \begin{bmatrix} 0,02424 \\ 0,081168 \\ 0,0195024 \end{bmatrix}$$

$$d^r = \frac{\left\|x^{(k)} - x^{(k-1)}\right\|_\infty}{\|x^{(k)}\|_\infty} = \frac{0,081168}{1,993168} = 0,0407 < 0,05$$

Como a distância relativa $d^r < \varepsilon$, a solução será a atual, x_2.

Solução: $\begin{bmatrix} 0,98424 \\ -1,993168 \\ 1,0011024 \end{bmatrix}$

Implementação do método em Python

O método iterativo de Gauss-Seidel para resolução de sistemas lineares é mostrado no Algoritmo 5, BURDEN et al. (2005). O algoritmo recebe como entrada a matriz de coeficientes A, o vetor de coeficientes independentes b, a tolerância, *eps* e o número máximo de iterações, *maxiter*. O código em Python 3, implementando esse algoritmo é mostrado no Programa 3.9.

Programa 3.9: Método iterativo de Gauss-Seidel

```
1  import numpy as np
2
3  def seidel(A, b, maxiter, eps):
4      """
5      :param A: matriz de coeficientes
6      :param b: vetor de coeficientes independentes
7      :param maxiter: numero maximo de iteracoes
8      :param eps: tolerancia
9      :return: vetor solucao (x) e no. de iteracoes (it)
10     """
```

```
11      n = len(A)
12      d = np.divide(b, np.diag(A))
13      x_old = d
14      it = 1
15      x = np.zeros(3, dtype=float)
16      while(it<=maxiter):
17          for i in range(n):
18              s = 0
19              s1=0
20              for j in range(i):
21                  s1=s1+A[i][j]*x[j]
22              s = s-s1
23              s2 = 0
24              for k in range(i+1,n):
25                  s2 = s2 + A[i][k]*x_old[k]
26              s=s-s2+b[i]
27              x[i] = s/A[i][i]
28          dif=float(np.divide(np.max(np.abs(np.subtract(
29                                      x,x_old))), np.max(np.abs(x))))
30          if(dif<=eps):
31              return x, it
32          x_old = x.copy()
33          it+=1
34      return x, it
```

3.8 Exercícios

1. Resolva o sistema linear a seguir pelo método de Eliminação de Gauss.

$$\begin{cases} 2x_1 + 2x_2 + x_3 + x_4 = 7 \\ x_1 - x_2 + 2x_3 - x_4 = 1 \\ 3x_1 + 2x_2 - 3x_3 - 2x_4 = 4 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = 12 \end{cases}$$

2. Analise os sistemas lineares abaixo com relação ao número de soluções, usando o método da Eliminação de Gauss (trabalhe com 3 casas decimais):

a)

$$\begin{cases} 3x_1 - 2x_2 + 5x_3 + x_4 = 7 \\ -6x_1 + 4x_2 - 8x_3 + x_4 = -9 \\ 9x_1 - 6x_2 + 19x_3 + x_4 = 23 \\ 6x_1 - 4x_2 - 6x_3 + 5x_4 = 11 \end{cases}$$

b)

$$\begin{cases} 0.252x_1 + 0.36x_2 + 0.2x_3 = 7 \\ 0.112x_1 + 0.16x_2 + 0.24x_3 = 8 \\ 0.147x_1 + 0.21x_2 + 0.25x_3 = 9 \end{cases}$$

3. Use "Eliminação de Gauss" ou "Decomposição L.U." para resolver o sistema $Ax = b$, onde:

$$A = \begin{bmatrix} 2 & 3 & -1 \\ 3 & 2 & -5 \\ 2 & 4 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 3 \\ -9 \\ -5 \end{bmatrix}$$

- Resolva manualmente usando o método escolhido.
- Calcule o resíduo e verifique a exatidão da solução encontrada.
- Implemente o algoritmo do método escolhido em linguagem Python e compare os resultados

4. Calcule a fatoração LU de A, se possível:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 3 & 2 & 0 \end{pmatrix}$$

5. Usando o método da decomposição de Cholesky, resolva o seguinte sistema de equações lineares:

$$\begin{bmatrix} 1 & 2 & 4 \\ 2 & 8 & 10 \\ 4 & 10 & 26 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ 10 \end{bmatrix}$$

   - Resolva manualmente usando o método de Cholesky.
   - Calcule o resíduo e verifique a exatidão da solução encontrada. Verifique também a unicidade da solução.
   - Implemente o algoritmo do método de Cholesky em linguagem Python e compare os resultados.

6. Considere o seguinte sistema de equações lineares:

$$\begin{bmatrix} -10 & 2 & 2 \\ 1 & 6 & 0 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -8 \\ 7 \\ 0 \end{bmatrix}$$

   (a) Caso haja convergência garantida, resolva o sistema dado usando o método iterativo de Jacobi a partir de $x^{(0)} = (0, 0, 0)^t$ e $\varepsilon = 0.01$.

   (b) Caso o critério de Sassenfeld esteja satisfeito, usando o método iterativo de Gauss-Seidel, resolva o sistema dado a partir de $x^{(0)} = (-2, 1, 0)^t$ e $\varepsilon = 0.01$.

7. Resolva o sistema linear:

$$\begin{cases} 5x_1 + x_2 + x_3 &= 5 \\ 3x_1 + 4x_2 + x_3 &= 6 \\ 3x_1 + 3x_2 + 6x_3 &= 0 \end{cases}$$

pelo método de Gauss-Seidel, se possível. Utilize: $x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ e $\varepsilon = 0.05$

- Verifique se o critério de Sassenfeld é satisfeito e então, se possível, resolva por Gauss-seidel (manualmente).
- Implemente o algoritmo de Gauss-Seidel em Python.
- Compare os resultados do seu programa com os obtidos manualmente.

8. Em cada caso:

   - verifique se o critério de Sassenfeld é satisfeito;
   - resolva por Gauss-Seidel, se possível.

$$\text{A} = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix} ; \text{b} = \begin{pmatrix} 12 \\ 12 \\ 12 \end{pmatrix}$$

$$\text{A} = \begin{pmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{pmatrix} ; \text{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

9. Para cada um dos sistemas lineares a seguir, analise a existência ou não de solução, bem como unicidade de solução, no caso de haver existência.

   a) $3x + 2y = 7$

   b) $\begin{cases} 4m - 2k = 8 \\ 5m + k = 20 \end{cases}$

   c) $\begin{cases} 4x_1 - 2x_2 - 3x_3 = 4 \\ 6x_1 + 3x_2 - 4x_3 = 6 \end{cases}$

d) $6x + 4y - 3z + w = 10$

e) $\begin{cases} 3x_1 - x_2 + 4x_3 = 6 \\ 6x_1 - 2x_2 + 5x_3 = 8 \end{cases}$

f) $\begin{cases} 3x - 2y + z = 8 \\ x - 3y + 4z = 6 \\ 9x + 4y - 5z = 11 \end{cases}$

Capítulo 4

Interpolação e Ajuste de Curvas

4.1 Interpolação

Interpolar uma função $f(x)$ consiste em aproximar essa função por uma outra $g(x)$, escolhida entre uma classe de funções definida *a priori* e que satisfaça algumas propriedades. A função $g(x)$ é então usada em substituição à $f(x)$.

A necessidade de se efetuar essa substituição surge em várias situações, por exemplo:

a) quando são conhecidos somente os valores numéricos da função para um conjunto de pontos e é necessário calcular o valor da função em um ponto não tabelado.

b) quando a função em estudo tem uma expressão tal que operações como a diferenciação e a integração são difíceis (ou impossíveis) de serem realizadas.

Consideraremos que $g(x)$ pertence à classe das funções polinomiais.

Algumas considerações,

i) Existem outras formas de interpolação polinomial como, por exemplo, fórmula de Taylor e a interpolação de polinômios de Hermite, onde as condições de interpolação são outras.

ii) Assim como $g(x)$ foi escolhida entre as funções polinomiais, poderíamos ter escolhido $g(x)$ como função racional, trigonométrica, etc.

Graficamente:

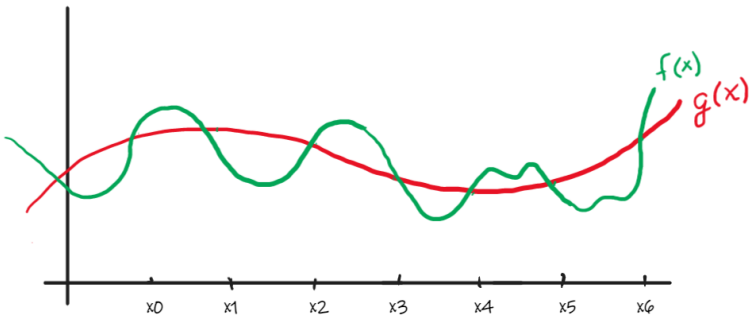


Figura 4.1: Esquema gráfico exemplo de interpolação.

4.1.1 Interpolação polinomial

Dados os pontos $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, $\ldots$, $(x_n, f(x_n))$, portanto $(n+1)$ pontos, queremos aproximar $f(x)$ por um polinômio $P_n(x)$, de grau menor ou igual a n, tal que,

$$f(x_k) = P_n(x_k) \qquad k = 0, 1, 2, \ldots, n$$

Representaremos $P_n(x)$ por:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Portanto, obter $P_n(x)$ significa obter os coeficientes $a_0, a_1, \ldots, a_n$

Da condição $P_n(x_k) = f(x_k)\ \forall\ k = 0, 1, 2, \ldots, n$ montamos o seguinte sistema linear,

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 + \cdots + a_n x_0^n &= f(x_0) \\ a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \cdots + a_n x_1^n &= f(x_1) \\ a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \cdots + a_n x_2^n &= f(x_2) \\ \qquad\qquad \vdots & \qquad\quad \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + a_3 x_n^3 + \cdots + a_n x_n^n &= f(x_n) \end{cases}$$

com $n+1$ equações e $n+1$ variáveis $a_0, a_1, a_2, \ldots, a_n$.

A matriz A dos coeficientes é uma matriz de Vandermonde e, portanto, desde que $x_0, x_1, \ldots, x_n$ sejam pontos distintos, temos $det(A) \neq 0$. Logo, o sistema linear admite solução única.

Formas de se obter $P_n(x)$:

- resolução de sistema linear;
- forma de Lagrange;
- forma de Newton (e de Gregory-Newton).

Teorema 1:

Existe um único polinômio $P_n(x)$ de grau $\leq n$ tal que $P_n(x_k) = f(x_k)$, $k = 0, 1, 2, \ldots, n$ desde que $x_k \neq x_j$, $j \neq k$

A escolha desse método depende de condições de estabilidade do sistema linear, tempo computacional, etc.

4.1.2 Resolução do sistema linear

Exemplo 1

Encontrar o polinômio de grau ≤ 2 que interpola os pontos da tabela:

x	-1	0	2
y	4	1	-1

Temos que $P_n(x) = a_0 + a_1x + a_2x^2$.

$x_0 = -1 \quad x_1 = 0 \quad x_2 = 2$

$P_2(x_0) = f(x_0) \Leftrightarrow a_0 - a_1 + a_2 = 4$

$P_2(x_1) = f(x_1) \Leftrightarrow a_0 = 1$

$P_2(x_2) = f(x_2) \Leftrightarrow a_0 + 2a_1 + 4a_2 = -1$

Resolvendo o sistema linear, obtemos:

$a_0 = 1$

$$\begin{cases} -a_1 + a_2 = 3 \\ 2a_1 + 4a_2 = -2 \end{cases} \rightarrow \begin{array}{ll} a_2 = 3 + a_1 & \\ 2a_1 + 4(3 + a_1) = -2 & a_2 = 3 - \frac{7}{3} \\ 2a_1 + 12 + 4a_1 = -2 & a_2 = \frac{9-7}{3} \\ 6a_1 = -14 & a_2 = \frac{2}{3} \\ a_1 = -\frac{7}{3} & \end{array}$$

$\boxed{a_0 = 1}$ $\boxed{a_1 = -\frac{7}{3}}$ $\boxed{a_2 = \frac{2}{3}}$

Então, $P_2(x) = 1 - \frac{7}{3}x + \frac{2}{3}x^2$ é o polinômio que interpola $f(x)$ em $x_0 = -1$, $x_1 = 0$ e $x_2 = 2$.

Suponha que precisamos calcular $f(x)$ em $x = 1$:

$$P_2(1) = 1 - \frac{7}{3} \cdot 1 + \frac{2}{3} \cdot 1^2$$

$P_2(x) = -\frac{2}{3}$

4.1.3 Interpolação linear

A Interpolação Linear é um caso particular de interpolação. O polinômio interpolador, nesse caso, de grau $n \leq 1$, é uma reta.

Sejam 2 pontos (x_0, y_0) e (x_1, y_1) , com $x_0 \neq x_1$ de uma função $y = f(x)$. Para obter uma aproximação de $f(z)$, $z \in (x_0, x_1)$, faz-se:

$$f(x) \approx P_1(x) = a_0 + a_1 x$$

onde $P_1(x)$ é um polinômio interpolador de grau 1.

Impondo que o polinômio passe pelos 2 pontos-base, tem-se:

$$\begin{array}{rcl} P_1(x_0) & = & y_0 \\ P_1(x_1) & = & y_1 \end{array} \rightarrow \begin{cases} a_0 + a_1 x_0 = y_0 \\ a_1 + a_1 x_1 = y_1 \end{cases} \leftrightarrow \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

Aplicando uma operação l-elementar para transformar esse sistema em um triangular equivalente, obtém-se:

(aplicamos $l_2 = l_2 - l_1$)

$$\begin{bmatrix} 1 & x_0 \\ 0 & x_1 - x_0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 - y_0 \end{bmatrix}$$

$$\mathrm{a}_0 + a_1 x_0 = y_0 \quad \Rightarrow \quad \mathrm{a}_0 = y_0 - a_1 x_0$$

$$\mathrm{a}_1(x_1 - x_0) = y_1 - y_0 \quad \Rightarrow \quad a_1 = \tfrac{y_1 - y_0}{x_1 - x_0}$$

Portanto, o polinômio interpolador torna-se:

$$P_1(x) = (y_0 - a_1 x_0) + a_1 x = y_0 + a_1(x - x_0) \text{ e}$$

$$\boxed{\mathrm{P}_1(x) = y_0 + \tfrac{y_1 - y_0}{x_1 - x_0}(x - x_0)} \tag{4.1}$$

Exemplo: Calcular $P_1(0.2)$ e $P_1(0.3)$ a partir dos pontos da tabela:

i	0	1
x_i	0.1	0.6
y_i	1.221	3.320

Usando 4.1 temos:

$P_1(0.2) = 1.221 + \frac{3.320 - 1.221}{0.6 - 0.1}(0.2 - 0.1) \sim 1.641$ e

$P_1(0.3) = 1.221 + \frac{3.320 - 1.221}{0.6 - 0.1}(0.3 - 0.1) \sim 2.061$

4.1.4 Método de Lagrange

Sejam,

$$P_n(x) = \sum_{k=0}^{n} y_k L_k(x), \text{ onde}$$

$$L_k(x) = \frac{\prod_{j=0 j \neq k}^{n}(x - x_j)}{\prod_{j=0 j \neq k}^{n}(x_k - x_j)}$$

Dado um tabelamento:

x	x_0	x_1	x_2	x_3
y	y_0	y_1	y_2	y_3

$\rightarrow$ temos 4 pontos

Vamos calcular um polinômio de grau 3 que interpola esses pontos.

$P_3(x) =?$

$P_3(x) = y_0L_0(x) + y_1L_1(x) + y_2L_2(x) + y_3L_3(x)$

$$L_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)}$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)}$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)}$$

$$L_3(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}$$

Exemplo:

x	-1	0	2
y	4	1	-1

Temos 3 pontos; queremos calcular um polinômio interpolador de grau 2.

a) $P_2(x) =?$

b) $P_2(1) =?$

$$P_2(x) =?$$

$$P_2(x) = y_0L_0(x) + y_1L_1(x) + y_2L_2(x)$$

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 0)(x - 2)}{(-1 - 0)(-1 - 2)} = \frac{x^2 - 2x}{3}$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x+1)(x-2)}{(0+1)(0-2)} = \frac{x^2-x-2}{-2}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x+1)(x-0)}{(2+1)(2-0)} = \frac{x^2+x}{6}$$

$$P_2(x) = 4\frac{x^2-2x}{3} + \frac{x^2-x-2}{-2} - \frac{x^2-x}{6}$$

$$P_2(x) = \frac{4x^2-8x}{3} - \frac{x^2+x+2}{2} - \frac{x^2-x}{6}$$

$$P_2(x) = \frac{8x^2-16x-3x^2+3x+6-x^2-x}{6}$$

$$P_2(x) = \frac{4x^2-14x+6}{6}$$

$\boxed{\mathrm{P}_2(x) = \frac{2}{3}x^2 - \frac{7}{3}x + 1}$ → polinômio interpolador de grau 2

$\boxed{\mathrm{P}_2(1) = \frac{2}{3}1^2 - \frac{7}{3}1 + 1 = -\frac{2}{3}}$

Implementação do método em Python

Programa 4.1: Função de interpolação por polinômios de Lagrange

```
1 def lagrange(x,y,xx):
2     """
3     :param x: lista de xi
4     :param y: lista de yi
5     :param xx: valor a ser interpolado
6     :return: valor interpolado e booleano se houve erro
7     """
8     n = len(x)  # numero de amostras
9     if len(y) != n:
```

```
10          print("x e y devem ter o mesmo tamanho")
11          return 0, True
12      soma = 0  # Inicializa o somatorio com 0
13      for i in range(n):  # Calcula o somatorio
14          produto = 1
15          d = 1
16          for j in range(n):  # Calcula o produtorio
17              if (i != j):  # Se i diferente de j
18                  produto = produto * (xx - x[j])
19                  d = d * (x[i] - x[j])
20          soma = soma +y[i] *(produto/d)
21      return soma, False
```

4.1.5 Método de Newton - Diferenças divididas

Seja:

$$P_n(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + \cdots + d_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

onde d_i , $0 \leqslant i \leqslant n$, é o operador de diferenças divididas de ordem i.

Operador de diferenças divididas

$$d_0 = f[x_0] = y_0 \quad \text{ordem 0}$$
$$d_1 = f[x_1, x_0] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} \quad \text{ordem 1}$$
$$d_2 = f[x_2, x_1, x_0] = \frac{f[x_1, x_2] - f[x_1, x_0]}{x_2 - x_0} \quad \text{ordem 2}$$
$$d_n = f[x_n, x_{n-1}, \ldots x_0] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_{n-1}, x_{n-2}, \ldots, x_0]}{x_n - x_0} \quad \text{ordem n}$$

onde a função com colchetes corresponde a diferenças divididas finitas. Por exemplo, a primeira diferença finita é representada em geral por

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

Podemos reparar que $f[x_j, x_i]$ é igual a $f[x_i, x_j]$:

$$\begin{aligned}
\frac{f(x_j) - f(x_i)}{x_j - x_i} &= f[x_j, x_i] \\
\frac{f(x_i) - f(x_j)}{x_j - x_i} &= -f[x_j, x_i] \\
\frac{f(x_i) - f(x_j)}{x_i - x_j} &= f[x_j, x_i] \\
f[x_i, x_j] &= \frac{f(x_i) - f(x_j)}{x_i - x_j} \\
f[x_j, x_i] &= f[x_i, x_j]
\end{aligned}$$

A *segunda diferença dividida finita*, que representa a diferença das duas primeiras diferenças divididas, é expressa em geral por

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

Dado um tabelamento:

x	x_0	x_1	x_2
y	y_0	y_1	y_2

x	Ordem 0	Ordem 1	Ordem 2
x_0	$y_0 = f[x_0]$	$\frac{f[x_1]-f[x_0]}{x_1-x_0} = f[x_1,x_0] = f[x_0,x_1]$	
x_1	$y_1 = f[x_1]$		$\frac{f[x_1,x_2]-f[x_1,x_0]}{x_2-x_0} = f[x_2,x_1,x_0]$
x_2	$y_2 = f[x_2]$	$\frac{f[x_2]-f[x_1]}{x_2-x_1} = f[x_2,x_1] = f[x_1,x_2]$	

$$P_2(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1)$$

Exemplo:

x	1	0	2
y	4	1	-1

Queremos calcular:

a) $P_2(x) = ?$

b) $P_2(1) = ?$

$\mathrm{P}_2(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1)$
$P_2(x) = d_0 + d_1(x + 1) + d_2(x + 1)(x - 0)$

x	Ordem 0	Ordem 1	Ordem 2
$x_0 = -1$	$y_0 = f[x_0] = 4$ $\boxed{d_0}$	$\frac{f[x_1]-f[x_0]}{x_1-x_0} = f[x_1,x_0] = \frac{1-4}{0-1} = 3$ $\boxed{d_1}$	
$x_1 = 0$	$y_1 = f[x_1] = 1$		$\frac{f[x_1,x_2]-f[x_1,x_0]}{x_2-x_0} = \frac{-1-3}{2-1} = -4$ $\boxed{d_2}$
$x_2 = 2$	$y_2 = f[x_2] = -1$	$\frac{f[x_2]-f[x_1]}{x_2-x_1} = f[x_1,x_2] = \frac{-1-1}{2-0} = -1$	

$$\boxed{P_2(x) = 4 + 3(x+1) + (-4)(x+1)(x)}$$

a)

$P_2(x) = 4 + 3x + 3 - 4x^2 - 4x$

$$\boxed{P_2(x) = -4x^2 - x + 7}$$

b)

$P_2(1) = 4 + 3(1+1) + (-4)(1+1)(1)$
$P_2(1) = 4 + 6 - 8$
$P_2(1) = 2$

Implementação do método em Python

Programa 4.2: Função de interpolação pelo método de Newton

```
1  import numpy as np
2  import pandas as pd
3
4  def metodo_newton(x, y, xx, imprime_tabela=True):
5      """
6      :param x: lista de xi
7      :param y: lista de yi
8      :param xx: valor a ser interpolado
9      :param imprime_tabela: booleano para indicar
10         se imprime ou nao a tabela de diferencas divididas
11     :return: valor interpolado
12     """
13     n = len(x)
14     difd = np.zeros([n, n])
15     # difd[:,0]=y[:]
16
17     for i in range(n):
18         difd[i][0] = y[i]
19     for j in range(1, n):
20         for i in range(n - j):
21             difd[i, j] = (difd[i + 1][j - 1] -
22                 difd[i][j - 1]) / (x[i + j] - x[i])
23
24     # imprimindo a tabela de diferen as divididas
25     if (imprime_tabela):
26         difd_table = pd.DataFrame(difd)
27         print(difd_table)
28
29     # calculo do termo interpolado (f(z))
30     xterm = 1
```

```
31      yint = difd[0][0]
32      for order in range(1, n):
33          xterm = xterm * (xx - x[order - 1])
34          yint = yint + difd[0][order] * xterm
35
36      if(imprime_tabela==False):
37          difd_table = []
38
39      return yint,difd_table
```

4.1.6 Método de Gregory-Newton

Polinômios de Gregory-Newton

Caso particular dos polinômios de Newton para pontos igualmente espaçados.

x	x_0	x_1	x_2
y	y_0	y_1	y_2

$$x_{i+1} - x_i = h$$

Operador diferença finita ascendente

ordem 0: $\Delta^0 y_i = y_i$
ordem 1: $\Delta^1 y_i = \Delta^0 y_{i+1} - \Delta^0 y_i$
ordem 2: $\Delta^2 y_i = \Delta^1 y_{i+1} - \Delta^1 y_i$
$\vdots$ $\vdots$
ordem n: $\Delta^n y_i = \Delta^{n-1} y_{i+1} - \Delta^{n-1} y_i$

Em forma de tabela, temos:

x	**Ordem 0**	**Ordem 1**	**Ordem 2**
x_0	$y_0 = \Delta^0 y_0$ ↘↗	$\Delta^1 y_0 = \Delta^0 y_1 - \Delta^0 y_0$ ↘	
x_1	$y_1 = \Delta^0 y_1$		$\Delta^2 y_0 = \Delta^1 y_1 - \Delta^1 y_0$
x_2	$y_2 = \Delta^0 y_2$ ↗↘	$\Delta^1 y_1 = \Delta^0 y_2 - \Delta^0 y_1$ ↗	

Cálculos preliminares

$\boxed{x_1 - x_0 = h}$ $\boxed{x_1 = x_0 + h}$
$\boxed{x_2 - x_0 = 2h}$

Fazendo: $u = (x - x_0)/h$
$uh = x - x_0 \qquad x - x_0 = uh$
$x - x_1 = x - (x_0 + h)$
$x - x_1 = x - x_0 - h$
$x - x_1 = uh - h$
$\boxed{x - x_1 = h(u - 1)}$

Dado um tabelamento:

x	x_0	x_1	x_2
y	y_0	y_1	y_2

Relembrando Newton, temos que $P_2(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1)$

	Ordem 0	Ordem 1	Ordem 2
x_0	y_0	$\frac{y_1 - y_0}{x_1 - x_0}$	$\frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0} = \frac{\frac{y_2 - y_1}{h} - \frac{y_1 - y_0}{h}}{2h}$
x_1	y_1		
x_2	y_2	$\frac{y_2 - y_1}{x_2 - x_1}$	

$\mathrm{P}_2(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1)$

$\mathrm{P}_2(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x_1 - x_0) + d_2(x - x_0)(x - x_1)$

$y_1 - y_0 = \Delta^1 y_0$
$x_1 - x_0 = h$

$\mathrm{P}_2(x) = y_0 + \frac{\Delta^1 y_0}{h}(x - x_0) + d_2(x - x_0)(x - x_1)$

$\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 u + d_2(x - x_0)(x - x_1)$

$\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 u + \frac{\Delta^2 y_0}{2h^2}(x - x_0)(x - x_1)$

$\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 u + \frac{\Delta^2 y_0}{2h^2}[(uh)(uh - h)]$

$\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 \cdot u + \frac{\Delta^2 y_0}{2h^2}[u^2 h^2 - uh^2]$

$\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 \cdot u + \frac{\Delta^2 y_0}{2}(u^2 - u)$
$\boxed{\mathrm{P}_2(x) = y_0 + \Delta^1 y_0 u + \frac{\Delta^2 y_0}{2} u(u - 1)}$

Trabalhando d_2 :

$\mathrm{d}_2 = \frac{\frac{y_2 - y_1}{h} - \frac{y_1 - y_0}{h}}{2h}$

$\mathrm{d}_2 = \frac{\frac{1}{h}[(y_2 - y_1) - (y_1 - y_0)]}{2h}$

$\mathrm{d}_2 = \frac{\frac{1}{h}\left(\Delta^1 y_1 - \Delta^1 y_0\right)}{2h}$

$\mathrm{d}_2 = \frac{\frac{1}{h} \cdot \Delta^2 y_0}{2h}$

$\mathrm{d}_2 = \frac{\Delta^2 y_0}{h} \cdot \frac{1}{2h} = \frac{\Delta^2 y_0}{2h^2}$

Deduzimos uma fórmula para um polinômio de Gregory-Newton de grau 2. Para um polinômio de qualquer outra ordem temos que fazer todos esses cálculos e deduções novamente. Para facilitar, apresentamos uma fórmula para o caso geral, Equação.4.1.6

Caso geral:

$$P_n(x) = y_0 + \sum_{i=1}^{n} \left(\frac{\Delta^i y_0}{i!} \prod_{j=0}^{i-1} (u - j) \right)$$

Exemplo: Calcule, por Gregory-Newton, o polinômio interpolador para os seguintes pontos:

x	-1	0	1
y	4	1	$-\frac{2}{3}$

Como os pontos estão igualmente espaçados podemos usar Gregory-Newton. Temos 3 pontos, então calculemos um polinômio de grau 2:

$$P_2(x) = y_0 + \frac{\Delta^1 y_0}{1!} u + \frac{\Delta^2 y_0}{2!} u(u-1)$$

x	O(0)	O(1)	O(2)
-1	4	-3	$\frac{4}{3}$
0	1	$-\frac{5}{3}$	
1	$-\frac{2}{3}$		

$u = \frac{x - x_0}{h}$
$u = \frac{x - x_0}{x_1 - x_0}$
$u = \frac{x - (-1)}{0 - (-1)}$
$u = \frac{x+1}{1}$
$\boxed{u = x + 1}$

$$\begin{aligned}
\mathrm{P}_2(x) &= y_0 + \Delta^1 y_0 u + \tfrac{\Delta^2 y_0}{2} u(u-1) \\
P_2(x) &= y_0 + \Delta^1 y_0 (x+1) + \tfrac{\Delta^2 y_0}{2} (x+1)x \\
P_2(x) &= 4 + (-3)(x+1) + \tfrac{(4/3)}{2} (x+1)x \\
P_2(x) &= 4 - 3(x+1) + \tfrac{2}{3}(x+1)x \\
P_2(x) &= 4 - 3x - 3 + \tfrac{2}{3}(x^2 + x) \\
P_2(x) &= 1 - 3x + \tfrac{2}{3}x^2 + \tfrac{2x}{3} \\
P_2(x) &= 1 - \tfrac{7}{3}x + \tfrac{2}{3}x^2 \\
P_2(0.6) &= -0.16
\end{aligned}$$

Implementação do Método em Python

Programa 4.3: Função do método de Gregory-Newton

```
1  import numpy as np
2  import pandas as pd
3  from scipy.special import factorial
4
5  def metodo_gregory_newton(x,y,xx, imprime_tabela=True):
6      """
7      :param x: abscissas
8      :param y: ordenadas
9      :param xx: valor a interpolar
10     :param imprime_tabela: imprimir (ou nao)
11         a tabela de interpolacao
12     :return: valor interpolado
```

```
13      """
14      n = len(x)
15      diff = np.zeros([n, n])
16      for i in range(n):
17          diff[i][0] = y[i]
18      #construcao para diferencas finitas
19      for k in range(n):
20          for i in range(n-1,k,-1):
21              diff[i][k+1]=diff[i][k]-diff[i-1][k]
22      # imprimindo a tabela de diferencas finitas
23      if (imprime_tabela):
24          difd_table = pd.DataFrame(diff)
25          print(difd_table)
26
27      #avaliacao do polinomio pelo processo de Horner
28      u = (xx - x[0])/(x[1]-x[0]) #(x-x0)/h
29      r = diff[0,0]
30      for i in range(1,n):
31          sum = diff[i][i]/factorial(i)
32          for j in range(0, i):
33              sum = sum * (u-j)
34          r = r + sum
35
36      if(imprime_tabela==False):
37          difd_table = []
38
39      return r, difd_table
```

4.2 Exercícios

1. Calcular a partir da tabela de $y = sen(x)$

x	y
0.3	0.2955
0.4	0.3894
0.5	0.4794

os valores de:

a) $P_1(0.33)$

b) $P_2(0.33)$

c) $P_1(0.38)$

d) $P_2(0.38)$

2. Considere a função

$$f(x) = \frac{x}{(x+1)}$$

tabelada nos pontos, como segue:

x_i	0	1	2
$f(x_i)$	0	1/2	2/3

Determine o polinômio interpolador pela fórmula de Gregory-Newton, avalie $f(1.3)$. Resolva manualmente e computacionalmente e compare os resultados. Calcule o erro comparando com o resultado exato.

3. Seja a tabela:

x	y
1.0	0.8415
1.3	1.2526
1.7	1.6858
2.0	1.8186

Usando Lagrange, calcule $L_1(1.1)$, $L_2(1.1)$, $L_3(1.2)$. Implemente o algoritmo de Lagrange em Python, calcule novamente, utilizando o programa e compare os resultados.

4. Considere a tabela:

x	y
2.0	0.9803
2.2	1.1695
2.4	1.3563
2.5	1.4488
2.7	1.6321
2.9	1.8131

Usando a fórmula de Newton, calcular $P_1(2.1)$, $P_2(2.1)$ e $P_3(2.1)$. Compare os três resultados obtidos com o valor exato $f(2.1) = 1.0752$.

Implemente o algoritmo de Newton, calcule novamente $P_1(2.1)$, $P_2(2.1)$ e $P_3(2.1)$ utilizando o programa, e compare os resultados.

5. Seja a tabela:

x	y
2.1	0.3693
2.2	0.5137
2.3	0.6732
2.4	0.8424

Construa a tabela de diferenças finitas ascendentes e a de diferenças divididas. Calcule, por Gregory-Newton, $P_1(2.15)$, $P_2(2.15)$ e $P_3(2.15)$ e compare com o resultado exato $f(2.15) = 0.43932$. Implemente o algoritmo do polinômio de Gregory-Newton em Python, resolva P(2.15) utilizando o programa e compare os resultados.

4.3 Ajuste de Curvas

4.3.1 Introdução

Vimos que uma forma de se trabalhar com uma função definida por uma tabela de valores é a interpolação polinomial. Qual o conceito embutido na Interpolação? Temos (n+1) pontos distintos, $x_0, x_1, \ldots, x_n$, chamados de nós de interpolação e valores de $f(x)$ nesses pontos: $f(x_0), f(x_1), \ldots, f(x_n)$. Onde interpolação de $f(x)$ consiste em se obter uma função $g(x)$ tal que:

$$\begin{aligned} g(x_0) &= f(x_0) \\ g(x_1) &= f(x_1) \\ \vdots &\quad\ \vdots \\ g(x_n) &= f(x_n) \end{aligned}$$

Contudo, a interpolação não é aconselhável quando:

1. é preciso obter um valor aproximado da função em algum ponto fora do intervalo de tabelamento, ou seja, quando se quer extrapolar.
2. os valores tabelados são resultados de algum experimento físico ou de alguma pesquisa, porque, nesses casos, esses valores poderão conter erros inerentes que, em geral, não são previsíveis.

Surge, então, a necessidade de se ajustar a essas funções tabeladas uma função que seja uma "boa aproximação" para os valores tabelados e que nos permita "extrapolar" com certa margem de segurança.

Caso discreto

O problema do ajuste de curvas no caso em que temos uma tabela de pontos $(x_1, f(x_1))$, $(x_2, f(x_2))$, ..., $(x_n, f(x_n))$ com x_1, x_2, ..., x_n pertencentes a um intervalo $[a, b]$, consiste em: "escolhidas" n funções $g_1(x)$, $g_2(x)$, ..., $g_n(x)$, contínuas em $[a, b]$, obter n constantes α_1, α_2, ..., α_n tais que a função $\phi(x) = \alpha_1 g_1(x) + \alpha_2 g_2(x) + \cdots + \alpha_n g_n(x)$ se aproxime ao máximo de $f(x)$.

Dizemos que esse é um modelo matemático linear porque os coeficientes a determinar, α_i aparecem linearmente, embora as funções $g_i(x)$ possam ser funções não lineares de x, como, por exemplo, $g_1(x) = e^x$, $g_2(x) = (1 + x^2)$, etc.

Mas, como escolher as funções contínuas $g_1(x), \ldots g_n(x)$?

A escolha das funções pode ser feita observando o gráfico dos pontos tabelados ou baseando-se em fundamentos teóricos do experimento que nos forneceu a tabela.

Portanto, um primeiro passo deve ser colocar esses pontos num gráfico cartesiano. O gráfico resultante é chamado *diagrama de dispersão*. Através desse gráfico é possível visualizar a curva que melhor se ajusta aos dados. Vejamos um exemplo.

Exemplo (a) Seja a tabela:

x	-1.0	-0.75	-0.6	-0.5	-0.3	0	0.2	0.4	0.5	0.7	1.0
f(x)	2.05	1.153	0.45	0.4	0.5	0	0.2	0.6	0.512	1.2	2.05

O diagrama de dispersão é apresentado na Figura 4.2.

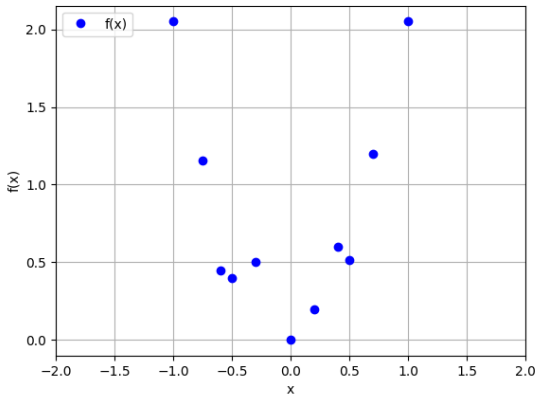


Figura 4.2: Gráfico de dispersão para pontos do Exemplo (a).

Nesse caso, é natural escolhermos apenas a função $g_1(x) = x^2$ e procurarmos então $\phi(x) = \alpha x^2$ (equação geral de uma parábola passando pela origem).

Exemplo (b) Se considerarmos uma experiência onde foram medidos vários valores de corrente elétrica que passa por uma resistência submetida a várias tensões, colocando os valores correspondentes de corrente e tensão em um gráfico, poderemos ter a Figura 4.3.

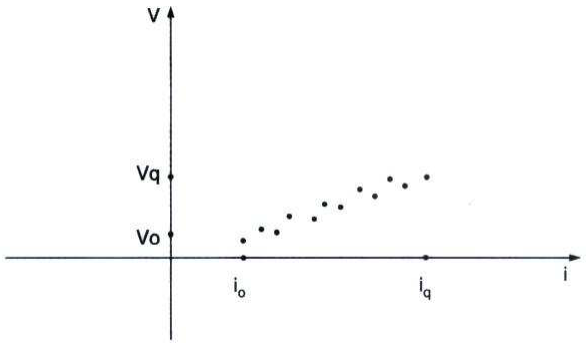


Figura 4.3: Gráfico de dispersão para pontos do Exemplo (b).

Nesse caso, existe uma fundamentação teórica relacionando a corrente com a tensão $V = RI$, isto é, V é uma função linear de I.

Assim, $g_1(I) = I$ e $\phi(I) = \alpha g_1(I)$

Qual parábola com equação αx^2 se ajusta melhor ao diagrama do Exemplo a) e qual reta, passando pela origem, melhor se ajusta ao diagrama do Exemplo b)?

No caso geral, escolhidas as funções $g_1(x)$, $g_2(x)$, ..., $g_n(x)$ temos de estabelecer o conceito de proximidade entre as funções $\phi(x)$ e $f(x)$ para obter as constantes, $\alpha_1, \alpha_2, \ldots, \alpha_n$. Uma ideia é impor que o desvio $(f(x_i) - \phi(x_i))$ seja mínimo para $i = 1, 2, \ldots, m$.

Caso contínuo

No caso contínuo, o problema de ajuste de curvas consiste em: dada uma função $f(x)$ contínua num intervalo $[a, b]$ e escolhidas as funções $g_1(x)$, $g_2(x)$, ... , $g_n(x)$ todas contínuas em $[a, b]$, determinar n constantes α_1, α_2, ..., α_n de modo que a função $\phi(x) = \alpha_1 g_1(x) + \alpha_2 g_2(x) + \cdots + \alpha_n g_n(x)$ se aproxime "ao máximo" de $f(x)$ no intervalo $[a, b]$.

Novamente o problema é: o que significa "ficar mais próxima"? Uma ideia é escolher a função $\phi(x)$ de tal forma que o módulo da área sob a curva $\phi(x) - f(x)$ seja mínimo.

4.3.2 Método dos Quadrados Mínimos

Sejam dados os pontos $(x_1, f(x_1))$,$(x_2, f(x_2))$, ..., $(x_m, f(x_m))$ com x_1, x_2, ..., x_n e as funções $g_1(x)$, $g_2(x)$, ..., $g_n(x)$ escolhidas de alguma forma. Consideraremos que o número de pontos m, tabelados, é sempre maior ou igual a n, o número de funções escolhidas ou o número de coeficientes α_i a se determinar.

Nosso objetivo é encontrar os coeficientes $\alpha_1, \alpha_2, \ldots, \alpha_n$ tais que a função $\phi(x) = \alpha_1 g_1(x) + \alpha_2 g_2(x) + \cdots + \alpha_n g_n(x)$ se aproxime "ao máximo" de $f(x)$.

Seja $d_k = f(x_k) - \phi(x_k)$ o desvio em x_k.

O método dos quadrados mínimos consiste em escolher os $\alpha_j's$ de tal forma que a soma dos quadrados dos desvios seja mínima.

Método dos Quadrados Mínimos Linear

O Método dos Quadrados Mínimos consiste em estabelecer a equação da reta (no caso linear) que melhor se ajusta aos pontos tabelados. Então, vamos medir o erro entre um

ponto qualquer e a reta.

Qual seria o menor erro possível? É a reta que tivesse a menor diferença entre todos os pontos. Então podemos escrever:

$$\text{Erro} = \sum_{i=1}^{n} [y_i - g(x_i)]^2 \qquad \text{onde } n \text{ é a quantidade de pontos}$$

Por onde se deve traçar a reta de modo a obter o menor valor do desvio D?

O método dos quadrados mínimos consiste em encontrar uma estimativa da reta,

$$u = b_0 + b_1 x$$

de modo a produzir o menor valor possível do desvio,

$$D(b_0, b_1) = \sum_{i=1}^{n} (y_i - u_i)^2 = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \tag{4.2}$$

A equação que melhor se ajusta aos pontos vai ter o menor erro possível (*MinErro*).

O MinErro vai ocorrer onde? No ponto de mínimo da função erro. Então, para calcular o ponto de mínimo basta calcular as derivadas parciais. Mas, no caso, nossa função erro é uma função modular e esse tipo de função tem um problema, como podemos observar na Figura 4.4.

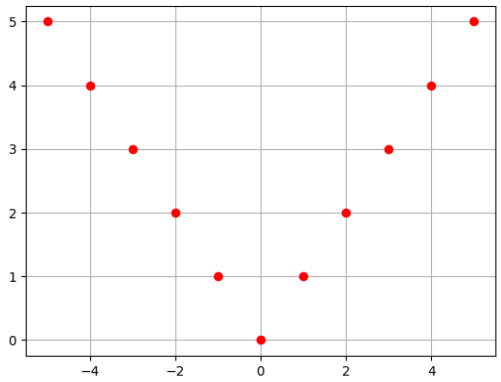


Figura 4.4: Exemplo de uma função modular.

Na quina, o ponto de mínimo não é diferenciável. Então, ao invés de usarmos a função modular, consideraremos a função ao quadrado, pois, se o erro é mínimo, então o erro ao quadrado também será mínimo. No ponto de mínimo teremos as derivadas parciais nulas.

$$\text{Erro} = \sum_{i=1}^{n} \left[y_i - g(x_i)\right]^2$$

$$\text{Erro} = \sum_{i=1}^{n} \left(y_i - ax_i - b\right)^2$$

$$\frac{\partial E}{\partial a} = 0, \qquad \frac{\partial E}{\partial b} = 0$$

$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a} \sum \left(y_i - ax_i - b\right)^2 = 2 \sum \left(y_i - ax_i - b\right)(-x_i) = 0 \rightarrow \text{regra da cadeia}$$

$$\frac{\partial E}{\partial a} = \sum x_i y_i - \sum a x_i^2 - b \sum x_i = 0$$

$$\boxed{a \sum x_i^2 + b \sum x_i = \sum x_i y_i} \quad \text{Ⓐ}$$

$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b} \sum \left(y_i - ax_i - b\right)^2 = 2 \sum \left(y_i - ax_i - b\right)(-1) = 0 \rightarrow \text{regra da cadeia}$$

$$\sum y_i - a \sum x_i - \sum b = 0$$

$$\sum y_i - a \sum x_i - nb = 0$$

$$\boxed{a \sum x_i + nb = \sum y_i} \quad \text{Ⓑ}$$

Portanto, para obter os valores de a e b em relação a x e y, devemos resolver o sistema linear:

$$\begin{cases} a \sum x^2 + b \sum x & = \sum xy \\ a \sum x + nb & = \sum y \end{cases}$$

Isolando ⓑ em Ⓑ, temos:

$\boxed{b = \frac{\sum y - a \sum x}{n}}$ Ⓒ

Substituindo Ⓒ em Ⓐ, temos:

$$a \sum x^2 + \left[\frac{\sum y - a \sum x}{n}\right] \sum x = \sum xy$$

$$a \sum x^2 + \frac{\sum x \sum y}{n} - \frac{a(\sum x)^2}{n} = \sum xy$$

$$n \cdot a \cdot \sum x^2 + \sum x \sum y - a \left(\sum x\right)^2 = n \sum xy$$

$$a \left(n \sum x^2 - \left(\sum x\right)^2\right) = n \sum xy - \sum x \sum y$$

$\boxed{\left(a = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}\right)}$ Ⓓ

Isolando ⓐ em Ⓑ, temos:

$$\boxed{a = \frac{\sum y - nb}{\sum x}} \quad \text{Ⓔ}$$

Substituindo Ⓔ em Ⓐ:

$$\left[\frac{\sum y - nb}{\sum x}\right] \sum x^2 + b \sum x = \sum xy$$

$$\frac{\sum y \cdot \sum x^2}{\sum x} - \frac{nb \sum x^2}{\sum x} + b \sum x = \sum xy \quad \cdot (\sum x)$$

$$\sum y \cdot \sum x^2 - nb \sum x^2 + b \left(\sum x^2\right) = \sum xy \cdot \sum x$$

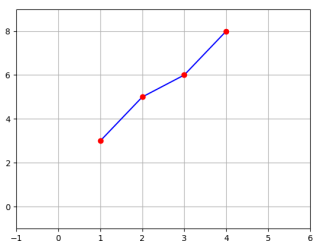
$$b \left[(\sum x)^2 - n \sum x^2\right] = \sum xy \cdot \sum x - \sum y \sum x^2$$

$$\boxed{\text{b} = \frac{\sum xy \cdot \sum x - \sum y \sum x^2}{(\sum x)^2 - n \sum x^2}} \quad \text{Ⓕ}$$

Vejamos um exemplo prático.

Exemplo: Determinar a reta que melhor se ajusta ao tabelamento.

x	y
1	3
2	5
3	6
4	8



Então, o objetivo do método dos quadrados mínimos é determinar uma reta que não necessariamente passa por todos os pontos, mas que tem o menor erro possível em relação ao ajuste dos pontos.

x	y	xy	x^2
1	3	3	1
2	5	10	4
3	6	18	9
4	8	32	16
10	22	63	30

$$a = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2} \qquad b = \frac{\sum x \sum xy - \sum y \sum x^2}{\left(\sum x\right)^2 - n\sum x^2}$$

$$a = \frac{4 \cdot 63 - 10 \cdot 22}{4 \cdot 30 - 10^2} \qquad b = \frac{10 \cdot 63 - 22 \cdot 30}{10^2 - 4 \cdot 30}$$

$$\boxed{a = 1.6} \qquad \boxed{b = 1.5}$$

$(\sum x)^2 = 10^2 = 100$

Então, nossa $g(x)$ pode ser escrita como:

$$\boxed{g(x) = 1.6x + 1.5}$$

Vamos calcular $g(3)$, já que temos esse ponto tabelado, só para efeito de comparação:

$$g(3) = 1.6 \cdot 3 + 1.5$$

$$g(3) = 6.3$$

Como o valor tabelado de $y_3 = 6$, podemos calcular o erro para esse ponto:

$$Erro_3 = |y_3 - g(x_3)|$$

$$Erro_3 = |6 - 6.3|$$

$$\boxed{Erro_3 = 0.3}$$

Método dos Quadrados Mínimos: ajustes não lineares

Os ajustes não lineares pelo método dos quadrados mínimos serão trabalhados fazendo-se transformações do modelo linear. Mostraremos aqui, o mesmo exemplo do ajuste linear, com transformações logarítmica, exponencial e potência.

Ⓐ Logaritmo

Escreveremos a $g_2(x)$ como,

$$g_2(x) = a \cdot ln(x) + b$$

Comparando com a função linear, $y = a \cdot x + b$, podemos colocar $ln(x)$ no lugar de x.

Vamos reescrever nossa tabela de valores e somatórios para o caso linear e uma tabela para o caso logarítmico fazendo a substituição de x por $ln(x)$.

Linear

x	y	xy	x^2
1	3	3	1
2	5	10	4
3	6	18	9
4	8	32	16
10	22	63	30

Logarítmica

$ln(x)$		$ln(x)y$	$ln(x)^2$
x	y	xy	x^2
0	3	0	0
0.6931	5	3.4655	0.4804
1.0986	6	6.5916	1.2069
1.3863	8	11.0904	1.9219
3.178	22	21.1475	3.6092

$$a = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

$$a = \frac{4 \cdot 21.1475 - 3.178 \cdot 22}{4 \cdot 3.6092 - (3.178)^2}$$

$$\boxed{a = 3.3833}$$

$$b = \frac{\sum x \sum xy - \sum y \sum x^2}{(\sum x)^2 - n\sum x^2}$$

$$b = \frac{3.178 \cdot 21.1475 - 22 \cdot 3.6092}{(3.178)^2 - 4 \cdot 3.6092}$$

$$\boxed{b = 2.8119}$$

$$\boxed{g_2(x) = 3.3833\ln(x) + 2.8119} \rightarrow \text{ajuste logarítmico,}$$

Ⓑ Exponencial

Nossa $g_3(x)$ será definida como:

$$g_3(x) = b \cdot e^{ax}$$

Para facilitar, chamaremos a $g_3(x)$ de y e faremos alguns cálculos para comparar mais facilmente essa função com a linear.

$y = b \cdot e^{ax}$

$ln(y) = ln(b \cdot e^{ax})$

$ln(y) = ln(b) + lne^{ax}$

$ln(y) = ax + ln(b)$ $\rightarrow$ comparando com $y = ax + b$, temos as seguintes transformações:

$ln(y) \rightarrow y_2$

$ln(b) \rightarrow b_2$

A tabela de valores ficará assim:

Exponencial

.	$ln(y)$	$xln(y)$	
x	y_2	xy_2	x^2
1	1.09861	1.09861	1
2	1.6094	3.2188	4
3	1.7917	5.3752	9
4	2.0794	8.3177	16
10	6.5792	18.0105	30

$$a = \frac{n\sum xy_2 - \sum x \sum y_2}{n\sum x^2 - (\sum x)^2} \qquad b = \frac{\sum x \sum xy_2 - \sum y_2 \sum x^2}{(\sum x)^2 - n\sum x^2}$$

$$a = \frac{4 \cdot 18.0105 - 10 \cdot 6.5792}{4 \cdot 30 - (10)^2} \qquad b = \frac{10 \cdot 18.0105 - 6.5792 \cdot 30}{(10)^2 - 4 \cdot 30}$$

$$\boxed{a = 0.3125} \qquad \boxed{b_2 = 0.86355}$$

O valor $\boxed{b_2 = 0.86355}$ é, na verdade, o valor de $ln(b)$. Então, se calcularmos e^{b_2} teremos o valor de b. $\boxed{b = 2.37156}$

$$\boxed{g_3(x) = 2.37156 \cdot e^{0.3125x}} \rightarrow \text{ajuste exponencial,}$$

© Potência

Nossa $g_4(x)$ será definida como,

$$g_4(x) = b \cdot x^a$$

Para facilitar, chamaremos a $g_4(x)$ de y e faremos alguns cálculos para comparar mais facilmente essa função com a linear.

$y = b \cdot x^a$

$ln(y) = ln(b \cdot x^a)$

$ln(y) = ln(b) + a \cdot ln(x)$ $\rightarrow$ comparando com $y = ax + b$, temos as seguintes transformações:

$ln(y) \to y_2$

$ln(x) \to x_2$

$ln(b) \to b_2$

A tabela de valores vai ficar assim:

Potência

$ln(x)$	$ln(y)$	$ln(x)ln(y)$	$ln(x)^2$
x_2	y_2	x_2y_2	x_2^2
0	1.09861	0	0
0.6931	1.6094	1.1156	0.4804
1.0986	1.7917	1.9684	1.2069
1.3863	2.0794	2.8827	1.9217
3.178	6.5792	5.9667	3.6092

$$a = \frac{n\sum x_2y_2 - \sum x_2 \sum y_2}{n\sum x_2^2 - (\sum x_2)^2}$$

$$a = \frac{4 \cdot 5.9667 - 3.178 \cdot 6.5792}{4 \cdot 3.6092 - (3.178)^2}$$

$$\boxed{a = 0.6820}$$

$$b_2 = \frac{\sum x_2 \sum x_2y_2 - \sum y_2 \sum x_2^2}{(\sum x_2)^2 - n\sum x_2^2}$$

$$b_2 = \frac{3.178 \cdot 5.9667 - 6.5792 \cdot 3.6092}{(3.178)^2 - 4 \cdot 3.6092}$$

$$\boxed{b_2 = 1.1029}$$

O valor $\boxed{b_2 = 1.1029}$ é, na verdade, o valor de $ln(b)$. Então, se calcularmos e^{b_2} teremos o valor de b. $\boxed{b = 3.0129}$

$\boxed{g_4(x) = 3.0129 \cdot x^{0.6820}} \to$ ajuste função potência,

4.3.3 Qualidade do ajuste

Qual função melhor se ajustou ao conjunto de pontos? Para responder a essa pergunta vamos estabelecer dois parâmetros para aferir a qualidade do ajuste obtido: um coeficiente de determinação R^2 e a variância residual σ^2.

Coeficiente de determinação

O R^2 varia entre 0 e 1, $0 \leqslant R^2 \leqslant 1$. Quanto mais próximo de 0, pior o ajuste. Quanto mais próximo de 1, melhor o ajuste.

O R^2 é determinado por:

$$R^2 = \frac{SQReg}{SQTot}$$

onde:

$SQReg$ é a soma dos quadrados da regressão

$SQTot$ é a soma dos quadrados totais

$SQReg = \sum (g(x_i) - \bar{y})^2$

$SQRes = \sum (g(x_i) - y_i)^2 \rightarrow$ Soma dos quadrados dos resíduos

$SQTot = \sum (y_i - \bar{y})^2$

$SQTot = SQReg + SQRes$

$SQReg = SQTot - SQRes$

$$R^2 = 1 - \frac{SQRes}{SQTot} \tag{4.3}$$

ou

$$R^2 = 1 - \frac{D(b_0, b_1)}{\sum y_i^2 - \frac{1}{n}\left(\sum y_i\right)^2} \tag{4.4}$$

Variância residual

Um outro parâmetro importante para aferir a qualidade do ajuste é a variância residual σ^2 definida por,

$$\sigma^2 = \frac{D(b_0, b_1)}{n - p} \tag{4.5}$$

onde $D(b_0, b_1)$ é dado por 4.2, n é o número de pontos e p é o número de parâmetros estimados. No caso de regressão linear simples $u = b_0 + b_1 x$, tem-se que $p = 2$.

Tanto o numerador quanto o denominador de 4.5 irão diminuir se forem introduzidos mais parâmetros no modelo, como será visto mais adiante nos exemplos. Todavia, será a redução global de σ^2 que definirá se mais parâmetros devem ou não ser incorporados ao modelo.

Exemplo 1:

Vamos calcular a qualidade do ajuste (R^2) para os ajustes que fizemos para o conjunto de pontos:

x	1	2	3	4
y	3	5	6	8

MQM - Linear

x	y	xy	x^2	$SQReg$	$SQTot$
1	3	3	1	5.76	6.25
2	5	10	4	0.64	0.25
3	6	18	9	0.64	0.25
4	8	32	16	5.76	6.25
10	22	63	30	12.8	13.0

$$\left.\begin{array}{l} a = 1.6 \\ b = 1.5 \end{array}\right\} g(x) = 1.6x + 1.5$$

$$R^2 = \frac{SQReg}{SQTot} = \frac{12.8}{13}$$

$$\bar{y} = \frac{\sum y}{n} = \frac{22}{4}$$

$$\boxed{R^2 = 0.9846}$$

$\bar{y} = 5.5$

$g(x) = 1.6x + 1.5$

$g(1) = 0.1$

$g(2) = 4.7$

$g(3) = 6.3$

$g(4) = 7.9$

MQM - Logarítmico

x	y	xy	x^2	$SQReg$	$SQTot$
0	3	0	0	7.2269	6.25
0.6931	5	3.4657	0.4805	0.1177	0.25
1.0986	6	6.5917	1.2069	1.0581	0.25
1.3863	8	11.0904	1.9218	4.0078	6.25
3.178	22	21.148	3.6092	12.4105	13.0

$$g(x) = a \cdot ln(x) + b$$

$$\left.\begin{array}{l} a = 3.3833 \\ b = 2.8117 \end{array}\right\} g(x) = 3.3833 \cdot ln(x) + 2.8117$$

$$R^2 = \frac{SQReg}{SQTot} = \frac{12.4105}{13}$$

$$\boxed{R^2 = 0.9548}$$

$\bar{y} = 5.5$

MQM - Exponencial

x	y	xy	x^2	$SQReg$	$SQTot$
1	3	0	0	7.2269	6.25
2	5	3.4657	0.4805	0.1177	0.25
3	6	6.5917	1.2069	1.0581	0.25
4	8	11.0904	1.9218	4.0078	6.25
10	6.5793	18.0105	30	0.4882	0.5101

$$g(x) = b \cdot e^a x$$

$$\left.\begin{array}{c} a = 0.3125 \\ b = 0.8636 \\ ln(b) = 0.8636 \quad \text{(linear)} \\ b = 2.37156 \quad \text{(exponencial)} \end{array}\right\} g(x) = 0.3125 \cdot x + 0.8636 \qquad \text{(linear)}$$

$g(x) = 2.37156 \cdot e^{(0.3125x)}$ (exponencial)

$\bar{y} = 1.6448$

$$R^2 = \frac{SQReg}{SQTot} = \frac{0.4882}{0.5101}$$

$$\boxed{R^2 = 0.9571}$$

Exemplo 2:

Calcule o coeficiente de determinação e a variância residual, utilizando os dados das Tabela abaixo, pelas equações (4.4) e (4.5).

i	x_i	y_i	x_i^2	$x_i y_i$	y_i^2
1	0.3	1.8	0.09	0.54	3.24
2	2.7	1.9	7.29	5.13	3.61
3	4.5	3.1	20.25	13.95	9.61
4	5.9	3.9	34.81	23.01	15.21
5	7.8	3.3	60.84	25.74	10.89
$\sum$	21.2	14.0	123.28	68.37	42.56

i	x_i	y_i	$x_i y_i$	y_i^2
1	0.3	1.8	1.7369	0.0631
2	2.7	1.9	2.3845	-0.4845
3	4.5	3.1	2.8701	0.2299
4	5.9	3.9	3.2478	0.6522
5	7.8	3.3	3.7604	-0.4604
$D(1.6560; 0.2698) = 0.9289$				

$$R^2 = 1 - \frac{D(b_0, b_1)}{\sum y_i^2 - \frac{1}{n}\left(\sum y_i\right)^2} \qquad \sigma^2 = \frac{D\left(b_0, b_1\right)}{n - p}$$

$$R^2 = 1 - \frac{0.9289}{42.56 - (14.0)^2/5} \therefore R^2 = 0.7235 \qquad \sigma^2 = \frac{0.9289}{5 - 2} \therefore \sigma^2 = 0.3096$$

4.3.4 Implementações em Python

O Programa 4.4 implementa o método dos quadrados mínimos em Python 3. A função $mmq(x, y, imprime\_tabela)$ recebe os valores de x e y e retorna os coeficientes a e b para o ajuste. Recebe ainda a opção booleana se o usuário deseja imprimir a tabela de somatórios ou não. Essa opção pode facilitar os testes.

Programa 4.4: Método dos Quadrados Mínimos

```
1  import numpy as np
2  import pandas as pd
3
4  def mmq(x,y, imprime_tabela=False):
5      """
6      :param x: lista de x
7      :param y: lista de y
8      :param imprime_tabela: booleano para imprimir (True)
9          ou nao(False) a tabela de somatorios
10     :return: valores dos coeficientes a e b
11     """
12     n = len(x)
13     somas = np.zeros([n,4], dtype=float)
14
15     for i in range(0,n):
16         somas[i][0]=float(x[i])
17         somas[i][1]=float(y[i])
18         somas[i][2]=float(x[i]*y[i])
19         somas[i][3]=float(x[i]*x[i])
20     s = np.einsum("ij->j", somas)
21     somas = np.vstack([somas, s])
22     # imprimindo a tabela de somatorios
23     if (imprime_tabela):
24         somas_table = pd.DataFrame(somas,
25             columns=["x","y","xy","x^2"])
26         print(somas_table)
27
28     a = (n*somas[n][2]-somas[n][0]*somas[n][1])/
```

```
29          (n*somas[n][3]-(somas[n][0])**2)
30      b = (somas[n][0]*somas[n][2]-somas[n][1]*somas[n][3])/
31          ((somas[n][0])**2-n*somas[n][3])
32      print("a={}".format(a))
33      print("b={}".format(b))
34      return [a,b]
```

Como pode ser observado, a função *mmq*() calcula os coeficientes a e b necessários para o ajuste de qualquer tipo de curva. No entanto, para cada tipo de ajuste teremos funções auxiliares que acionam a *mmq*(). O Programa 4.5 mostra a função *ajuste_linear*() a qual aciona a função *mmq*() prosseguindo num ajuste linear dos parâmetros x e y, recebidos. O Programa também apresenta um exemplo de uso. O gráfico de saída do Programa 4.5 é exibido na Figura 4.5.

Programa 4.5: Ajuste linear usando o Método dos Quadrados Mínimos

```
1  import numpy as np
2
3  def ajuste_linear(x,y):
4      abr2 = mmq(x,y,True)
5      y_ = np.sum(y)/len(y)
6
7      sq_reg = 0
8      sq_tot = 0
9      for i in range(len(x)):
10         sq_reg += pow(((abr2[0]*x[i]+abr2[1])-y_),2)
11         sq_tot += pow((y[i]-y_),2)
12     r2 = sq_reg/sq_tot
13     abr2.append(r2)
14     return abr2
15
```

```
16 def g_linear(ab,x):
17     return ab[0]*x + ab[1]
18
19 if (__name__=="__main__"):
20     x = [1, 2, 3, 4]
21     y = [3, 5, 6, 8]
22     plt.plot(x, y, "ko")
23     plt.plot(x, y, "k--")
24     legenda = ["nos de interpolacao", "f(x)"]
25
26     print("\n\nAjuste linear")
27     ab=ajuste_linear(x,y)
28
29     gx=[]
30     for i in x:
31         gx.append(g_linear(ab,i))
32     print("R2 = {}".format(ab[2]))
33     plt.plot(x, gx, "b-")
34     legenda.append("lin(x)={}*x+{}--r2{}".format(
35         round(ab[0],2),round(ab[1],2), round(ab[2],2)))
```

Para os demais tipos de ajuste, a implementação segue a mesma lógica. O Programa 4.6 traz implementações de ajustes exponencial, logarítmico e de potência. Pode-se usar a mesma lógica do Programa 4.5 como exemplo de aplicação.

Programa 4.6: Ajustes não lineares usando o MQM

```
1 import numpy as np
2
3 def ajuste_exponencial(x,y):
4     lny = np.log(y)
```

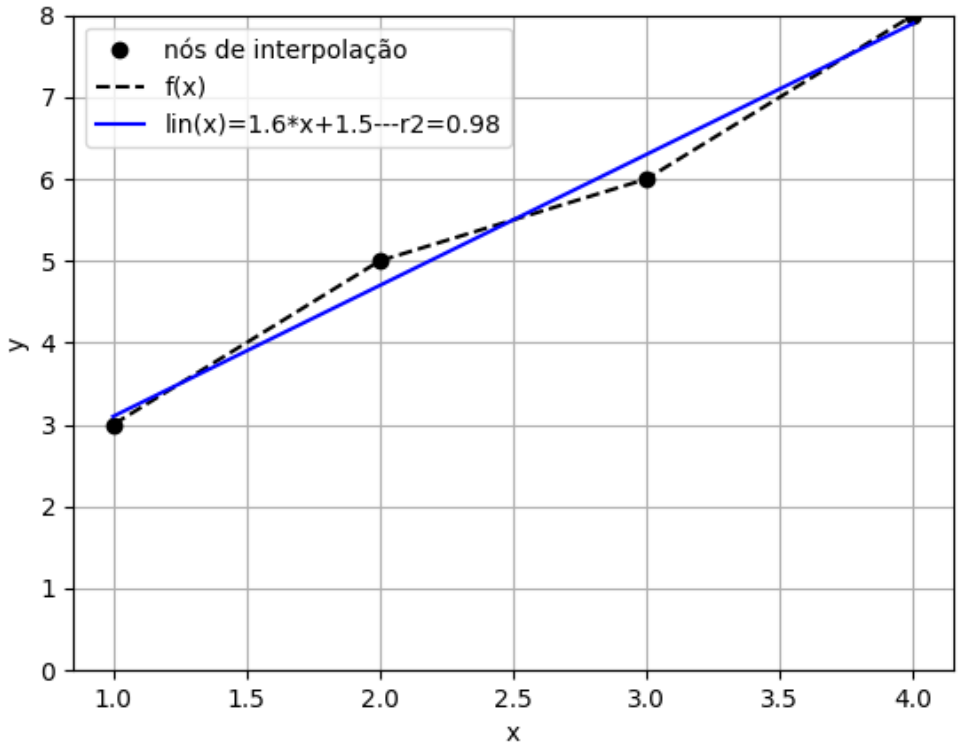


Figura 4.5: Saída do Programa 4.5.

```
5       print(lny)
6       abr2 = mmq(x,lny,1)
7       y_ = np.sum(lny) / len(y)
8       sq_reg = 0
9       sq_tot = 0
10      for i in range(len(x)):
11          sq_reg += pow(((abr2[0] * x[i] + abr2[1]) - y_), 2)
12          sq_tot += pow((lny[i] - y_), 2)
13      r2 = sq_reg / sq_tot
14
15      abr2.append(r2)
16      abr2[1] = np.exp(abr2[1])
17
18      return abr2
```

```
19
20 def g_exponencial(ab,x):
21     return ab[1]*np.exp(ab[0]*x)
22
23 def ajuste_logaritmico(x,y):
24     lnx = np.log(x)
25     abr2 = mmq(lnx,y,1)
26     y_ = np.sum(y) / len(y)
27
28     sq_reg = 0
29     sq_tot = 0
30     for i in range(len(x)):
31         sq_reg += pow(((abr2[0] * lnx[i] + abr2[1]) - y_), 2)
32         sq_tot += pow((y[i] - y_), 2)
33     r2 = sq_reg / sq_tot
34     abr2.append(r2)
35     return abr2
36
37 def g_log(ab,x):
38     return ab[0]*np.log(x)+ab[1]
39
40 def ajuste_potencia(x,y):
41     lnx= np.log(x)
42     lny = np.log(y)
43     abr2 = mmq(lnx,lny,1)
44     y_ = np.sum(lny) / len(y)
45     sq_reg = 0
46     sq_tot = 0
47     for i in range(len(x)):
```

```
48          sq_reg += pow(((abr2[0] * lnx[i] + abr2[1]) - y_), 2)
49          sq_tot += pow((lny[i] - y_), 2)
50
51      r2 = sq_reg / sq_tot
52      abr2.append(r2)
53      abr2[1] = np.exp(abr2[1])
54      return abr2
55
56  def g_potencia(ab,x):
57      return ab[1]*pow(x,ab[0])
```

4.4 Regressão Linear Múltipla

Num modelo de regressão linear múltipla temos mais de uma variável de entrada e/ou de saída. Novamente, o melhor ajuste é obtido pela minimização da soma dos quadrados dos resíduos estimados.

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_{1i} - a_2 x_{2i} - \cdots - a_n x_{ni})^2$$

Consideremos a regressão linear múltipla da função:

$$y = a_0 + a_1 x_1 + a_2 x_2 + e$$

onde e é o erro da estimativa

Se considerarmos um problema com duas variáveis de entrada, aplicando a derivada parcial em função de cada um dos coeficientes $a_0, a_1, a_2, \ldots, a_n$ obtemos o sistema,

$$\begin{bmatrix} n & \sum x_{1,i} & \sum x_{2,i} \\ \sum x_{1,i} & \sum x_{1,i}^2 & \sum x_{1,i} \sum x_{2,i} \\ \sum x_{2,i} & \sum x_{1,i} x_{2,i} & \sum x_{2,i}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_1 y_i \\ \sum x_2 y_i \end{bmatrix}$$

$$\begin{cases} na_0 + \sum x_{1,i}a_1 + \sum x_{2,i}a_2 = \sum y_i \\ \sum x_{1,i}a_0 + \sum x_{1,i}^2 a_1 + \sum x_{1,i}\sum_{x_2,i} a_2 = \sum x_1 y_i \\ \sum x_{2,i}a_0 + \sum x_{1,i}x_{2,i}a_1 + \sum x_{2,i}^2 a_2 = \sum x_2 y_i \end{cases} \tag{4.6}$$

Exemplo: Encontre o modelo de Regressão Linear Múltipla para os dados abaixo.

x_1	x_2	y
0	0	5
2	1	10
2.5	2	9
1	3	0
4	6	3
7	2	27

$\rightarrow$

x_1	x_2	y	x_1^2	x_1x_2	x_2^2	x_1y	x_2y
0	0	5	0	0	0	0	0
2	1	10	4	2	1	20	10
2.5	2	9	6.25	5	4	22.5	18
1	3	0	1	3	9	0	0
4	6	3	16	24	36	12	18
7	2	27	49	14	4	189	54
16.5	14	54	76.25	48	54	243.5	100

$n = 6$

Ajustando os dados da tabela ao sistema de 4.6, obtemos:

$6a_0 + 16.5a_1 + 14a_2 = 54$ $\qquad a_0 = 5$

$16.5a_0 + 76.25a_1 + 48a_2 = 243.4$ $\qquad a_1 = 4$

$14a_0 + 48a_1 + 54a_2 = 100$ $\qquad a_2 = -3$

$\boxed{y = 5 + 4x_1 - 3x_2}$

O caso bidimensional do exemplo pode ser facilmente estendido para m dimensões como em,

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_mx_m + e$$

Além disso, o coeficiente de determinação, R^2, pode ser calculado pela equação 4.3 e a variância residual, σ^2, por 4.5.

Quadrados mínimos linear geral

Regressões linear, polinomial e múltipla, todos pertencem ao modelo de mínimos quadrados linear geral:

$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots + a_m z_m + e$

onde, $z_0, z_1 \ldots z_m$ são um conjunto de $m+1$ funções base e e é o erro do ajuste. Podemos ver facilmente como a regressão linear simples e a linear múltipla se encaixam nesse modelo, isto é, $z_0 = 1$, $z_1 = x_1$, $z_2 = x_2$, $\ldots$, $z_m = x_m$. Além disso, a regressão polinomial também é incluída se as funções-base forem monômios simples como em $z_0 = x^0 = 1, z_1 = x, z_2 = x^2, \ldots, z_m = x^m$.

$y = [z]\{a\} + \{e\}$

4.5 Regressão Polinomial

Existem dados que não são representados de forma satisfatória por uma reta. Um método para atingir esse objetivo é usar transformações lineares, como já visto. Outra alternativa é ajustar polinômios aos dados usando regressão polinomial.

O procedimento dos mínimos quadrados pode ser prontamente estendido para ajustar dados por um polinômio de grau mais alto. Por exemplo, suponha que queira ajustar um polinômio de $2^{\underline{o}}$ grau ou quadrático.

$$y = a_0 + a_1 x + a_2 x^2 + e$$

Nesse caso, a soma dos quadrados dos resíduos é:

$$S_r = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_1 - a_2 x_2^2\right)^2 \tag{4.7}$$

Toma-se a derivada parcial da Equação 4.7 com relação a cada um dos coeficientes desconhecidos do polinômio.

$$\begin{aligned}
\frac{\partial S_r}{\partial a_0} &= -2\sum_{i=1}^{n}\left(y_1 - a_0 - a_{x1} - a_2 x_i^2\right) \\
\frac{\partial S_r}{\partial a_1} &= -2\sum_{i=1}^{n} x_i\left(y_1 - a_0 - a_{x1} - a_2 x_i^2\right) \\
\frac{\partial S_r}{\partial a_2} &= -2\sum_{i=1}^{n} x_i^2\left(y_1 - a_0 - a_{x1} - a_2 x_i^2\right)
\end{aligned}$$

Essas equações podem ser igualadas a 0 e reorganizadas para determinar o seguinte conjunto de equações normais:

$$\begin{aligned} na_0 + (\sum x_i)\, a_1 + (\sum x_i^2)\, a_2 &= \sum y_i \\ (\sum x_i)\, a_0 + (\sum x_i^2)\, a_1 + (\sum x_i^3)\, a_2 &= \sum x_i y_i \\ (\sum x_i^2)\, a_0 + (\sum x_i^3)\, a_1 + (\sum x_i^4)\, a_2 &= \sum x_1^2 y_i \end{aligned}$$

O caso bidimensional pode ser estendido para um polinômio de grau n.

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + e$$

Exemplo 1: Ajuste um polinômio do 2^{o} grau aos dados a seguir.

$$n = 6 \qquad \sum_{i=1}^{6} x_i = 15 \qquad \sum_{i=1}^{6} y_i = 152.6 \qquad \sum_{i=1}^{6} x_i^2 = 55$$
$$\sum_{i=1}^{6} x_i^3 = 225 \quad \sum_{i=1}^{6} x_i^4 = 979 \quad \sum_{i=1}^{6} x_i y_i = 585.6 \quad \sum_{i=1}^{6} x_i^2 y_i = 2488.8$$

Substituindo os valores nas equações normais, temos:

$$\begin{cases} 6a_0 + 15a_1 + 55a_2 &= 152.6 \\ 15a_0 + 55a_1 + 225a_2 &= 585.6 \\ 55a_0 + 225a_1 + 979a_2 &= 2488.8 \end{cases}$$

Resolvendo esse sistema por qualquer método, temos:

$a_0 \cong 2.47857$

$a_1 \cong 2.35929$

$a_2 \cong 1.86071$

$$\boxed{y = 2.47857 + 2.35929x + 1.86071x^2}$$

Exemplo 2: Dada a seguinte tabela,

x	-1	0	1	2
y	0	-1	0	7

aproximar $f(x)$ por um polinômio quadrático usando o método dos mínimos quadrados.

$$a_0 = -1.6$$
$$a_1 = 0.2$$
$$a_2 = 2$$

$$\boxed{y = 2x^2 + 0.2x - 1.6}$$

4.5.1 Algoritmo e implementação

O algoritmo de regressão linear múltipla e polinomial via equações normais, apresentado no Algoritmo 6 foi proposto por Campos (2002). Nos Algoritmos 7 e **??** temos, respectivamente, os cálculos das equações normais e do coeficiente de determinação e variância residual.

O programa que implementa esse algoritmo pode ser visto em 4.7 e precisou passar por adaptações para se adequar à linguagem Python e resolver alguns problemas observados durante os testes. Observe que o programa utiliza funções já vistas anteriormente, como a *cholesky()*, Programa 3.7.

Programa 4.7: Regressão linear múltipla e polinomial via equações normais

```
1  import numpy as np
2  from sistemas_lineares import cholesky
3
4  def regressao_linear_EN(n,v,p,xx,yy):
5      """
6      Algoritmo Frederico Campos Filho para calcular
7      parametros de quadrados minimos de modelo linear
8      multiplo via equacoes normais
9      :param n: numero de pontos
10     :param v: numero de variaveis
11     :param p: numero de parametros
12     :param x: variaveis explicativas
13     :param y: variaveis respostas
14     :return: b (coef. de regressao), r2 (coef. de determinacao),
15     sigma2 (variancia residual) e condErro (condicao de erro)
```

```
16      """
17      if (v > 1 and (v + 1) != p):
18          condErro = True
19          return
20
21      y = yy.copy()
22      condErro = False
23      # equacoes normais
24      sxx = np.zeros([p, p])
25      sxy = np.zeros(p)
26
27      #regressao polinomial
28      x = np.ones([n,p])
29      for j in range(1,p):
30          for i in range(n):
31              x[i][j] = pow(xx[i],j)
32      if (v == 1 and p > 2):
33          for i in range(p):
34              for j in range(p):
35                  soma = 0
36                  for k in range(n):
37                      soma += pow(xx[k], (i + j))
38                  sxx[i][j] = soma
39          for i in range(p):
40              soma = 0
41              for k in range(n):
42                  soma = soma + (pow(xx[k],i) * y[k])
43              sxy[i] = soma
44
```

```
45          b = cholesky(sxx, sxy)
46          # se houve erro na resolucao do sistema por cholesky
47          # tenta por gauss
48          if(isinstance(b, bool)):
49              b = gauss(sxx,sxy)
50
51          D = 0
52          sy2 = 0
53          for i in range(n):
54              u = 0
55              for j in range(p):
56                  u = u + b[j] * x[i][j]
57              d = y[i] - u
58              D = D + (d * d)
59              sy2 = sy2 + pow(y[i], 2)
60          r2 = 1 - D / (sy2 - pow(sxy[1], 2) / n)
61          sigma2 = D / (n - p)
62      #regressao linear multipla
63      else:
64          uns = np.ones([n, 1], float)
65          x = np.column_stack((uns, xx))
66          for i in range(p):
67              for j in range(p):
68                  soma = 0
69                  for k in range(n):
70                      soma = soma + x[k][i]*x[k][j]
71                  sxx[i][j] = soma # matriz dos coeficientes
72              soma = 0
73              for k in range(n):
```

```
74                  soma = soma + x[k][i]* y[k]
75              sxy[i] = soma # vetor dos termos independentes
76
77          b = cholesky(sxx, sxy)
78
79          D = 0
80          sy2 = 0
81          for i in range(n):
82              u = 0
83              for j in range(p):
84                  u = u + b[j] * x[i][j]
85              d = y[i] - u
86              D = D + (d * d)
87              sy2 = sy2 + pow(y[i], 2)
88          r2 = 1 - D / (sy2 - pow(sxy[1], 2) / n)
89          sigma2 = D / (n - p)
90
91      return b, r2, sigma2, condErro
```

Exemplos de uso do Programa 4.7

Exemplo 1: produto interno bruto dos Estados Unidos de 1947 a 1962

Ajustar os dados da tabela ao modelo $u = b_0 + b_1x_1 + b_2x_2$, usando o Programa 4.7.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_{i1}	60.3	61.1	60.2	61.2	63.2	63.6	65.0	63.8	66.0	67.9	68.2	66.5	68.7	69.6	69.3	70.6
x_{i2}	108	109	110	112	112	113	115	116	117	119	120	122	123	125	128	130
y_i	234	259	258	285	329	347	365	363	396	419	443	445	483	503	518	555

- x_1: total de empregos (milhões),
- x_2: população com 14 anos ou mais (milhões) e
- y: PIB americano (bilhões de dólares).

Para resolver esse problema de regressão, utilizamos o Programa 4.8, o qual chama a função *reg_linear_EN()*, para regressão linear múltipla.

Programa 4.8: Uso do Programa 4.7 com dados do Exemplo 1

```
import numpy as np
# ... funcao regressao_linear_EN()...

if (__name__=="__main__"):
    n= 16
    v = 2
    p = 3
    x = [[60.3, 61.1,60.2,61.2,63.2,63.6,65.0,63.8,
          66.0,67.9,68.2,66.5,68.7,69.6,69.3,70.6],
          [108.0,109.0,110.0,112.0,112.0,113.0,115.0,116.0,
          117.0,119.0,120.0,122.0,123.0,125.0,128.0,130.0]]
    x = np.transpose(x)
    y = [234,259,258,285,329,347,365,363,
         396,419,443,445,483,503,518,555]
    b, r2, sigma2, condErro = regressao_linear_EN(n,v,p,x,y)
    print("coef. de regressao: {}\n
        coef. de determinacao: {}\n
        variancia residual:{}\n
        Erro:{}".format(b, r2, sigma2, condErro))
```

A saída do Programa 4.8 é:

```
-------------------------------------------------------------
coef. de regressão: [-1407.40086505    13.45112442     7.80271346]
coef. de determinação: 1.0000001035035568
variância residual:83.75810960349015
Erro:False
```

Exemplo 2: raiz quadrada de x para $0.01 \leq x \leq 1$

A partir da tabela, que compila os valores de $\sqrt{x}$ para $0.01 \leq x \leq 1$, determine o polinômio de quadrados mínimos de grau $g = 3$ utilizando o Programa 4.7.

i	x_i	$\sqrt{x_i}$
1	0.01	0.1000
2	0.10	0.3162
3	0.20	0.4472
4	0.30	0.5477
5	0.40	0.6325
6	0.50	0.7071
7	0.60	0.7746
8	0.70	0.8367
9	0.80	0.8944
10	0.90	0.9487
11	1.00	1.0000

Programa 4.9: Aplicação do Programa 4.7 no Exemplo 2

```
1  import numpy as np
2  # ... funcao regressao_linear_EN()...
3  if (__name__=="__main__"):
4      n = 11
5      v = 1
6      p = 4
7      x = [[0.01,0.1,0.2,0.3,0.4,0.5,0.6,
8          0.7,0.8,0.9,1.0]]
9      x = np.transpose(x)
10     y = [0.1,0.3162,0.4472,0.5477,0.6325,
11         0.7071,0.7746,0.8367,0.8944,
12         0.9487,1.0]
13     b, r2, sigma2, condErro =
14         regressao_linear_EN(n, v, p, x, y)
15     print("coef. de regressao: {}\n
16         coef. de determinacao: {}\n
17         variancia residual:{}\n
18         Erro:{}".format(b, r2,
19             sigma2, condErro))
```

A saída do Programa 4.9 é:

```
------------------------------------------------------------
coef. de regressão: [-1407.40086505    13.45112442     7.80271346]
coef. de determinação: 1.0000001035035568
variância residual:83.75810960349015
Erro:False
```

Exemplo 3: População do Brasil

Sejam os dados históricos dos censos demográficos do Brasil com o Instituto Brasileiro de Geografia e Estatística (IBGE), apresentados na Tabela 4.1.

Tabela 4.1: Censos demográficos do Brasil.

Ano	Urbana	Rural
1940	12.880.182	28.356.133
1950	18.782.891	33.161.506
1960	31.303.034	38.767.423
1970	52.084.984	41.054.053
1980	80.436.409	38.566.297
1991	110.990.990	35.834.485
1996	123.076.831	33.993.332
2000	137.953.959	31.845.211

Deseja-se determinar qual o melhor grau para uma regressão polinomial.

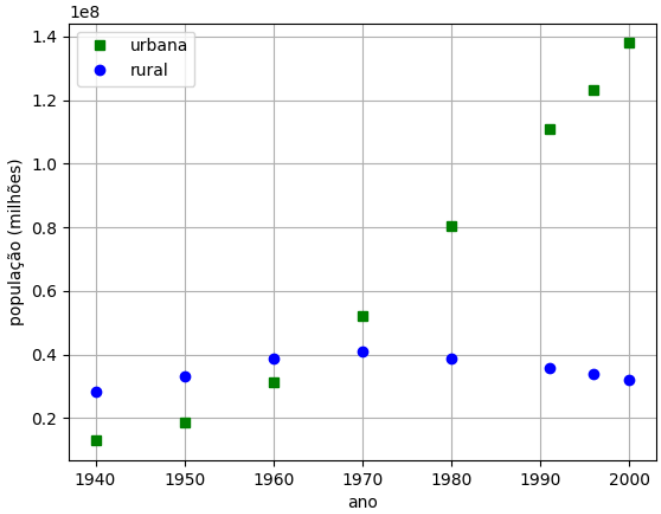


Figura 4.6: Gráfico de dispersão - censo demográfico do Brasil.

Pelo gráfico de dispersão dos dados, Figura 4.6, podemos observar que o ajuste ano x urbana não deve ser feito por um polinômio de grau 1 e sim por um polinômio de grau mais elevado.

Para trabalharmos com esses dados, fizemos uma mudança de variáveis para reduzir erros de arredondamento. Assim, a variável explicativa fica centrada,de modo que $x = ano - 1970$ e a variável de resposta $y = urbana * 10^{-6}$ dada em milhões de habitantes. Para polinômios com grau maior que 1, calculamos o coeficiente de determinação e variância residual, onde concluímos que o melhor ajuste para esse caso é um polinômio de grau 3, como mostra o gráfico da Figura 4.7.

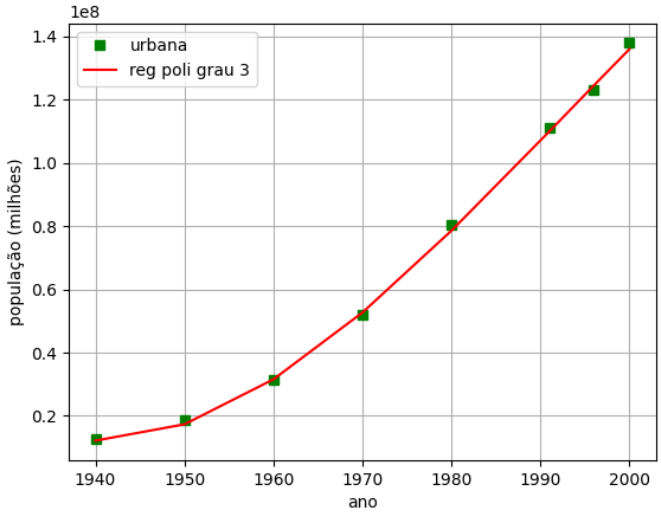


Figura 4.7: Ajuste do censo demográfico do Brasil com polinômio de grau 3.

4.6 Exercícios

1. Ajuste os dados abaixo pelo método dos quadrados mínimos utilizando:

   a) uma reta;

   b) uma parábola do tipo $ax^2 + bx + c$.

   Trace as duas curvas no gráfico de dispersão dos dados. Como você compararia as duas curvas com relação aos dados?

x	1	2	3	4	5	6	7	8
y	0.5	0.6	0.9	0.8	1.2	1.5	1.7	2.0

2. Dada a tabela abaixo, faça o gráfico de dispersão dos dados e ajuste uma curva da melhor maneira possível:

x	0.5	0.75	1	1.5	2.0	2.5	3.0
y	-2.8	-0.6	1	3.2	4.8	6.0	7.0

3. Seja a tabela,

i	x_i	y_i
1	0.5	5.1
2	1.2	3.2
3	2.1	2.8
4	3.5	1.0
5	5.4	0.4

Faça o diagrama de dispersão, usando um pacote gráfico do Python.

Encontre a reta de quadrados mínimos usando os cinco pontos da tabela e calcule D.

4. Dada a tabela, calcular o coeficiente de determinação r^2 do modelo $u = b_0 + b_1 x$. Calcule também a variância residual σ^2 do modelo.

i	x_i	y_i
1	1.4	4.2
2	2.1	2.3
3	3.0	1.9

5. Calcule o coeficiente de determinação e a variância residual do modelo linear $u = b_0 + b_1 x$ a partir dos dados da Tabela abaixo.

i	x_i	y_i	u_i	d_i
1	0.3	1.8	1.7369	0.0631
2	2.7	1.9	2.3845	-0.4845
3	4.5	3.1	2.8701	0.2299
4	5.9	3.9	3.2478	0.6522
5	7.8	3.3	3.7604	-0.4604
$D(1.6560; 0.2698) = 0.9289$				

6. Usando os dados (Ano, Urbana) da Tabela 4.1, calcule o coeficiente de determinação e a variância residual do modelo $u = b_0 + b_1 x$.

7. Ache a equação de quadrados mínimos $u = b_0 + b_1x_1 + b_2x_2$, a partir dos pontos da tabela:

i	x_{i1}	x_{i2}	y_i
1	-1	-2	13
2	0	-1	11
3	1	0	9
4	2	1	11
5	4	2	9
6	5	3	1
8	6	4	-1

8. Calcule os coeficiente do modelo $u = b_0 + b_1x + b_2x^2$ usando os pontos da tabela:

i	x_i	y_i
1	-2.0	-30.5
2	-1.5	-20.2
3	0.0	-3.3
4	1.0	8.9
5	2.2	16.8
6	3.1	21.4

9. Implemente o programa de regressão em Python e determine os parâmetros do modelo dos exercícios 7 e 8 usando o programa.

10. Dada a tabela, qual dos modelos abaixo é o melhor? Por quê?

i	x_i	y_i
1	5	0.01
2	6	0.05
3	7	0.08
4	8	0.14
5	9	0.18
6	10	0.26
7	11	0.44
8	12	0.51
9	13	0.79
10	14	1.02

(a) $u = b_0 + b_1 x$

(b) $u = b_0 + b_1 x + b_2 x^2$

(c) $u = b + 0 + b_1 x + b_2 x^2 + b_3 x^3$

(d) $u = a x^b$

(e) $u = a b^x$

Capítulo 5

Integração Numérica

5.1 Introdução - Lógica da Soma de Riemann

Seja a partição p no intervalo $[ab]$ (Figura 5.1).

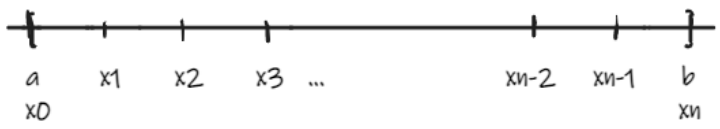


Figura 5.1: Partição em um intervalo $[ab]$.

$$p = \{x_0, x_1, x_2, \ldots, x_{n-2}, x_{n-1}, x_n\}$$

Dada uma partição p no intervalo, conseguimos calcular a amplitude (Δ) de cada pedaço.

$$\begin{aligned}
\Delta_{x_1} &= x_1 - x_0 \\
\Delta_{x_2} &= x_2 - x_1 \\
\Delta_{x_3} &= x_3 - x_2 \\
&\ldots \\
\Delta_{x_n} &= x_n - x_{n-1}
\end{aligned}$$

A amplitude da partição é definida como a maior amplitude da partição.

$\Delta_p = max\Delta_{xi}$

No gráfico, Figura 5.2, vamos definir a área de um pequeno retângulo. Chamaremos a área de A_R:

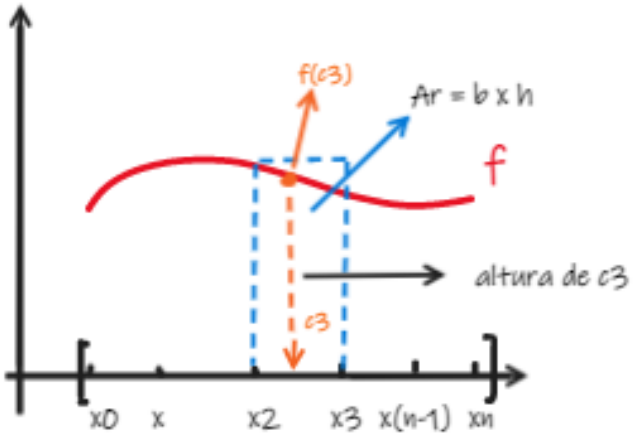


Figura 5.2: Esquema gráfico de partição em um intervalo $[ab]$.

$$\begin{aligned} A_R = &\quad b \times h \\ A_R = &\quad (x_3 - x_2) \times f(c_3) \\ A_R = &\quad \Delta_{x_3} \times f(c_3) \end{aligned}$$

Algumas variáveis podem afetar o cálculo da área desse retângulo, por exemplo:

- Posicionamento de c_i

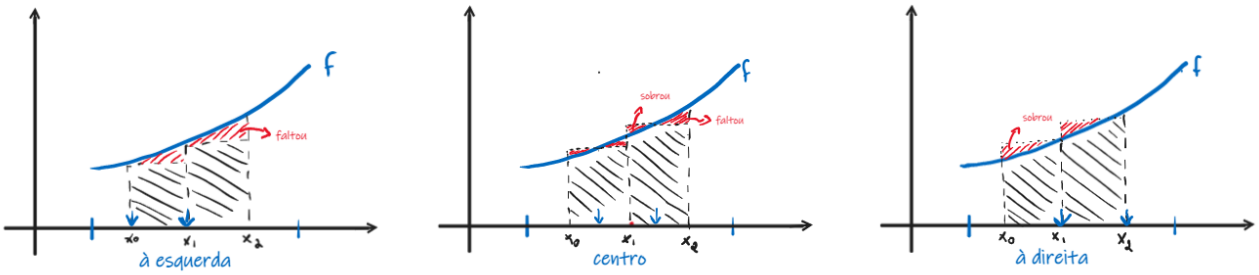


Figura 5.3: Posicionamento de C_i.

Então, dependendo do posicionamento de C, podemos calcular faltando, tentando equilibrar (centro) ou sobrando área, dependendo, claro, do comportamento da função f.

- Particionamento

Em quantos subintervalos dividiremos o intervalo?

Quanto mais dividimos o intervalo, menor tende a ser o erro na aproximação da soma de Riemann pela área sob a curva.

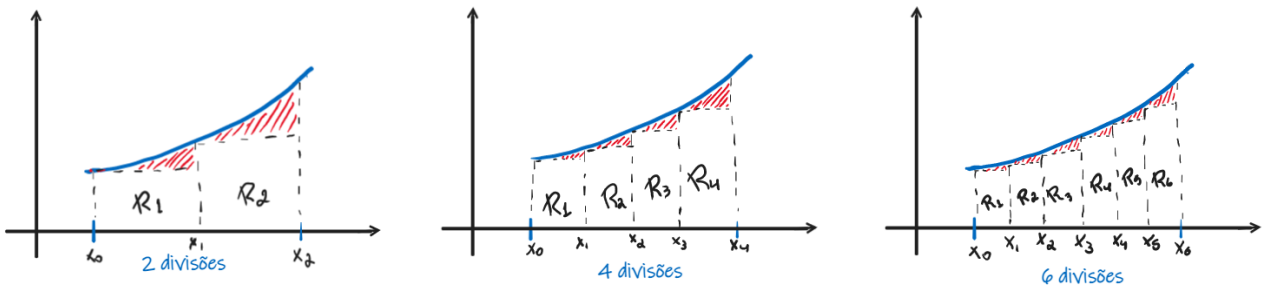
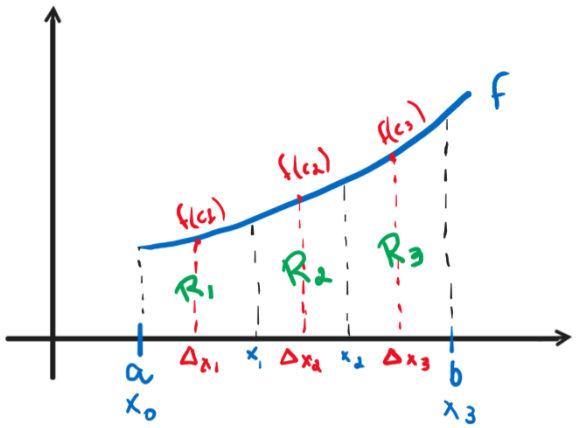


Figura 5.4: Particionamento do intervalo.

Soma de Riemann

$\sum_{i=1}^{n} f(c_i) \cdot \Delta x_i$

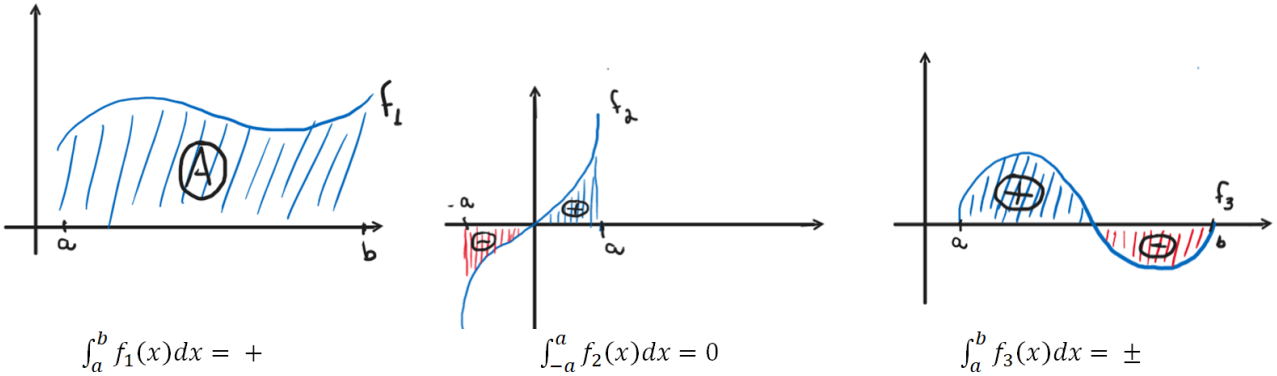
ou seja, vamos multiplicar cada partição pelo valor da função no ponto c que escolhermos aleatoriamente. Então teremos:



$$A_{R_1} = \Delta_{x_1} \cdot f(c_1) +$$
$$A_{R_2} = \Delta_{x_2} \cdot f(c_2) +$$
$$A_{R_3} = \Delta_{x_3} \cdot f(c_3)$$

$\sim$ Area no intervalo$[a, b]$

Vamos lembrar que a integral está relacionada ao cálculo de áreas. Veja alguns exemplos nos esquemas da Figura 5.1.



$\int_a^b f_1(x)dx = +$ $\int_{-a}^{a} f_2(x)dx = 0$ $\int_a^b f_3(x)dx = \pm$

Figura 5.5: Diferentes situações do intervalo de cálculo de áreas.

O resultado do cálculo da integral definida é, na verdade, o saldo de áreas.

Exemplo: Calcular $\int_1^2 x^2 dx$, utilizando 2 e 4 subintervalos iguais.

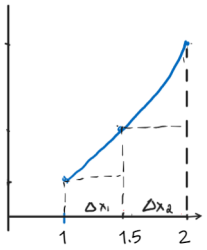
Solução: Analiticamente, sabemos que o resultado dessa integral definida é:

$$\int_1^2 x^2 dx = \frac{x^3}{3}\Big|_1^2 = \frac{2^3}{3} - \frac{1^3}{3} = \frac{7}{3} \cong 2.333\ldots$$

Aproximação utilizando 2 subintervalos:

Cálculo da amplitude: $h = \frac{b-a}{n} = \frac{2-1}{2} = 0.5$

x	y
1	1
1.5	2.25
2	4



$S \cong \sum_{i=1}^{2} f(c_i) \cdot \Delta_{x_i}$

$S \cong 1 \cdot 0.5 + 2.25 \cdot 0.5$

$S \cong 1.625$

Vamos calcular o erro dessa aproximação com relação ao resultado analítico que encontramos.

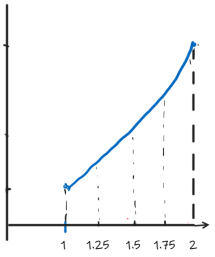
$\text{Erro}_{abs} = |vReal - vAprox| = 2.333 - 1.625$ $\text{Erro}_{rel} = \frac{|vReal - vAprox|}{vReal} = \frac{2.333-1.625}{2.333}$

$Erro_{abs} = 0.708$ $Erro_{rel} \cong 30\%$

Aproximação utilizando 4 subintervalos:

Cálculo da amplitude: $h = \frac{b-a}{n} = \frac{2-1}{4} = 0.25$

x	y
1	1
1.25	1.5625
1.5	2.25
1.75	3.0625
2	4



$S \cong \sum_{i=1}^{4} f(c_i) \cdot \Delta_{x_i}$

$S \cong 1 \cdot 0.25 + 1.5625 \cdot 0.25 + 2.25 \cdot 0.25 + 3.0625 \cdot 0.25$

$S \cong 1.96875$

Vamos calcular o erro dessa aproximação com relação ao resultado analítico que encontramos.

$\text{Erro}_{abs} = |vReal - vAprox| = 2.333 - 1.96875$ $\text{Erro}_{rel} = \frac{|vReal - vAprox|}{vReal} = \frac{2.333-1.96875}{2.333}$

$Erro_{abs} = 0.36455$ $Erro_{rel} \cong 15.62\%$

5.2 Regra dos Trapézios

5.2.1 Regra do Trapézio Simples

Existe alguma figura geométrica mais eficiente que o retângulo para aproximar a área?

O trapézio tende a acompanhar melhor a curvatura da função e teremos um erro associado um pouco menor.

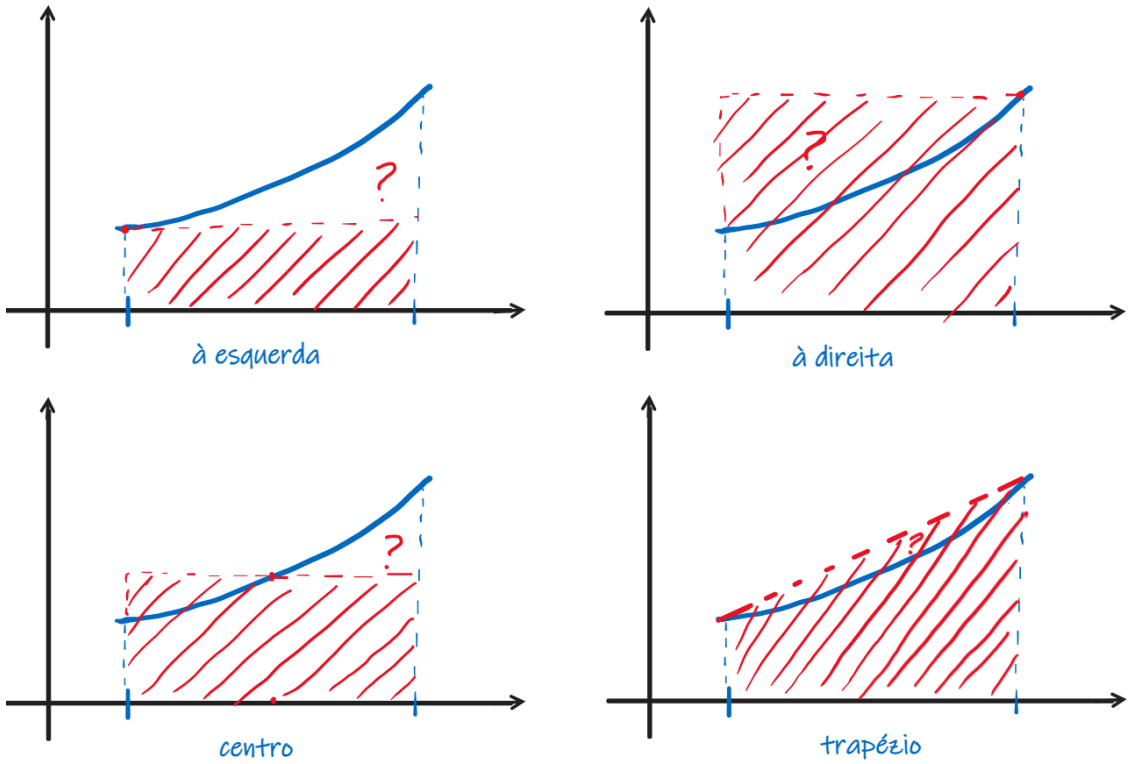


Figura 5.6: Esquemas gráficos comparando as somas de Riemann e Regra dos Trapézios.

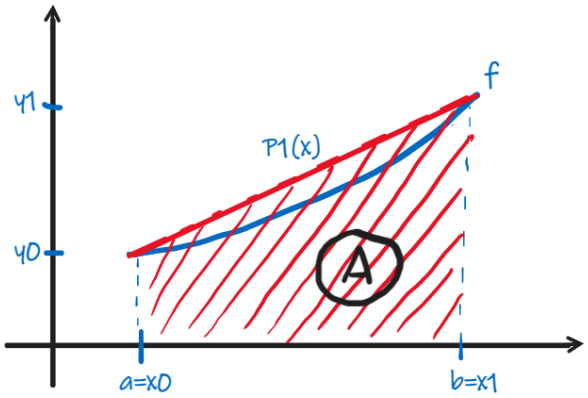


Figura 5.7: Regra do trapézio simples.

A regra do trapézio tenta aproximar o cálculo da área da função pela área do trapézio, um polinômio de grau 1.

$$\int_a^b f(x)dx \cong \int_{x_0}^{x_1} P_1(x)dx$$

$$A_T = \frac{(B+b)h}{2}$$

Estamos calculando uma aproximação da integral utilizando, para isso, uma interpolação de 1º grau, que faz como figura geométrica um trapézio.

$$A_T = (B + b) \times \frac{h}{2} = (y_1 - y_0) \times \frac{h}{2} \text{ onde, } h = x_1 - x_0$$

$\int_a^b f(x)dx \cong \int_{x_0}^{x_1} P_1(x)dx$

Lembrando Gregory-Newton, temos que:

$$\begin{gathered} P_1(x) = y_0 + \Delta y_0 \cdot u(x) \\ u = \frac{x-x_0}{h} \therefore uh = x - x_0 \therefore x = uh + x_0 \therefore dx = h \cdot du \\ \Delta y_0 = y_1 - y_0 \end{gathered} \tag{5.1}$$

Quando $x = x_0$ e $x = x_1$, temos:

$x = x_0 \quad \Rightarrow \quad \frac{x_0-x_0}{h} = 0;$

$x = x_1 \quad \Rightarrow \quad \frac{x_1-x_0}{h} = \frac{h}{h} = 1.$

Então, podemos reescrever a integral como:

$\int_0^1 (y_0 + \Delta y_0 \cdot u)h \cdot du$

$= h \int_0^1 (y_0 + \Delta y_0 \cdot u)du$

$= h \left[uy_0 + \Delta y_0 \frac{u^2}{2} \right]_0^1$

$= h \left[y_0 + \Delta y_0 \cdot \frac{1}{2} \right]$

$= h \left[y_0 + \frac{(y_1-y_0)}{2} \right]$

$= h \left[\frac{2y_0+y_1-y_0}{2} \right]$

$= \frac{h}{2}[y_0 + y_1] \leftarrow$ Regra do Trapézio

Exemplo: Calcular numericamente $\int_1^3 \frac{1}{x}dx$, pela regra do trapézio.

Solução:

Sabemos que o valor real dessa integral é $= \ln x \Big|_1^3 = \ln 3 - \ln 1 = 1.0986$. Vamos guardar esse valor para fazermos algumas análises do método numérico.

x	y
1	1
3	0.3333

$$A_T = (y_0 + y_1) \cdot \frac{h}{2}$$

$$A_T = (1 + 0.3333) \cdot \frac{2}{2}$$

$$A_T = 1.3333$$

$$\text{Erro}_{abs} = 1.3333 - 1.0986$$

$$Erro_{abs} = 0.2342$$

$$Erro_{rel} = \frac{0.2342}{1.0986} = 0.2136$$

$$Erro_{rel} = 21.36\%$$

Podemos melhorar um pouco esse cálculo? Vejamos a regra do trapézio repetida.

5.2.2 Regra do Trapézio repetida

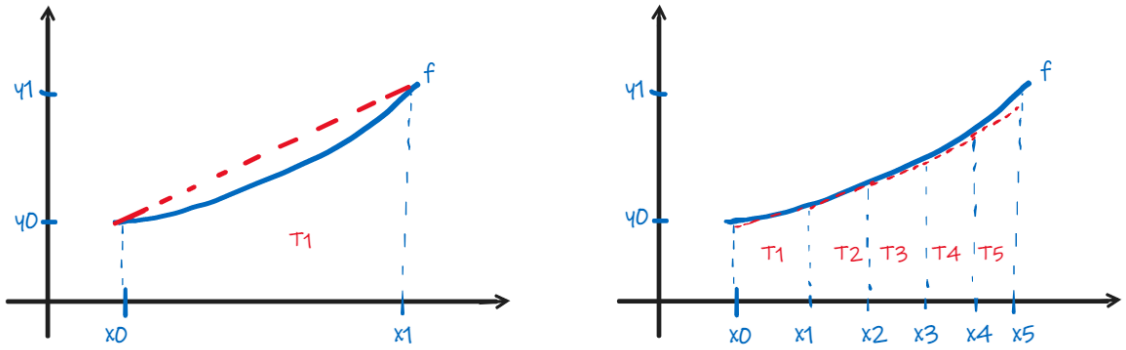
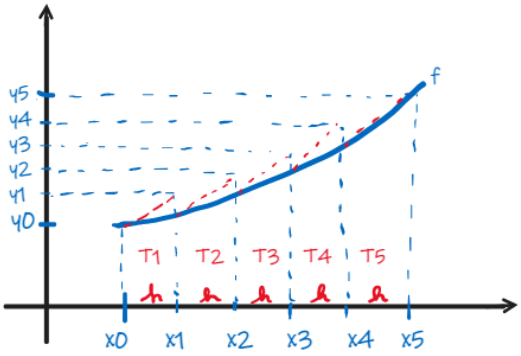


Figura 5.8: Comparativo Regra do trapézio simples e repetida.

A regra repetida vale nos casos em que h é fixo $\rightarrow$ subintervalos iguais. Ou seja, $x_1 - x_0 = x_2 - x_1 = x_3 - x_2, \cdots = h$.

Com várias subdivisões, o erro tende a ser menor porque estamos usando trapézios que acompanham melhor a curvatura da função.

Então:



$$\mathrm{A}_{Total} = A_{T1} + A_{T2} + A_{T3} + A_{T4} + A_{T5}$$

$$A_{Total} = (y_0 + y_1) \cdot \frac{h}{2} + (y_1 + y_2) \cdot \frac{h}{2} + (y_2 + y_3) \cdot \frac{h}{2} + (y_3 + y_4) \cdot \frac{h}{2} + (y_4 + y_5) \cdot \frac{h}{2}$$

$$A_{Total} = \frac{h}{2}\left[y_0 + 2(y_1 + y_2 + y_3 + y_4) + y_5\right]$$

$$A_{Total} = \frac{h}{2}\left[y_0 + 2\sum_{i=1}^{n-1} y_i + y_n\right]$$

Exemplo: Calcular $\int_1^3 \frac{1}{x} dx$, com 4 subintervalos.

Solução:

$$\boxed{\int_1^3 \frac{1}{x} dx = \ln 3 - \ln 1 \cong 1.0986}$$

Para 4 subintervalos $h = \frac{(b-a)}{n} = \frac{3-1}{4} = \frac{1}{2} = 0.5$

x	y
1	1
1.5	0.6666
2	0.5
2.5	0.4
3	0.3333

$$\mathrm{S}_T = \frac{h}{2}\left[y_0 + 2 \cdot \sum_{i=1}^{3} y_i + y_4\right]$$

$$S_T = 0.25\left[1 + 2 \cdot 1.5666 + 0.3333\right]$$

$$S_T = 1.1166$$

$$\mathrm{Erro}_{abs} = 1.1166 - 1.0986$$

$$Erro_{abs} = 0.018$$

$$Erro_{rel} = \frac{0.018}{1.0986}$$

$$Erro_{rel} \cong 0.0164$$

$$Erro_{rel} \cong 1.64\%$$

5.2.3 Implementação do método em Python

Programa 5.1: Regras do trapézio

```
1 def trapezio_simples(x,y):
2     n = len(y)
3     h = x[n-1]-x[0]
4     I = h/2*(y[0]+y[1])
5     return I
6
7
```

```
8  def trapezio_repetido(y,h):
9      n = len(y)
10     s = 0
11     for i in range(1,n-1):
12         s += y[i]
13     I = h/2*(y[0]+2*s+y[n-1])
14     return I
```

As funções do Programa 5.1 implementam os métodos do trapézio simples (função `trapezio_simples(x,y)`) e trapézio repetida (função `trapezio_repetido(y,h)`).

5.3 Método de 1/3 de Simpson

5.3.1 Regra de 1/3 de Simpson Simples

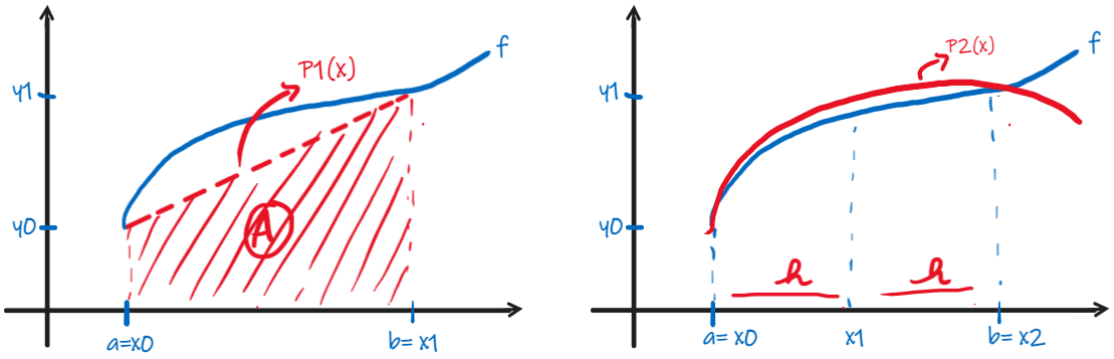


Figura 5.9: Comparativo dos métodos do trapézio e de 1/3 de Simpson.

$\int_a^b f(x)dx \cong \int_{x_0}^{x_2} P_2(x)dx$

$$\boxed{P_2(x) = y_0 + \Delta y_0 \cdot u + \frac{\Delta^2 y_0}{2}(u^2 - u)}$$

$\downarrow$

polinômio de Gregory-Newton de 2 º grau.

$\rightarrow$ lembrando que esse polinômio vai ter amplitude igualmente espaçada, recaindo no polinômio de Gregory-Newton.

Polinômio de Gregory-Newton

$P_n(x) = y_0 + \sum_{i=1}^{n} \left(\frac{\Delta^i y_0}{i!} \prod_{j=0}^{i-1} (u - j) \right)$

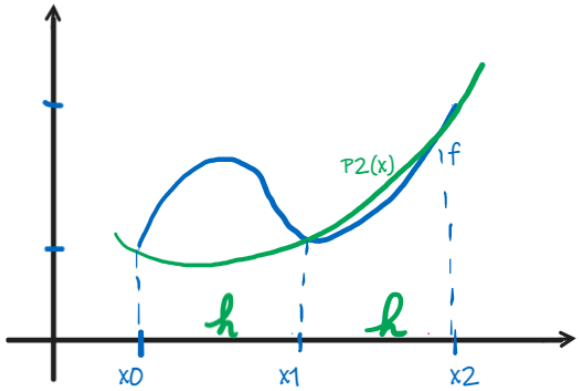
onde, $u = \frac{x - x_0}{b}$ e $\Delta^i y_0 =$ operador diferença finita ascendente

x	$O(0)$	$O(1)$	$O(2)$
x_0	y_0 $\boxed{\Delta^0 y_0}$	$(y_1 - y_0)$ $\boxed{\Delta^1 y_0}$	
x_1	y_1	$(y_2 - y_1)$	$(y_2 - y_1) - (y_1 - y_0)$ $\boxed{\Delta^2 y_0}$
x_2	y_2	$(y_3 - y_2)$	$(y_3 - y_2) - (y_2 - y_1)$
x_3	y_3		

Então temos:

$\int_a^b f(x)dx \cong \int_{x_0}^{x_2} P_2(x)dx$

$$\boxed{\mathrm{P}_2(x) = y_0 + \Delta y_0 \cdot u + \frac{\Delta^2 y_0}{2}(u^2 - u)}$$



$u = \frac{x - x_0}{h} \therefore x = uh + x_0 \therefore dx = hdu$

para $a = x_0 \rightarrow u = \frac{x_0 - x_0}{h} = 0$

para $b = x_2 \rightarrow u = \frac{x_2 - x_0}{h} = \frac{2h}{h} = 2$

$\mathrm{x}_1 - x_0 = x_2 - x_1 = h$

$x_2 - x_0 = 2h$

Fazendo a transformação dos limites e substituindo na Equação de $P_2(x)$, nossa integral fica assim reescrita:

$$\mathrm{I}_2 = \int_0^2 \left[y_0 + u\Delta y_0 + (u^2 - u)\frac{\Delta^2 y_0}{2} \right] h \cdot du$$

$$\mathrm{I}_2 = h \left[uy_0 + \frac{u^2}{2}\Delta y_0 + \left(\frac{u^3}{3} - \frac{u^2}{2} \right) \frac{\Delta^2 y_0}{2} \right]_0^2$$

$$\mathrm{I}_2 = h\left[2y_0 + 2\Delta y_0 + \left(\tfrac{8}{3} - 2\right)\tfrac{\Delta^2 y_0}{2}\right]$$

$$\mathrm{I}_2 = h\left[2y_0 + 2(y_1 - y_0) + \tfrac{1}{3}(y_2 - 2y_1 + y_0)\right]$$

$$\mathrm{I}_2 = h\left[2y_1 + \tfrac{1}{3}y_2 - \tfrac{2}{3}y_1 + \tfrac{1}{3}y_0\right]$$

$$\boxed{\mathrm{I}_2 = \tfrac{h}{3}\left[y_0 + 4y_1 + y_2\right]} \rightarrow \text{Regra 1/3 de Simpson Simples}$$

Para aplicar a regra de Simpson Simples precisamos de 3 pontos, para termos 2 subintervalos.

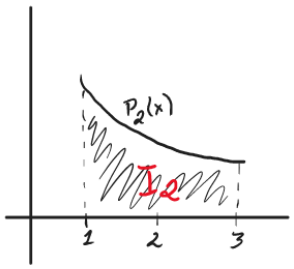
Exemplo: Calcular $\int_1^3 \frac{1}{x}dx$

Solução:

$$\boxed{\int_1^3 \tfrac{1}{x}dx = \ln 3 - \ln 1 \cong 1.0986}$$

Definindo a amplitude (h), temos: $h = \frac{b-a}{n} = \frac{3-1}{2} = 1$.

x	y
1	1
2	0.5
3	0.3333



$I_2 = \frac{h}{3}\left[y_0 + 4y_1 + y_2\right]$

$I_2 = \frac{1}{3}\left[1 + 4 \cdot 0,5 + 0.3333\right]$

$I_2 = 1.1111$

$Erro_{abs} = 1.1111 - 1.0986 = 0.075$

$Erro_{rel} = \frac{0.075}{1.0986} = 1.14\ \%$

5.3.2 Regra de 1/3 de Simpson Repetida

A regra de 1/3 de Simpson com repetição consiste em aplicar repetidamente a regra de Simpson num número par de intervalos.

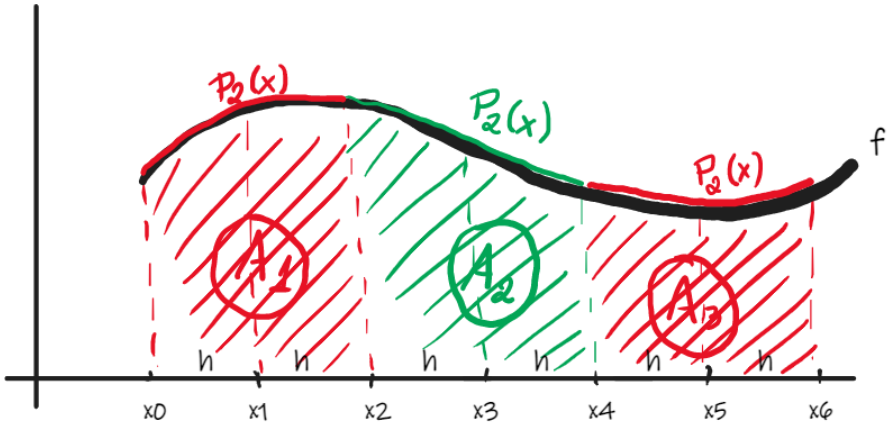


Figura 5.10: Regra de 1/3 de Simpson com repetição.

$$I_T = \frac{h}{3}(y_0 + 4y_1 + y_2) + \frac{h}{3}(y_2 + 4y_3 + y_4) + \frac{h}{3}(y_4 + 4y_5 + y_6)$$

$$I_T = \frac{h}{3}(y_0 + 4(y_1 + y_3 + y_5) + 2(y_2 + y_4) + y_6)$$

$$I_T = \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{3} y_{2i-1} + 2\sum_{i=1}^{2} y_{2i} + y_6\right)$$

$$I_T = \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{n/2} y_{2i-1} + 2\sum_{i=1}^{n/2-1} y_{2i} + y_n\right) \tag{5.2}$$

A Equação 5.2 é a forma geral da regra de 1/3 de Simpson com repetição.

Vejamos a aplicação dessa regra no mesmo exemplo que estamos trabalhando nos métodos anteriores.

Exemplo: Calcular $\int_1^3 \frac{1}{x} dx$ com 6 subintervalos.

Solução:

$$\boxed{\int_1^3 \frac{1}{x} dx = \ln 3 - \ln 1 \cong 1.0986}$$

Cálculo da amplitude h para dividir em 6 subintervalos.

$$h = \frac{3-1}{6} = \frac{1}{3} = 0.3333$$

x	y
1	1
1.3333	0.75
1.6666	0.6
2	0.5
2.3333	0.4285
2.6666	0.375
3	0.3333

$$I_T = \frac{h}{3}\left(y_0 + 4(y_1 + y_3 + y_5) + 2(y_2 + y_4) + y_6\right)$$

$$I_T = \frac{0.3333}{3}(1 + 4 \cdot (0.75 + 0.5 + 0.375) + 2 \cdot (0.6 + 0.4285) + 0.3333)$$

$$I_T = 1.0988$$

$$Erro_{abs} = 1.0986 - 1.0988 = 0.0002$$

$$Erro_{rel} = \frac{0.0002}{1.0986} \cong 0.018\%$$

Quando dividimos o intervalo de integração em uma quantidade maior de subintervalos, o cálculo numérico se aproxima mais do valor real.

5.4 Método de 3/8 de Simpson

5.4.1 Regra de 3/8 de Simpson Simples

Motivação em relação a 1/3 de Simpson

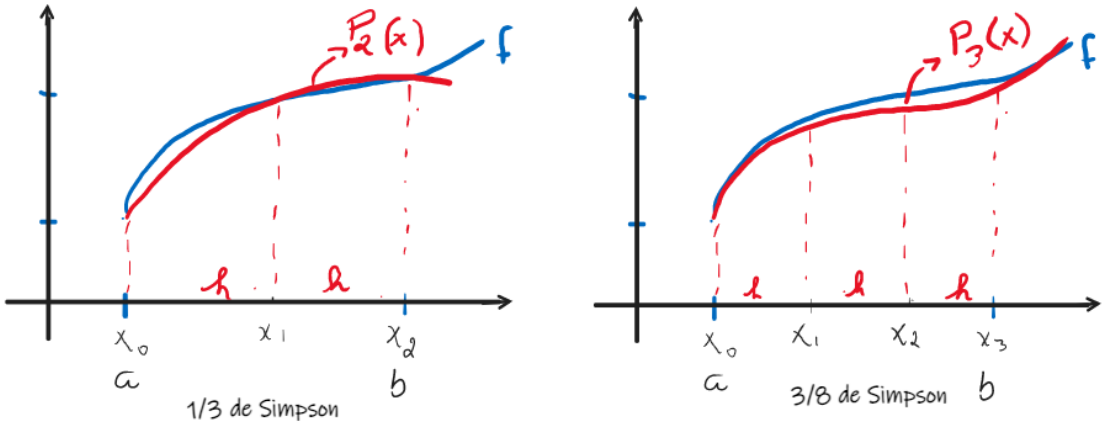


Figura 5.11: Comparativo das regras de 1/3 de Simpson e de 3/8 de Simpson.

No método 1/3 de Simpson, para calcular a integral da função f no intervalo $[a, b]$, dividimos em 2 subintervalos com amplitude h iguais, ficando com 3 pontos e, a partir daí, fazemos uma interpolação de 2 º grau.

Já no caso do método de 3/8 de Simpson, este tenta melhorar essa aproximação (nem sempre isso é possível), dividindo em 3 subintervalos de amplitude h iguais, ficando com 4

pontos, fazendo a aproximação por um polinômio de 3 º grau.

$$\int_a^b f(x)dx \cong \int_{x_0}^{x_3} P_3(x)dx$$

Neste texto faremos a demonstração usando um polinômio interpolador de Gregory-Newton de 3 ºgrau. Então:

$$P_3(x) = y_0 + \Delta y_0 \cdot u + \frac{\Delta^2 y_0}{2}(u^2 - u) + \frac{\Delta^3}{6}(u^3 - 3u^2 + 2u)$$

Demonstração:

Seja $u = \frac{x - x_0}{h}$:

Para $a = x_0 \therefore \frac{x_0 - x_0}{h} = 0$

Para $b = x_3 \therefore \frac{x_3 - x_0}{h} = \frac{3h}{h} = 3$

$x = uh + x_0$

$dx = h \cdot du$

Então, podemos reescrever nossa integral como:

$\mathrm{I} = \int_0^3 \left[y_0 + u\Delta y_0 + (u^2 - u)\frac{\Delta^2 y_0}{2} + (u^3 - 3u^2 + 2u)\frac{\Delta^3 y_0}{6} \right] h \cdot du$

$\mathrm{I} = \mathrm{h} \left[uy_0 + \frac{u^2}{2}\Delta y_0 + \left(\frac{u^3}{3} - \frac{u^2}{2}\right)\frac{\Delta^2 y_0}{2} + \left(\frac{u^4}{4} - \frac{3u^3}{3} + \frac{2u^2}{2}\right)\frac{\Delta^3 y_0}{6} \right]_0^3$

$\mathrm{I} = \mathrm{h} \left[3 \cdot y_0 + \frac{9}{2}\Delta y_0 + \left(\frac{27}{3} - \frac{9}{2}\right)\frac{\Delta^2 y_0}{2} + \left(\frac{81}{4} - 27 + 9\right)\frac{\Delta^3 y_0}{6} \right]$

$\mathrm{I} = \mathrm{h} \left[3 \cdot y_0 + \frac{9}{2}(y_1 - y_0) + \frac{9}{4}(y_2 - 2y_1 + y_0) + \frac{3}{8}(y_3 - 3y_2 + 3y_1 - y_0) \right]$

$\mathrm{I} = \frac{h}{8} \left[24y_0 + 36(y_1 - y_0) + 18y_2 - 36y_1 + 18y_0 + 3y_3 - 9y_2 + 9y_1 - 3y_0 \right]$

$\mathrm{I} = \frac{h}{8} \left[3y_0 + 9y_1 + 9y_2 + 3y_3 \right]$

$$I = \frac{3h}{8} \left[y_0 + 3y_1 + 3y_2 + y_3 \right] \tag{5.3}$$

A Equação 5.3 é a fórmula da regra de 3/8 de Simpson Simples, obtida por meio de um polinômio interpolador de Gregory-Newton de 3 º grau.

Vejamos a aplicação dessa regra em um exemplo.

Exemplo: Calcular $\int_1^3 \frac{1}{x} dx$

Solução:

$$\boxed{\int_1^3 \frac{1}{x} dx = \ln 3 - \ln 1 \cong 1.0986}$$

Cálculo da amplitude h para dividir em 3 subintervalos.

$$h = \frac{b-a}{3} = \frac{3-1}{3} = \frac{2}{3} = 0.66667$$

x	y
1	1
1.6666	0.6
2.3333	0.4285
3	0.3333

$$I = \frac{3 \cdot 0.6666}{8} [1 + 3 \cdot 0.6 + 3 \cdot 0.4285 + 0.3333]$$

$$I \cong 1.1047$$

$$Erro_{abs} = 1.1047 - 1.0986 = 0.0061$$

$$Erro_{rel} = \frac{0.0061}{1.0986} = 0.56\ \%$$

5.4.2 Regra 3/8 de Simpson com repetição

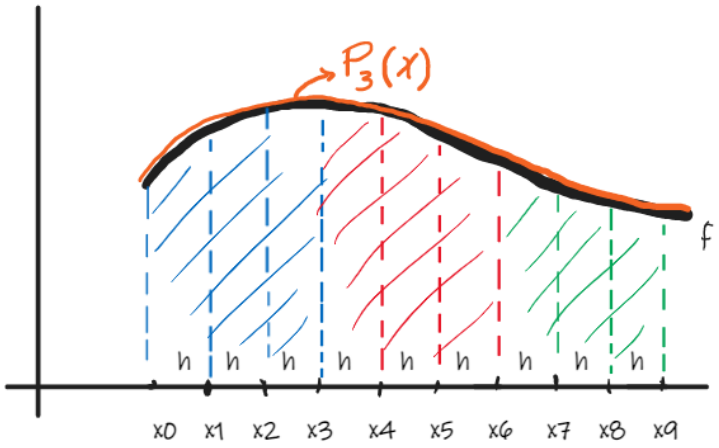


Figura 5.12: Regra de 3/8 de Simpson com repetição.

Nesse caso, para implementarmos a regra de 3/8 de Simpson com repetição devemos ter uma quantidade de subintervalos múltiplos de 3.

Para demonstrar a fórmula, vamos repetir a Equação 5.3 para cada trecho da Figura 5.12.

$$\mathrm{I} = \tfrac{3h}{8}\,[y_0 + 3y_1 + 3y_2 + y_3] + \tfrac{3h}{8}\,[y_3 + 3y_4 + 3y_5 + y_6] + \tfrac{3h}{8}\,[y_6 + 3y_7 + 3y_8 + y_9]$$

$$\mathrm{I} = \tfrac{3h}{8}\,[y_0 + 3(y_1 + y_2 + y_4 + y_5 + y_7 + y_8) + 2(y_3 + y_6) + y_9]$$

$$I = \frac{3h}{8}\left[y_0 + 3\sum_{i=0}^{\frac{n}{3}-1}(y_{3i+1} + y_{3i+2}) + 2\sum_{i=0}^{\frac{n}{3}-1}(y_{3i}) + y_n\right] \tag{5.4}$$

A Equação 5.4 é a fórmula geral da regra de 3/8 de Simpson, com repetição, obtida a partir do polinômio interpolador de Gregory-Newton, de 3ºgrau.

Exemplo: Calcula $\int_1^3 \frac{1}{x}dx$, com 6 subintervalos.

Solução:

$$\boxed{\int_1^3 \frac{1}{x}dx = \ln 3 - \ln 1 \cong 1.0986}$$

Cálculo da amplitude h para dividir em 6 subintervalos:

$$h = \frac{b-a}{6} = \frac{3-1}{6} = \frac{2}{6} = 0.3333$$

x	y
1	1
1.3333	0.75
1.6666	0.6
2	0.5
2.3333	0.4285
2.6666	0.375
3	0.3333

$\mathrm{I} = \frac{3h}{8}\left[y_0 + 3\sum_{i=0}^{1}(y_{3i+1} + y_{3i+2}) + 2\sum_{i=0}^{1}(y_{3i}) + y_6\right]$

$I = \frac{3h}{8}\,[y_0 + 3(y_1 + y_2 + y_4 + y_5) + 2(y_3) + y_6]$

$I = \frac{3\cdot 0.3333}{8}\,[1 + 3(0.75 + 0.6 + 0.4285 + 0.375) + 2\cdot 0.5 + 0.3333]$

$I \cong 1.0991$

$Erro_{abs} = 0.0005$

$Erro_{rel} = 0.0495\%$

5.4.3 Implementação dos métodos de Simpson em Python

Programa 5.2: Métodos de Simpson

```
1  def simpson_1(fx,n,h):
2      """
3      :param fx: lista de valores de f(x)
4      :param n: numero de pontos ou o numero de subintervalos
5      :param h: amplitude
6      :return: integral f(x) no intervalo
7      """
8      if (n % 3 == 0 or n%3 == 1):
9          I = 0
10         I = I + fx[0]
11         s = 0
12         for i in range(1, int((n - 1) / 2)+1):
13             s += fx[(2 * i) -1]
14         s = 4 * s
15         I += s
16         s = 0
17         for i in range(1, int((n - 1) / 2)):
18             s += fx[2 * i]
19         s = 2 * s
20         I += s
21         I += fx[n - 1]
22         I = h / 3 * I
23         return I
24     else:
25         return "ERRO: Nao foi possivel realizar a aproximacao
26             com o metodo 1/3 de Simpson."
```

```
27
28
29 def simpson_38(fx,n,h):
30     """
31     :param fx: lista dos valores de f(x)
32     :param n: numero de pontos
33     :param h: amplitude
34     :return: integral f(x) no intervalo [a,b]
35     """
36     if((n-1)%3==0):
37         I = 0
38         I = I + fx[0]
39         s = 0
40         for i in range(0,int((n-1)/3)):
41             #print((3*i)+1, (3*i)+2)
42             s += fx[(3*i)+1] + fx[(3*i)+2]
43         s = 3*s
44         I += s
45         s = 0
46         for i in range(1,int((n-1)/3)):
47             #print((3 * i))
48             s += fx[3*i]
49         s = 2*s
50         I+=s
51         I+=fx[n-1]
52         I = (3*h)/8 * I
53         return I
54     else:
55         return "ERRO: Nao foi possivel realizar a aproximacao
```

```
56              com o metodo 3/8 de Simpson.\n" \
57                 "\t  O numero de subintervalos deve
58                 ser multiplo de 3."
```

5.5 Quadratura Gaussiana

Vimos que nas fórmulas de Newton-Cotes os pontos $x_0, \ldots, x_n$ são igualmente espaçados e sobre esses mesmos pontos construímos os polinômios de Lagrange ou de Gregory-Newton. Utilizando polinômios de Lagrange teremos a forma:

$$\int_a^b f(x)dx = A_0 f(x_0) + A_1 f(x_1) + \cdots + A_n f(x_n)$$

onde:

$\mathrm{A}_K = \int_a^b L_K(x)dx,\ k = 0, 1, \ldots, n$

> **Importante!** Alguns autores podem se referir ao método "Quadratura Gaussiana" como "Fórmula de Gauss-Legendre".

O fato dos pontos serem igualmente espaçados vão simplificar os cálculos. Mas, se não impusermos a condição de espaçamento constante podemos obter fórmulas que,

- Fornecem uma maior exatidão;
- Usando o mesmo número de pontos.

A Quadratura Gaussiana vai possuir a mesma forma:

$$\int_a^b f(x)dx = A_0 f(x_0) + A_1 f(x_1) + \cdots + A_n f(x_n)$$

onde:

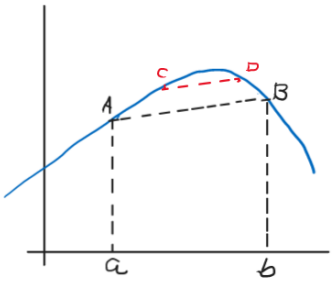
$\mathrm{A}_K = \int_a^b L_K(x)dx,\ k = 0, 1, \ldots, n$

mas que serão exatas para polinômios de grau $\leqslant 2n + 1$.

5.5.1 Quadratura Gaussiana - Fórmula para 2 pontos

Para fórmula para 2 pontos temos grau 1, ou seja, $n = 1$.

A Regra do Trapézio é baseada em um polinômio de grau 1 que passa por A e B.



Poderíamos escolher os pontos C e D de tal modo que a área do trapézio seja a mais próxima possível da área sob a curva.

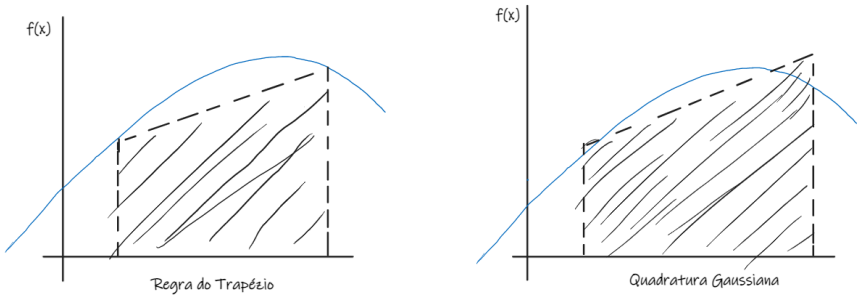


Figura 5.13: Comparativo da Regra do Trapézio com Quadratura Gaussiana.

Para $n = 1$, precisamos descobrir os valores de $x_0, x_1, A_0 e A_1$, tais que:

$$\int_a^b f(x)dx = A_0 f(x_0) + A_1 f(x_1)$$

seja exata para polinômios de grau 3.

Para facilitar os cálculos, iremos considerar o intervalo $[a, b] = [-1, 1]$. Então, para isso vamos fazer a mudança de variável $x \to t$.

Dizer que a fórmula é exata para polinômios de grau $\leqslant 3$ significa dizer que a fórmula é exata para:

$$g_0(t) = 1, g_1(t) = t, g_2(t) = t^2 \text{ e } g_3(t) = t^3$$

Então precisamos de valor de t que satisfaça essas 4 equações:

$\int_{-1}^{1} 1dt = A_0 g_0(t_0) + A_1 g_0(t_1)$

$\boxed{2 = A_0 + A_1}$

$\int_{-1}^{1} t dt = A_0 g_1(t_0) + A_1 g_1(t_1)$

$\boxed{0 = A_0 t_0 + A_1 t_1}$

$\int_{-1}^{1} t^2 dt = A_0 g_2(t_0) + A_1 g_2(t_1)$

$\boxed{\frac{2}{3} = A_0 t_0^2 + A_1 t_1^2}$

$\int_{-1}^{1} t^3 dt = A_0 g_3(t_0) + A_1 g_3(t_1)$

$\boxed{0 = A_0 t_0^3 + A_1 t_1^3}$

$$\begin{cases} A_0 + A_1 = 2 \\ A_0 t_0 + A_1 t_1 = 0 \\ A_0 t_0^2 + A_1 t_1^2 = \frac{2}{3} \\ A_0 t_0^3 + A_1 t_1^3 = 0 \end{cases}$$

$\boxed{t_0 = -\sqrt{3}/3}$ $\boxed{t_1 = \sqrt{3}/3}$ e $\boxed{A_0 = A_1 = 1}$ $\rightarrow$ valores para 2 pontos.

Para o caso de um intervalo $[a, b]$ genérico efetuamos a mudança de variáveis.

Para $t \in [-1, 1]$ corresponde a $x \in [a, b]$.

$x = x(t) = \frac{b-a}{2} t + \frac{a+b}{2}$

$t = -1 \therefore \frac{-b+1}{2} + \frac{a+b}{2} \therefore x = \frac{2a}{2} \therefore x = a$

$t = 1 \therefore \frac{b-a}{2} + \frac{a+b}{2} \therefore x = b$

$dx = \frac{b-a}{2}dt$

Então, dada $f(x)$ e os limites a e b, queremos integrar $f(x)$ usando:

$$I = \frac{b-a}{2}\left(A_0 f(x_0) + A_1 f(x_1)\right)$$

Para isso precisamos descobrir x_0 e x_1 a partir de t_0 e t_1. Onde $t_0 = -0.57735$, $t_1 = 0.57735$ (com 5 casas decimais). Calcula-se $f(x_0)$ e $f(x_1)$ e com $A_0 = 1$ e $A_1 = 1$ descobre-se o valor da integral I.

$$I = \frac{b-a}{2}\left(A_0 f(x_0) + A_1 f(x_1)\right) \rightarrow \text{fórmula para 2 pontos}$$

Podemos usar mais pontos. Neste caso a fórmula pode ser escrita como:

$$I_n = \frac{b-a}{2}\sum_{i=1}^{n} A_i f(x_i)$$

Exemplo: Calcular a quadratura gaussiana:

$$\int_1^5 \left(2x^3 + 3x^2 + 6x + 1\right) dx$$

Resultado analítico:

$\mathrm{I} = \frac{6x^4}{4} + \frac{6x^3}{3} + \frac{6x^2}{2} + x\Big|_1^5$

$\mathrm{I} = \frac{x^4}{2} + x^3 + 3x^2 + x\Big|_1^5$

$\mathrm{I} = 512$

Resultado usando aproximação pela Quadratura Gaussiana:

① $\boxed{I = \frac{b-a}{2}\left(A_0 f_(x_0) + A_1 f(x_1)\right)}$

② $\boxed{x = x(t) = \frac{b-a}{2}t + \frac{a+b}{2}}$

$$x_i = \frac{5-1}{2}t_i + \frac{5+1}{2} \therefore x_i = 2t_i + 3$$

i	t_i	x_i	$f(x_i)$	A_i
0	-0.5774	1.8453	34.8542	1
1	0.5774	$4,1547$	221.1458	1

$$\mathrm{I}_2 = \tfrac{5-1}{2}\,(1 \times 34.8542 + 221.1458)$$

$\boxed{\mathrm{I}_2 = 512}$ $\rightarrow$ resultado exato

Implementação do Método e testes

O Programa 5.3 é uma adaptação escrita em Python 3 do algoritmo de CAMPOS (2007).

Programa 5.3: Integração numérica pelo método de Gauss-Legendre

```
1  import math
2  import numpy as np
3  import pandas as pd
4
5  def f(x):
6      # f(x) deve ser reescrita conforme a funcao
7      # que se deseja integrar
8      return 2*pow(x,3)+3*pow(x,2)+6*x + 1
9
10 def pes_abs_gauss(n):
11     """
12     Objetivo: calcular pesos e abscissas para
13         a formula de Gauss-Legendre
14     :param n: numero de pontos
15     :return: pesos, abscissas e condicao de erro, sendo:
16     CondErro = 0 se nao houve erro  (n>=1) e
17     CondErro = 1 se n < 1
18     """
```

```
19      CondErro = 0
20      m = math.trunc(0.5*(n+1))
21      T = np.zeros(m)
22      A = np.zeros(m)
23      if(n<1):
24          CondErro = 1
25      for i in range(m):
26          z = math.cos(math.pi*((i+1)-0.25)/(n+0.5))
27          while(1):
28              p1 = 1
29              p2 = 0
30              for j in range(1,n+1):
31                  p3 = p2
32                  p2 = p1
33                  #polinomio de Legende no ponto z
34                  p1 = ((2*j-1)*z*p2-(j-1)*p3)/j
35              # derivada do polinomio de Legendre no ponto z
36              pp = n * (z*p1-p2)/(pow(z,2)-1)
37              z1 = z
38              # metodo de Newton para calcular os
39              # zeros do polinomio
40              z = z1 - p1/pp
41              if (abs(z-z1)< pow(10, -15)):
42                  break
43          T[m-1-i]=z # abscissa
44          A[m-1-i]=2/((1-pow(z,2))*pow(pp,2)) # peso
45          # somente as raizes nao negativas sao calculadas
46          # devido a simetria
47      return A, T, CondErro
```

```
48
49 def quadratura_gauss(a, b, n, imprime_tabela=True):
50     """
51     Objetivo: Integrar uma funcao pelo metodo de Gauss-Legendre
52     :param a: limite inferior
53     :param b: limite superior
54     :param n: numero de pontos
55     :return: valor da Integral e Condicao de erro, sendo:
56         CondErro = 0 se nao houve erro (n>=1) e
57         CondErro = 1 se n < 1
58     """
59     Integral = 0
60     # calculo dos pesos e abscissas
61     Avet, Tvet, CondErro = pes_abs_gauss(n)
62
63     if(CondErro>0):
64         return
65     # calculo da integral
66
67     e1 = (b-a)/2
68     e2 = (a+b)/2
69
70     if(n%2==0):
71         k_ = np.arange(np.trunc(n/2),dtype=int)
72         k=np.pad(k_, (int(np.trunc(n/2)), 0), "symmetric",
73             reflect_type="odd")
74         sinal=np.ones(n)
75         for s in range(int(n/2)):
76             sinal[s]=-1
```

```
77          #print(sinal)
78      else:
79          k_ = np.arange((np.trunc(n / 2) + 1), dtype=int)
80          k = np.pad(k_, (int(np.trunc(n / 2)), 0), "reflect",
81              reflect_type="odd")
82          sinal = np.sign(k)
83      somas = np.zeros([n,5], dtype=float)
84      for i in range(n):
85          t = sinal[i] * Tvet[abs(k[i])]
86          x = e1 * t + e2
87          y = f(x) # avaliar a funcao integrando em x
88          c = Avet[abs(k[i])]
89          Integral = Integral + y * c
90          somas[i]=[i,t,x,y,c]
91      #fim for
92
93      if (imprime_tabela):
94          somas_table = pd.DataFrame(somas,
95              columns=["i", "t", "x", "y", "c"])
96          print(somas_table)
97
98      Integral = e1 * Integral
99
100     return Integral, CondErro, somas_table
101
102 if(__name__=="__main__"):
103     integral, erro, tabela = quadratura_gauss(1,5,2)
104     print("Integral = {} Erro:{}".format(integral, erro))
105     print(tabela)
```

A saída do Programa 5.3, para a função $f(x)$ implementada é o seguinte,

```
-----  Quadratura de Gauss-Legendre  -----
     i        t         x           y    c
0  0.0 -0.57735  1.845299   34.854157  1.0
1  1.0  0.57735  4.154701  221.145843  1.0
Integral = 512.0000000000001 Erro:0
```

Vamos testar o programa para outras funções $f(x)$, limites e números de pontos.

Teste 1: Calcular $\int_0^{\pi} sen(x)dx$ com 5 e 6 pontos.

- Para n = 5

```
# Alteramos a f(x)
def f(x):
    return math.sin(x)
# e a chamada...
if(__name__=="__main__"):
        print("-----  Quadratura de Gauss-Legendre  -----")
    integral, erro, tabela = quadratura_gauss(0,math.pi,5)
    print("Integral = {} Erro:{}".format(integral, erro))


# Produz a seguinte saída
-----  Quadratura de Gauss-Legendre  -----
     i         t         x         y         c
0  0.0 -0.906180  0.147372  0.146839  0.236927
1  1.0 -0.538469  0.724971  0.663114  0.478629
2  2.0  0.000000  1.570796  1.000000  0.568889
3  3.0  0.538469  2.416622  0.663114  0.478629
4  4.0  0.906180  2.994220  0.146839  0.236927
Integral = 2.000000110284472 Erro:0
```

- Para n = 6

```
# A chamada
if(__name__=="__main__"):
 print("-----  Quadratura de Gauss-Legendre  -----")
    integral, erro = quadratura_gauss(0,math.pi,6)
    print("Integral = {} Erro:{}".format(integral, erro))


# Produz a seguinte saída
-----  Quadratura de Gauss-Legendre  -----
     i         t         x         y         c
0  0.0 -0.932470  0.106077  0.105878  0.171324
1  1.0 -0.661209  0.532171  0.507405  0.360762
2  2.0 -0.238619  1.195974  0.930573  0.467914
3  3.0  0.238619  1.945618  0.930573  0.467914
4  4.0  0.661209  2.609422  0.507405  0.360762
5  5.0  0.932470  3.035516  0.105878  0.171324
Integral = 1.999999999477271 Erro:0
```

Teste 2: Verificar que $\pi = \int_0^1 \frac{4}{1+x^2}$ com n = 10 utilizando o Programa 5.3.

```
# Alteramos a f(x)
def f(x):
    return 4/(1+pow(x,2))


# E a chamada


if(__name__=='__main__'):
 print("-----  Quadratura de Gauss-Legendre  -----")
    integral, erro, tabela = quadratura_gauss(0,1,10)
    print('Integral = {} Erro:{}'.format(integral, erro))
```

```
# Produz a seguinte saída

-----  Quadratura de Gauss-Legendre  -----

     i         t         x         y         c

0  0.0 -0.973907  0.013047  3.999319  0.066671

1  1.0 -0.865063  0.067468  3.981875  0.149451

2  2.0 -0.679410  0.160295  3.899796  0.219086

3  3.0 -0.433395  0.283302  3.702812  0.269267

4  4.0 -0.148874  0.425563  3.386663  0.295524

5  5.0  0.148874  0.574437  3.007568  0.295524

6  6.0  0.433395  0.716698  2.642609  0.269267

7  7.0  0.679410  0.839705  2.345898  0.219086

8  8.0  0.865063  0.932532  2.139478  0.149451

9  9.0  0.973907  0.986953  2.026264  0.066671

Integral = 3.1415926535900325 Erro:0
```

5.6 Exercícios

1. Calcular $\int_2^5 \frac{1}{xlog_e(x)}dx$ com $m = 6$ pelas regras abaixo.

   (a) Trapézio.

   (b) 1/3 de Simpson.

   (c) 3/8 de Simpson.

   (d) Compare esses três resultados com o valor exato $log_e(log_e(5)) - log_e(log_e(2)) \cong 0.84240$.

2. Seja a função $f(x) = 10^x$

   (a) Ache o polinômio de Newton de grau 2 que passa pelos pontos das abscissas -1,0 e 1.

   (b) Integre, analiticamente, o polinômio obtido no item (a) no intervalo $[-1, 1]$.

(c) Calcule $\int_{-1}^{1} 10^x dx$ utilizando a regra do 1/3 de Simpson com 2 subintervalos.

3. Calcule $\int_0^2 e^x dx$ com $E < 2 \times 10^{-3}$ usando uma das fórmulas de Newton-Cotes com o menor número de subintervalos.

4. Implemente em Python os programas para algoritmos das fórmulas de Newton-Cotes e Quadratura Gaussiana. Calcule $\int_0^{\pi} sen(x)$ usando algum método numérico de Newton Cotes e com o método de Gauss (quadratura). Utilize polinômios de $n = 2$ e $n = 3$ com 6 subintervalos. Compare os resultados.

5. Usando o Programa 5.3, calcule as integrais a seguir, com $Toler = 10^{-10}$ e $IterMax = 10$.

   (a) $\int_0^{\pi} \sqrt{1 + cos(x)}$ dx

   (b) $\int_2^5 \frac{1}{xlog_e(x)} dx$

   (c) $\int_0^{\pi} e^{1-x^2} sex(10x) dx$

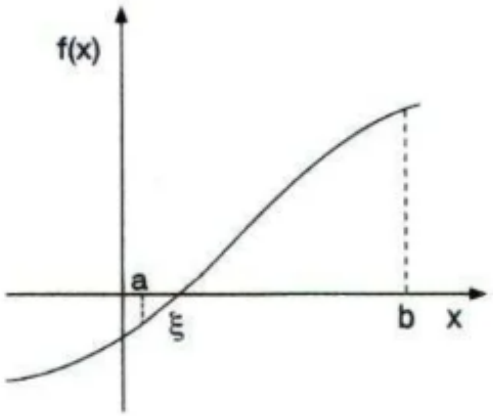
   (d) $\int_{-\pi}^{\pi} cos(2 + cos^2(x)) dx$

Capítulo 6

Raízes de Equações

6.1 Introdução

Definição: um zero (ou raiz) de uma função $f(x)$ é um número real ξ tal que $f(\xi) = 0$.



Motivação para aplicação de métodos numéricos para cálculo de raízes de funções.

a) polinômio de grau elevado;
b) funções "complicadas".

Temos duas fases para a aplicação de métodos para o cálculo de raízes:

Fase 1: Isolamento de raízes.
Fase 2: Refinamento.

6.2 Isolamento de Raízes

Isolamento: análise teórica da função, buscando garantir que só existe uma raiz no intervalo de busca.

Ferramentas:

i) Teorema do anulamento (ou Teorema de Bolzano);
ii) Corolário do teorema;
iii) Avaliação do polinômio pelo Método de Horner;
iv) Tabelas (tabelamento);
v) Gráficos.

6.2.1 Equações algébricas

Seja uma equação algébrica de grau n; $n \geq 1$, escrita na forma de potências:

$$P(x) = c_n x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_2 x^2 + c_1 x^1 + c_0 = 0 \tag{6.1}$$

com os coeficientes c_i sendo reais e $c_n \neq 0$.

Para obter o valor de um polinômio $P(x)$ em um ponto $x = a$, usualmente se faz:

$$P(a) = c_n a^n + c_{n-1}a^{n-1} + c_{n-2}a^{n-2} + \cdots + c_2 a^2 + c_1 a^1 + c_0$$

Portanto, para avaliar $P(x)$ de grau n, em $x = a$, são necessárias $n(n+1)/2$ multiplicações e n adições.

No entanto, uma maneira mais eficiente de avaliar um polinômio é o **método de Horner**, que consiste em reescrever o polinômio de forma a evitar potências.

Por exemplo, avaliar $P(x) = 3x^5 - 2x^4 + 5x^3 + 7x^2 - 3x + 1$ em $x = 2$, usando o processo de Horner.

$P(x) = ((((3x - 2)x + 5)x + 7)x - 3)x + 1,$

$P(2) = ((((3 \times 2 - 2) \times 2 + 5) \times 2 + 7) \times 2 - 3) \times 2 + 1 = 127$

O processo de Horner requer apenas n multiplicações e n adições para avaliar um polinômio de grau n. O Algoritmo 8, Campos (2007), foi implementado em Python e originou o Programa 6.1.

Programa 6.1: Função de avaliação de polinômios pelo algoritmo de Horner

```
1  def horner(n, c, a):
2      """
3      Objetivo: avaliar um polinomio de grau 'n' no ponto 'a'
4      :param n: grau do polinomio
5      :param c: lista de coeficientes
6      :param a: ponto a ser avaliado
7      :return: y -> ordenada (P(a))
8      """
9      y = c[0]
10     for i in range(1,n+1):
11         y = y *a + c[i]
12     return y
```

Como exemplo de aplicação desse programa, vamos avaliar o polinômio $P(x) = 3x^5 - 2x^4 + 5x^3 + 7x^2 - 3x + 1$ no ponto $x = 2$ (Programa 6.2).

Programa 6.2: Exemplo de aplicação do Programa 6.1

```
1  if(__name__ == "__main__"):
2      n = 5
3      c = [3,-2,5,7,-3,1]
4      a = 2
5      y = horner(n,c,a)
6      print("y = {}".format(y))
```

O resultado é mostrado a seguir.

```
# os parâmetros de entrada
n = 5
c = [3,-2,5,7,-3,1]
a = 2
```

```
# produzem o resultado
y = 127
```

Propriedades Gerais das equações algébricas

Teorema 1: Uma equação algébrica de grau n tem exatamente n raízes, reais ou complexas, contando cada raiz de acordo com a sua multiplicidade.

Exemplo: Seja:

$P(x) = x^4 + 2x^3 - 12x^2 + 14x - 5 \rightarrow P(1) = 0$

$P'(x) = 4x^3 + 6x^2 - 24x + 14 \rightarrow P'(1) = 0$

$P''(x) = 12x^2 + 12x - 24 \rightarrow P''(1) = 0$ e

$P'''(x) = 24x + 12 \rightarrow P'''(1) = 36.$

Portanto, $\xi = 1$ é uma raiz de multiplicidade $m = 3$. Considerando também que $P(-5) = 0$, o polinômio de grau 4 pode ser escrito na forma fatorada $P(x) = (x-1)^3(x+5)$.

Teorema 2: Se os coeficientes de uma equação algébrica forem reais, então suas raízes complexas serão complexos conjugados em pares, ou seja, se $\xi_1 = a + bi$ for uma raiz de multiplicidade m, então $\xi_2 = a - bi$ também será uma raiz e com a mesma multiplicidade.

Exemplo: As raízes de $P(x) = x^2 - 4x + 13 = 0$ são:

$$\xi = \frac{4 \pm \sqrt{(-4)^2 - 4 \times 1 \times 13}}{2} \rightarrow \begin{cases} \xi_1 = 2 + 3i \\ \xi_2 = 2 - 3i \end{cases}$$

Corolário 1: Uma equação algébrica de grau ímpar com coeficientes reais tem, no mínimo, uma raiz real.

Exemplo: As raízes da equação $P(x) = x^3 - 9x^2 + 33x - 65 = 0$ são $\xi_1 = 5$, $\xi_2 = 2 + 3i$ e $\xi_3 = 2 - 3i$. Portanto, essa equação de grau 3 tem uma raiz real.

Relação entre raízes e coeficientes

Se $\xi_i, i = 1, 2, \ldots, n$ forem raízes de $P(x) = 0$, então ela pode ser escrita na forma fatorada

$$P(x) = c_n(x - \xi_1)(x - \xi_2) \ldots (x - \xi_n) = 0$$

Multiplicando os fatores e comparando a expressão obtida com $P(x) = 0$ escrita na forma de potências e aplicando a condição de igualdade das equações algébricas, obtemos

expressões que relacionam os coeficientes da equação algébrica com suas raízes. Essas relações são conhecidas como **relações de Girard**.

$$\xi_1 + \xi_2 + \cdots + \xi_n = -\frac{c_{n-1}}{c_n},$$
$$\xi_1\xi_2 + \xi_1\xi_3 + \cdots + \xi_1\xi_n + \xi_2\xi_3 + \cdots + \xi_2\xi_n + \cdots + \xi_{n-1}\xi_n = \frac{c_{n-2}}{c_n},$$
$$\xi_1\xi_2\xi_3 + \xi_1\xi_2\xi_4 + \cdots + \xi_1\xi_2\xi_n + \xi_1\xi_3\xi_4 + \cdots + \xi_{n-2}\xi_{n-1}\xi_n = -\frac{c_{n-3}}{c_n},$$

...

$$\xi_1\xi_2\xi_3 \ldots \xi_n = (-1)^n \frac{c_0}{c_n}$$

Exemplo: As raízes da equação $P(x) = x^3 - 9x^2 + 33x - 65 = 0$ são $\xi_1 = 5$, $\xi_2 = 2 + 3i$ e $\xi_3 = 2 - 3i$. Assim, as relações de Girard são:

$$5 + (2 + 3i) + (2 - 3i) = 9 = -\frac{-9}{1},$$

$$5(2 + 3i) + 5(2 - 3i) + (2 + 3i)(2 - 3i) = 33 = \frac{33}{1} \text{ e}$$

$$5(2 + 3i)(2 - 3i) = 65 = (-1)^3\frac{-65}{1}$$

Limites das raízes reais

Uma equação algébrica na forma (6.1) pode ter suas raízes reais delimitadas usando o Teorema de Lagrange.

Teorema 3 (Lagrange): Dada a equação $P(x) = c_n x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_2x^2 + c_1x + c_0 = 0$, se $c_n > 0$ e k $(0 \leq k \leq n-1)$ for o maior índice de coeficiente escolhido dentre os coeficientes negativos, então o limite superior das raízes positivas de $P(x) = 0$ pode ser dado por

$$L = 1 + \sqrt[n-k]{\frac{B}{c_n}}$$

onde B é o valor absoluto do maior coeficiente negativo em módulo.

Desse modo, se ξ_p for a maior das raízes positivas de $P(x) = 0$, então $\xi_p \leq L$.

Exemplo: Seja $P(x) = x^4 + 2x^3 - 13x^2 - 14x + 24 = 0$. Os coeficientes negativos são $c_2 = -13$ e $c_1 = -14$, portanto, $k = 2$, pois $2 > 1$, $B = |-14|$ e
$\mathrm{L} = 1 + \sqrt[4-2]{\frac{14}{1}} \rightarrow L = 4.74$.

Para essa equação, o Teorema de Lagrange garante que $P(x) = 0$ não tem nenhuma raiz maior que 4.74.

Se $c_i > 0 (i = 0, 1, \ldots, n)$, então $P(x) = 0$ não tem raízes positivas, pois $P(x) = \sum_{i=0}^{n} c_i x > 0$ para $c_i > 0$ e $x > 0$.

Para determinar os limites superior e inferior das raízes positivas e negativas, são necessárias três equações auxiliares:

$\mathrm{P}_1(x) = x^n P(1/x) = 0$,

$\mathrm{P}_2(x) = P(-x) = 0$ e

$\mathrm{P}_3(x) = x^n P(-1/x) = 0$

Exemplo: Seja $P(x) = x^4 - 6x^3 - 5x^2 + 42x + 40 = 0$, com raízes $\xi_1 = -2$, $\xi_2 = -1$, $\xi_3 = 4$, $\xi_4 = 5$, então as equações auxiliares e suas respectivas raízes são:

$\mathrm{P}_1(x) = x^4 P(1/x) = 40x^4 + 42x^3 - 5x^2 - 6x + 1$,

$(\xi_1 = -0.5; \xi_2 = -1; \xi_3 = 0.25; \xi_4 = 0.2)$

$\mathrm{P}_2(x) = P(-x) = x^4 + 6x^3 - 5x^2 - 42x + 40$

$(\xi_1 = 2, \xi_2 = 1; \xi_3 = -4; \xi_4 = -5)$

$\mathrm{P}_3(x) = x^4 P(1/x) = 40x^4 - 42x^3 - 5x^2 + 6x + 1,$

$(\xi_1 = 0.5; \xi_2 = 1; \xi_3 = -0.25; \xi_4 = -0.2)$

Se $1/\xi_q$ for a maior das raízes positivas de $P_1(x)$, então ξ_q será a menor das raízes positivas de $P(x)$. Sendo L_1 o limite superior das raízes positivas de $P_1(x)$, tem-se que:

$$\frac{1}{\xi_q} \leq L_1 \rightarrow \xi_q \geq \frac{1}{L_1}$$

consequentemente, o limite inferior das raízes positivas de $P(x)$ é $1/L_1$. Desse modo, se $P(x)$ possuir raízes positivas ξ^+, elas estarão no intervalo

$$\frac{1}{L_1} \leq \xi^+ \leq L$$

Por outro lado, se $-\xi_r$ for a maior das raízes positivas de $P_2(x)$, então ξ_r será a menor das raízes negativas de $P(x)$. Sendo L_2 o limite superior das raízes positivas de $P_2(x)$, tem-se que:

$$-\xi_r \leq L_2 \rightarrow \xi_r \geq -L_2$$

Se $-1/\xi_s$ for a maior das raízes positivas de $P_3(x)$, então ξ_s será a menor das raízes negativas de $P(x)$. Sendo L_3 o limite superior das raízes positivas de $P_3(x)$

$$-\frac{1}{\xi_s} \leq L_3 \rightarrow \xi_s \leq -\frac{1}{L_3}$$

Então, se $P(x)$ tiver raízes negativas ξ^-, elas estarão no intervalo

$$-L_2 \leq \xi^- \leq -\frac{1}{L_3}$$

Exemplo: Calcular os limites das raízes reais de $P(x) = x^4 + 2x^3 - 13x^2 - 14x + 24 = 0$

As equações auxiliares são:

$$P_1(x) = x^4 P\left(\frac{1}{x}\right) = x^4\left(\frac{1}{x^4} + \frac{2}{x^3} - \frac{13}{x^2} - \frac{14}{x} + 24\right) \rightarrow P_1(x) = 24x^4 - 14x^3 - 13x^2 + 2x + 1 = 0,$$

$$\mathrm{L}_1 = 1 + \sqrt[4-3]{\frac{14}{24}} \rightarrow L_1 = 1.58334 \rightarrow \frac{1}{L_1} = 0.63$$

$$\mathrm{P}_2(x) = P(-x) = (-x)^4 + 2(-x)^3 - 13(-x)^2 - 14(-x) + 24 = 0 \rightarrow$$

$$\mathrm{P}_2(x) = x^4 - 2x^3 - 13x^2 + 14x + 24 = 0,$$

$$\mathrm{L}_2 = 1 + \sqrt[4-3]{\frac{13}{1}} \rightarrow L_2 = 14 \rightarrow -L_2 = -14$$

$$P_3(x) = x^4 P(-\tfrac{1}{x}) = x^4 \left(\tfrac{1}{(-x)^4} + \tfrac{2}{(-x)^3} - \tfrac{13}{(-x)^2} - \tfrac{14}{(-x)} + 24 \right) = 0 \rightarrow$$

$$P_3(x) = 24x^4 + 14x^3 - 13x^2 - 2x + 1 = 0$$

$$\mathrm{L}_3 = 1 + \sqrt[4-2]{\frac{13}{24}} \rightarrow L_3 = 1.736 \rightarrow -\frac{1}{L_3} = -0.58$$

Considerando que $L = 4.74$, então os limites das raízes reais são

$$0.63 \leq \xi^+ \leq 4.74 \quad \text{e} \quad -14 \leq \xi^- \leq -0.58$$

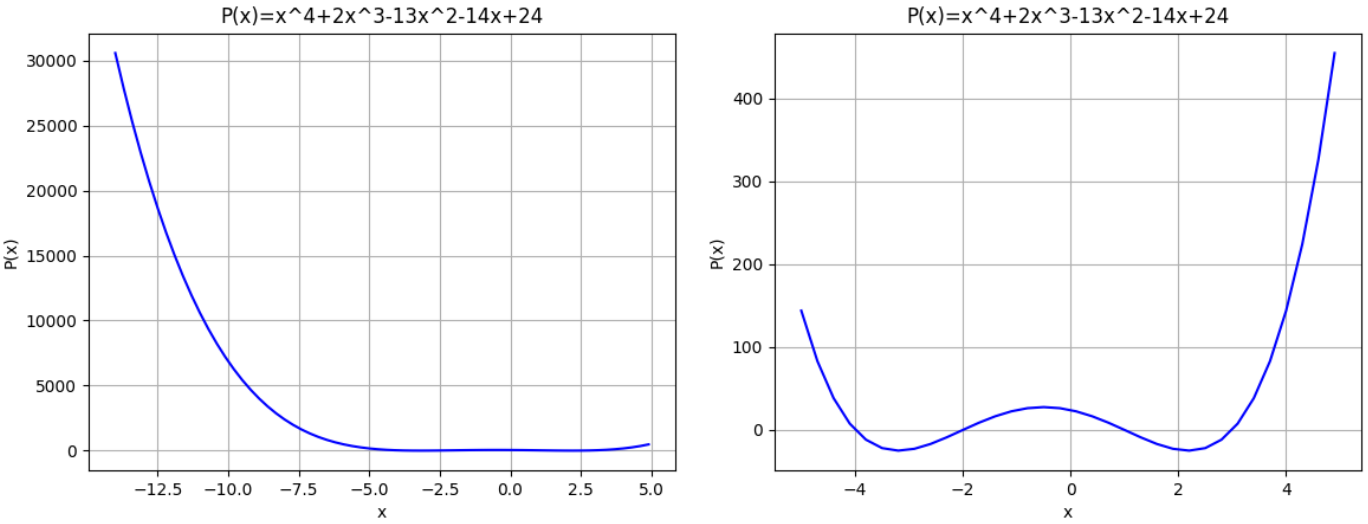
Podemos ver o comportamento da função nos gráficos da Figura 6.1. Na Figura 6.1a usamos o intervalo $[-14, 5]$, como calculamos acima. Mas, para visualizar melhor o comportamento e localização das raízes, reduzimos o intervalo para $[-5, 5]$, Figura 6.1b.

Dispositivo prático para cálculo dos limites de raízes reais

Usaremos esse dispositivo como base para construirmos um algoritmo de determinação de limites de raízes reais.

O dispositivo é constituído de dois blocos. No primeiro bloco, são definidos os coeficientes de $P(x) = 0$ e de suas equações auxiliares $P_1(x) = 0$, $P_2(x) = 0$ e $P_3(x) = 0$. Para isso:

1. colocar os coeficientes de $P(x) = 0$ na coluna $P(x)$, com c_n no topo,
2. inverter a ordem dos coeficientes da coluna $P(x)$ e colocá-los em $P_1(x)$,
3. trocar o sinal dos coeficientes de $P(x)$, cujos índices sejam ímpares e atribuí-los a $P_2(x)$,
4. inverter a ordem da coluna $P_2(x)$ e colocá-los em $P_3(x)$ e



(a) Intervalo [-14,5] (b) Intervalo[-5,5]

Figura 6.1: Limites das raízes reais de uma equação algébrica.

5. se algum $c_n < 0$, então trocar o sinal de todos os coeficientes da coluna para garantir que $c_n > 0$, conforme exige o Teorema de Lagrange.

No segundo bloco, são atribuídos os parâmetros necessários para aplicar o Teorema de Lagrange a cada uma das quatro equações. Assim:

- k é o índice do primeiro coeficiente negativo,
- n é o grau do polinômio,
- B é o valor absoluto do maior coeficiente negativo em módulo,
- L_i é o limite superior das raízes positivas de $P_i(x) = 0$ dado pelo Teorema de Lagrange,
- L_ξ são os limites superior e inferior das raízes positivas e negativas de $P(x) = 0$, sendo que $L_{\xi(P)} = L, L_{\xi(P_1)} = 1/L_1, L_{\xi(P_2)} = -L_2$ e $L_{\xi(P_3)} = -1/L_3$.

Exemplo: Calcular os limites das raízes reais de $P(x) = x^4 + 2x^3 - 13x^2 - 14x + 24 = 0$, usando o dispositivo prático.

$n = 4$	$P(x)$	$P_1(x)$	$P_2(x)$	$P_3(x)$
c_4	1	24	1	24
c_3	2	-14	-2	14
c_2	-13	-13	-13	-13
c_1	-14	2	14	-2
c_0	24	1	24	1
k	2	3	3	2
$n - k$	2	1	1	2
B	|-14|	|-14|	|-13|	|-13|
L_i	4.74	1.58	14	1.74
L_ξ	4.74	0.63	-14	-0.58

Algoritmo para determinar os limites de raízes reais

Como falado, esse algoritmo foi baseado no dispositivo prático que apresentamos acima. O Algoritmo 9 foi proposto por Campos (2007).

Esse algoritmo foi implementado em Python 3 e o código final é apresentado no Programa 6.3. Algumas adaptações foram necessárias para adequar à indexação das estruturas utilizadas, como arrays *NumPy*. No mesmo programa, aproveitamos para mostrar uma chamada da função *limites()* para o polinômio que usamos no exemplo anterior, cujo resultado é mostrado a seguir.

Programa 6.3: Determinação dos limites das raízes reais de um polinômio

```
1 def limites(n,cc):
2     """
3     Objetivo: achar os limites das raizes reais
4         de uma equacao polinomial
5     :param n: grau do polinomio
6     :param cc: lista de coeficientes, sendo
7         P(x) = c(1)x^n+ c(2)x^(n-1)+ ... + c(n)x+c(n-1)
8     :return: L (limites inferior e superior das
```

```
9           raizes positivas e negativas, respectivamente)
10      :return: msg (Mensagem)
11      """
12
13      msg = ""
14      c = np.zeros([n+1])
15      for i in range(len(cc)):
16          c[i] = cc[i]
17
18      if(cc[0] == 0):
19          return
20      t = n
21      while(True):
22          #se c(n) for nulo, entao o polinomio eh deflacionado
23          if(c[t]!=0):
24              break
25          t -= 1
26          c = cc.copy()
27      # calculo dos quatro limites das raizes reais
28      L = np.zeros(4)
29      # Verifica se todos os coeficientes s o positivos
30      all_coef_pos = all(c_i >= 0 for c_i in c)
31
32      # Se forem todos positivos, troca o sinal de todos
33      if all_coef_pos:
34          c = [-c_i for c_i in c]
35          msg = "O polinomio n o possui ra zes reais positivas"
36
37      for i in range(1,5):
```

```
38          # inversao da ordem dos coeficientes
39          if(i == 2 or i == 4):
40              for j in range (0, int(t/2)):
41                  aux = c[j]
42                  c[j] = c[t-j]
43                  c[t-j] = aux
44          else:
45              # reinversao da ordem e troca
46              # de sinais dos coeficientes
47              if(i == 3):
48                  for j in range(0, int(t/2)):
49                      aux = c[j]
50                      c[j] = c[t-j]
51                      c[t-j] = aux
52                  for j in range (t-1,0,-2):
53                      c[j] = -c[j]
54          # se c(0) for negativo, entao eh trocado
55          # o sinal de todos os coeficientes
56          if (c[0] < 0):
57              for j in range(t):
58                  c[j] = -c[j]
59          # calculo de k,
60          # o maior indice dos coeficientes negativos
61          k = 1
62          while(1):
63              if(c[k]<0 or k>t):
64                  break
65              k+=1
66          if(k==0):
```

```
67              k=1
68          if (k<=t):
69              B = 0
70              for j in range(1,t+1):
71                  if(c[j] < 0 and abs(c[j]) > B):
72                      B = abs(c[j])
73              # limite das raizes positivas de P(x) = 0
74              # e das equacoes auxiliares
75              if(((i-1)==0 or(i-1)==1) and all_coef_pos):
76                  L[i-1]=None
77              else:
78                  L[i-1] = 1 + pow((B/c[0]),(1/k))
79          else:
80              L[i-1] = 10**100
81      L[0] = L[0]
82      L[1] = 1/L[1]
83      L[2] = -L[2]
84      L[3] = -1/L[3]
85      print(L)
86      return L, msg
87
88  if(__name__ == "__main__"):
89      n = 4
90      c = [1,2,-13,-14,24]
91      L, msg = limites(n,c)
92      print("--- Limites das raizes reais ---")
93      print("--- Intervalo das raizes positivas: [{},{}]".format(
94          L[1],L[0]))
95      print("--- Intervalo das raizes negativas: [{},{}]".format(
```

```
96          L[2], L[3]))
97      print(msg)
```

A saída do Programa 6.3 é:

```
--- Limites das raízes reais ---
--- Intervalo das raízes positivas: [0.631578947368421,4.741657386773941]
--- Intervalo das raízes negativas: [-14.0,-0.5760434788494824]
```

Mesmo resultado que encontramos quando realizamos os cálculos manualmente.

Número de raízes reais

A Regra de sinais de Descartes é uma maneira simples para determinar o número de raízes reais de uma equação algébrica.

Sabendo-se os limites e o número das raízes reais, a tarefa de isolar essas raízes ficará mais fácil.

Teorema 4 (Regra de sinais de Descartes): O número de raízes reais positivas n^+ de $P(x) = 0$ é igual ao número de variações de sinais na sequência dos coeficientes ou é menor que esse número por um inteiro par, sendo as raízes contadas de acordo com a sua multiplicidade e não sendo considerados os coeficientes nulos.

Corolário 2: Se $P(x) = 0$ não possuir coeficientes nulos, então o número de raízes reais negativas n^- (contando multiplicidades) é igual ao número de permanências de sinais na sequência dos coeficientes ou é menor que esse número por um inteiro par.

Importante: A Regra de sinais de Descartes consegue discernir entre as raízes positivas e negativas; no entanto, não consegue separar as raízes reais das complexas, as quais aparecem em pares conjugados. Por exemplo, se o número de variações de sinais for 5, então $n^+ = 5$ ou 3 ou 1.

Exemplo: Para $P(x) = x^4 + 2x^3 - 13x^2 - 14x + 24 = 0$, tem-se que,

$$n^+ = 2 \text{ ou } 0, \qquad \text{e} \qquad n^- = 2 \text{ ou } 0.$$

Para essa equação, se existirem duas raízes positivas, elas estarão no intervalo $0.63 \leq \xi^{+} \leq 4.74$ e, se houver duas negativas, elas estarão no intervalo $-14 \leq \xi^{-} \leq -0.58$.

Exemplo: Calcular os limites e o número de raízes reais de $P(x) = x^3 - 3x^2 - 6x + 8 = 0$.

Solução: Para aplicar o Teorema de Lagrange e obter os limites, o coeficiente $c_n = c_3$ deve ser positivo em cada um dos quatro polinômios. Como isso não é verdadeiro para $P_2(x)$, então todos os coeficientes desse polinômio devem ter os seus sinais trocados. Os zeros do novo polinômio serão iguais aos do polinômio original e, por conseguinte, os dois polinômios terão o mesmo L_2.

$\mathrm{P(x)} = \mathrm{x}^3 - 3x^2 - 6x + 8 = 0$

$B = |-6| = 6$

$k = 2$

$L = 1 + \sqrt[3-2]{\frac{6}{1}}$

$\boxed{\mathrm{L} = 7}$

$\mathrm{P}_1(x) = x^n P(1/x) = x^3 \left(\frac{1}{x^3} - \frac{3}{x^2} - \frac{6}{x} + 8\right)$

$P_1(x) = 8x^3 - 6x^2 - 3x + 1$

$B_1 = |-6|$

$k_1 = 2$

$L_1 = 1 + \sqrt[3-2]{\frac{6}{8}}$

$\boxed{\mathrm{L}_1 = 1.75}$

$\mathrm{P}_2(x) = P(-x)$

$P - 2(x) = (-x)^3 - 3(-x)^2 - 6(-x) + 8 = 0$

$P_2(x) = -x^3 - 3x^2 + 6x + 8$

$c_n < 0 \rightarrow$ troca de sinais dos coeficientes

$P_2(x) = x^3 + 3x^2 - 6x - 8$

$B_2 = |-8| = 8$

$k_2 = 2$

$L_2 = 1 + \sqrt[3-1]{\frac{8}{1}}$

$\boxed{\mathrm{L}_2 = 3.82}$

$\mathrm{P}_3(x) = x^n P(-1/x)$

$P_3(x) = x^3 \left(-\frac{1}{x^3} - \frac{3}{x^2} - \frac{6}{x} + 8\right)$

$P_1(x) = 8x^3 + 6x^2 - 3x - 1$

$B_3 = |-3|$

$k_3 = 1$

$L_3 = 1 + \sqrt[3-1]{\frac{3}{8}}$

$\boxed{\mathrm{L}_3 = 1.61}$

$\frac{1}{L_1} \leq \xi^+ \leq L \rightarrow 0.57 \leq \xi^+ \leq 7$

$-L_2 \leq \xi^- \leq \frac{1}{L_3} \rightarrow -3.82 \leq \xi^- \leq 0.62$

Quanto ao número de raízes, temos: $n^+ = 2$ ou 0 e $n^- = 1$. Ou seja, neste caso, uma raiz real negativa e as outras duas serão reais positivas ou complexas.

Exemplo: Achar os limites e o número de raízes reais de $P(x) = x^6 - 5x^5 + 7x^4 + 19x^3 - 98x^2 - 104x = 0$

Solução: Como $c_0 = 0$, tem-se, pelas relações de Girard, que existe pelo menos uma raiz nula. Para obter as equações auxiliares, $P(x)$ deve ser deflacionado pela divisão por $(x - 0)$, resultando em $P(x) = x^5 - 5x^4 + 7x^3 + 19x^2 - 98x - 104 = 0$. Esse novo $P(x)$ terá todas as raízes do antigo, com exceção da raiz nula. Neste exemplo, como $c_5 < 0$ para $P_i(x), i = 1, 2, 3$, os sinais de todos os coeficientes de $P_1(x), P_2(x)$ e $P(3(x)$ devem ser trocados para que se possa aplicar o Teorema.

$n = 5$	$P(x)$	$P_1(x)$	$P_2(x)$	$P_3(x)$
c_5	1	104	1	104
c_4	-5	98	5	-98
c_3	7	-19	7	-19
c_2	19	-7	-19	7
c_1	-104	-1	104	1
k	4	3	2	4
$n - k$	1	2	3	1
B	|-104|	|-19|	|-98|	|-98|
L_i	105	1.43	5.61	1.94
L_ξ	105	0.70	-5.61	-0.51

Portanto, os limites das raízes são $0.70 \leq \xi^+ \leq 105$ e $-5.61 \leq \xi^- \leq -0.51$ Quanto ao número de raízes reais, temos: $n^+ = 3$ ou 1 e $n^- = 2$ ou 0, garantindo pelo menos uma raiz real positiva.

6.2.2 Equações transcendentes

As equações transcendentes não dispõem de teoremas, como visto para as equações algébricas, que forneçam informações sobre os limites e o número de raízes reais.

Uma equação transcendente pode ter um número infinito de raízes, como por exemplo, $f(x) = sen(x)$, ou mesmo não ter raízes como $f(x) = sen(x) - 2$.

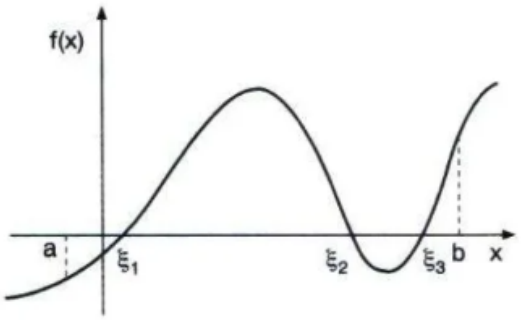
Neste caso, o método gráfico é a maneira mais simples para achar um intervalo que contenha uma única raíz. O método consiste em fazer um esboço da função no intervalo de interesse. A dificuldade com as equações transcendentes consiste em determinar esse intervalo, já que não existe um teorema pra isso. Na prática, usamos a intuição, o conhecimento a respeito da função e o método da tentativa e erro, ou seja, a partir de um intervalo inicial, tenta-se um outro que isole a raiz observando-se a forma da curva.

Teorema de Bolzano

Teorema de Bolzano:

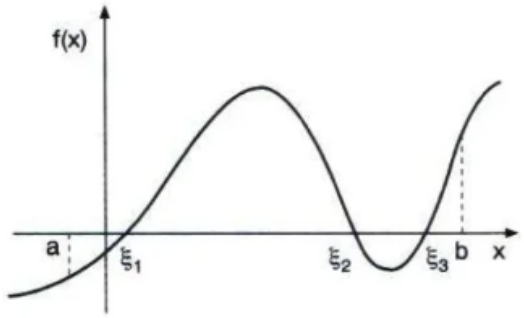
Seja $f(x)$ uma função contínua em um intervalo $[a, b]$. Se $f(a) \cdot f(b) < 0$, então existe pelo menos um ponto c, onde $c \in [a, b]$ tal que $f(c) = 0$.

$f(a) \cdot f(b) < 0$, ou seja, sinais opostos



Corolário:

Se f' preservar o sinal em $[a, b]$, então existe um único $c \in [a, b]$ tal que $f(c) = 0$.

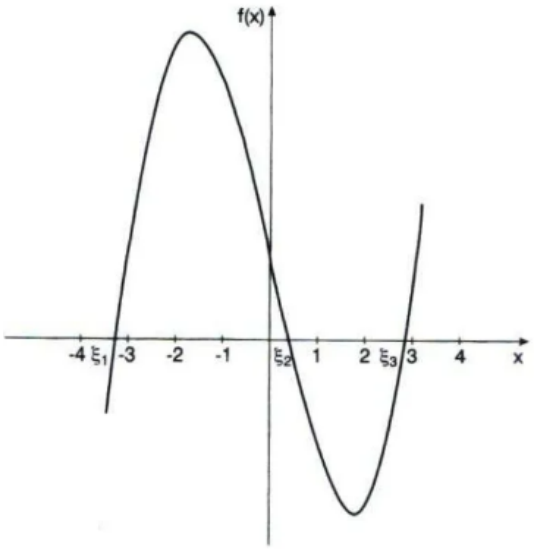


Tabelamento e Gráfico

Exemplo:

$$f(x) = x^3 - 9x + 5,\ [0, 3]$$

x	$f(x)$
0	5
0.5	0.625
1	-3
1.5	-5.125
2	-5
2.5	-1.875
3	5



Pela tabela e gráfico podemos ver que existe raiz nos intervalos $[0.5; 1.0]$ e $[2.5; 3.0]$.

Algoritmo da troca de sinais

O Algoritmo 11, proposto por Campos (2007), fornece um intervalo $[a, b]$, no qual uma função troca de sinal, ou seja, $f(a)f(b) < 0$. Esse intervalo é expandido a partir de um dado valor z. Apesar de a raiz não estar necessariamente isolada, o algoritmo fornecerá um intervalo de partida para o isolamento de uma raiz.

Esse algoritmo foi codificado em linguagem Python, como podemos ver no Programa 6.4.

Programa 6.4: Determinação de intervalo onde uma função troca de sinal

```
1  def troca_sinal(z, imprime_tabela=False):
2      """
3      Objetivo: Achar um intervalo [a,b] onde uma funcao troca
4          de sinal
5      parametros de entrada: z (ponto a partir do qual o
6          intervalo serah gerado)
7      parametros de saida: a (limite inferior),
8          b (limite superior) e
9          CondErro (condicao de erro), sendo
10             CondErro = 0 se f(a)f(b) < 0 e
11             CondErro = 1 se f(a)f(b) > 0
12     """
13     if(z == 0):
14         a = -0.05
15         b = 0.05
16     else:
17         a = 0.95 * z
18         b = 1.05 * z
19     iter = 0
20     aureo = 2/(pow(5,0.5)-1)
21     Fa = ff(a)
22     Fb = ff(b)
23     tab = np.zeros([20,5])
24     linha = [iter,a,b,Fa,Fb]
25     tab[iter] = linha
26     while (1==1):
27         if (Fa * Fb < 0 or iter >= 20):
28             break
```

```
29          iter+=1
30          if (abs(Fa)< abs(Fb)):
31              a = a - aureo * (b-a)
32              Fa = ff(a)
33          else:
34              b = b + aureo * (b-a)
35              Fb = ff(b)
36          linha = [iter,a,b,Fa,Fb]
37          tab[iter] = linha
38      if(Fa*Fb <=0):
39          CondErro = 0
40          if(imprime_tabela):
41              columns = ["Iter","a","b","Fa","Fb"]
42              tab_table = pd.DataFrame(tab[0:iter + 1, 0:5],
43                  columns=columns)
44              print(tab_table)
45      else:
46          CondErro = 1
47      return [a,b,CondErro]
```

Exemplo: Achar um intervalo, a partir de $z = 5$, onde $f(x) = 2x^3 - cos(x + 1) - 3$ troca de sinal, usando o Programa 6.4.

Para resolver esse problema utilizaremos o código apresentado no Programa 6.5, considerando que a função troca_sinal(), apresentada no Programa 6.4 está declarada no mesmo arquivo.

Programa 6.5: Exemplo de uso do Programa 6.4

```
1 def ff(x):
2     fx = 2 * (x ** 3) - math.cos(x + 1) - 3
3     return fx
```

```
4
5 if(__name__=="__main__"):
6     z = 5
7     print("Deter. de intervalo onde ocorre troca de sinal")
8     ab= troca_sinal(z,True)
9
10    print("a =",ab[0])
11    print("b =",ab[1])
12    print("condErro =",ab[2])
```

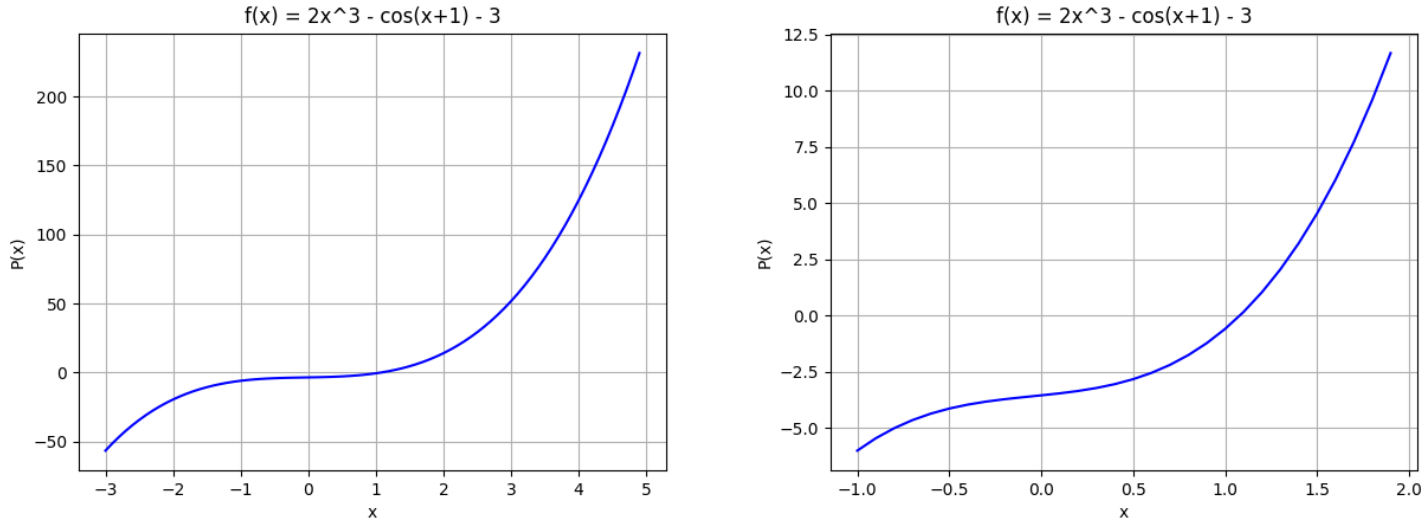
A saída do Programa 6.5 será:

```
Determinação de intervalo onde ocorre troca de sinal
Iter         a     b          Fa          Fb
0   0.0  4.750000  5.25  210.482558  285.406801
1   1.0  3.940983  5.25  119.190941  285.406801
2   2.0  1.822949  5.25   10.065502  285.406801
3   3.0 -3.722136  5.25 -105.221837  285.406801
a = -3.7221359549995783
b = 5.25
condErro = 0
```

Vamos esboçar um gráfico para visualizarmos melhor o comportamento dessa função. Na Figura 6.2a, traçamos o gráfico no intervalo $[-3, 5]$ e verificamos que existe uma única raiz entre $[-1, 2]$. A curva pode ser esboçada nesse intervalo menor, Figura 6.2b, para visualizarmos o gráfico com uma melhor definição.

6.3 Métodos para o Cálculo de Raízes de Funções

A Figura 6.3 mostra um algoritmo genérico para o cálculo de zeros de função por métodos numéricos.



(a) Intervalo [-3,5] (b) Intervalo[-1,2]

Figura 6.2: Esboços da função $f(x) = 2x^3 - cos(x+1) - 3$ em dois intervalos.
Fonte: A autora.

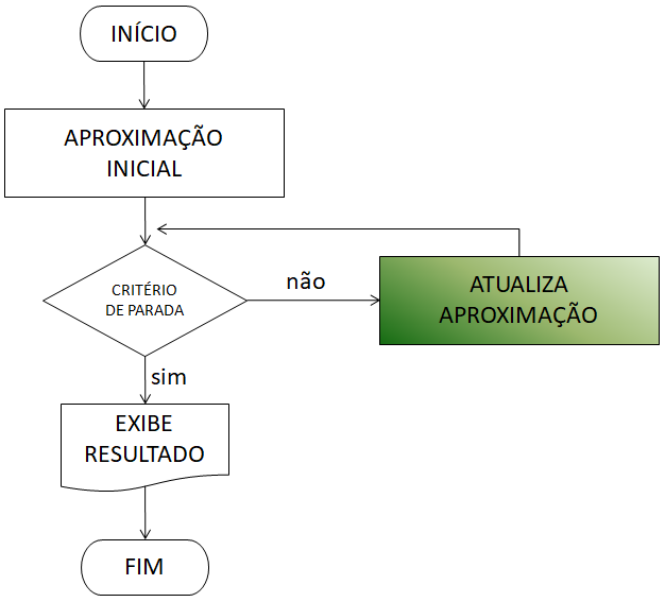


Figura 6.3: Algoritmo geral para o cálculo de zeros de funções.

Critérios de parada

i) $f(x_k) < \varepsilon$, onde ε é a precisão;

ii) $|x_k - x_{k-1}| < \varepsilon$

iii) número limite de iterações (alguns métodos não têm garantia de convergência).

Neste e-book, trataremos dos seguintes métodos numéricos para o cálculo das raízes de funções:

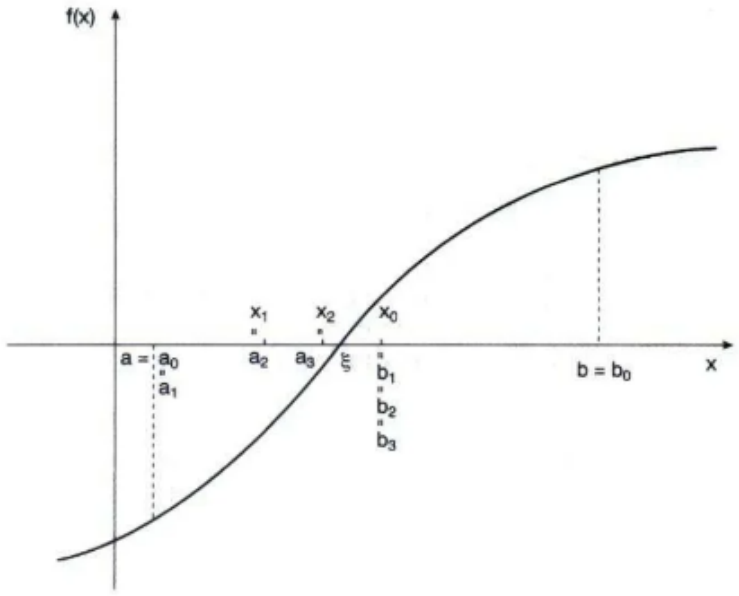
- Método da Bisseção;

- Método da Falsa Posição;
- Método do Ponto Fixo;
- Método de Newton-Raphson;
- Método da Secante.

6.3.1 Método da Bisseção

Também conhecido como método do ponto médio.

Dada uma função $f(x)$, o método da bisseção atualiza o cálculo da raiz aproximada por meio do ponto médio de um intervalo. Graficamente, temos,



$$\bar{x}_k = \frac{a+b}{2}$$

$$f(\bar{x}_k) \approx 0$$

$$f(\bar{x}_k) < \varepsilon$$

Algoritmo do Método da Bisseção

O Algoritmo 12 mostra em pseudocódigo do Método da Bisseção para cálculo de raízes de funções. A implementação desse método em Python é mostrada no Programa 6.6.

Programa 6.6: Método da Bisseção

```
1 def metodo_bissecao(f, va, vb, Toler, IterMax):
2     """
3     :param f: funcao
4     :param va: valor inicial do intervalo
5     :param vb: valor final do intervalo
6     :param Toler: tolerancia
```

```
7       :param IterMax: numero maximo de iteracoes
8       :return: raiz, iter(numero de iteracoes gastas),
9           rebis_table(tabela de calculo), msg_erro, CondErro,
10          sendo
11          CondErro = 0 se a raiz foi encontrada e
12          CondErro = 1 em caso contrario
13      """
14      msg_erro = ""
15      a = va
16      b = vb
17      f_a = f(a)
18      f_b = f(b)
19      cond_erro = 0
20      if (f_a * f_b > 0):
21          msg_erro = "Funcao nao muda de sinal
22              nos extremos do intervalor dado"
23          cond_erro = 1
24
25      if(not(cond_erro)):
26          delta_x = abs(b - a) / 2
27          iter = 1
28          k = math.ceil(math.log(((b - a) / Toler), 2) - 1)
29          rebis = []
30
31          while (iter <= IterMax):
32              x = (a + b) / 2
33              f_x = f(x)
34              rebis.append([iter,a, f_a, b, f_b, x, f_x, delta_x])
35              if ((delta_x <= Toler and abs(f_x) <= Toler)
```

```
36                  or iter >= IterMax):
37                      break
38              if ((f_a * f_x) > 0):
39                  a = x
40                  f_a = f_x
41              else:
42                  b = x
43                  f_b = f_x
44              delta_x = delta_x / 2
45              iter += 1
46          rebis_table = pd.DataFrame(rebis,
47              columns=["iter", "a", "f(a)", "b", "f(b)",
48                  "x", "f(x)", "delta_x"])
49
50          # teste de converg ncia
51          if (delta_x <= Toler and abs(f_x) <= Toler):
52              cond_erro = 0
53          else:
54              cond_erro = 1
55      else:
56          iter = 0
57          x = 0
58          rebis_table = []
59      return x, iter, rebis_table, msg_erro, cond_erro
```

Exemplo: Usando o método da bisseção, calcule as raízes de $f(x) = x^3 - 9x + 5$, no intervalo $[0, 3]$.

Solução:

Isolamento das raízes:

Vimos na seção 6.2 que essa função tem raízes nos intervalos $[0.5; 1.0]$ e $[2.5; 3.0]$. Neste

exemplo, vamos refinar a raiz do intervalo $[0.5; 1.0]$. A raiz de $[2.5; 3.0]$ ficará como exercício para treinar o uso do método.

Então, temos as informações iniciais:

$f(x) = x^3 - 9x + 5$, $I = [0.5; 1.0]$

$a = 0.5$

$b = 1$

$\varepsilon = 10^{-2}$

A atualização da raiz é dada pela fórmula:

$$\bar{x} = \frac{a+b}{2}$$

a	b	$\bar{x}$	$f(\bar{x})$
0.5	1	0.75	-1.3281
0.5	0.75	0.625	-0.3808
0.5	0.625	05625	0.1154
0.5625	0.625	0.5937	-0.1344
0.5625	0.5937	0.5781	-0.009

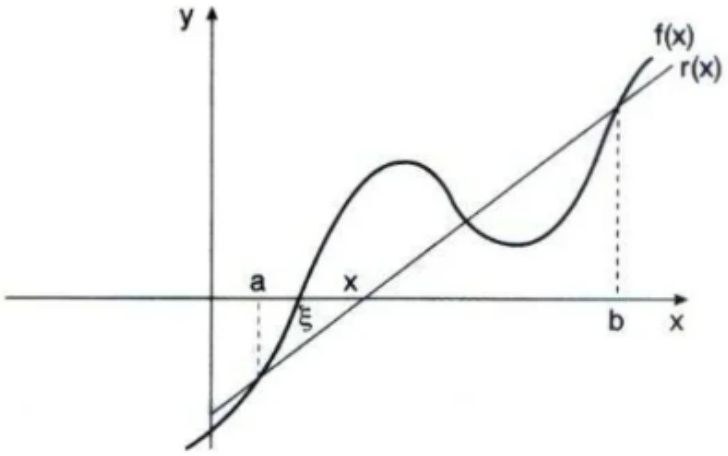
Como $f(0.5781) = -0.009 < \varepsilon$, nossa raiz aproximada é 0.5781.

6.3.2 Método da Falsa Posição

O método da Falsa Posição difere do método da bisseção pela forma como a raiz é atualizada. Neste caso:

$$\bar{x}_k = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

Graficamente, temos,



Exemplo: Usando o método da falsa posição, calcule as raízes de $f(x) = x^3 - 9x + 5$, no intervalo $[0, 3]$.

Solução:

$f(x) = x^3 - 9x + 5$, $I = [0.5; 1]$

$a = 0.5$

$b = 1$

$\varepsilon = 10^{-2}$

$$\bar{x} = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

a	$f(a)$	b	$f(b)$	$\bar{x}$	$f(\bar{x})$
0.5	0.625	1	-3	0.5862	-0.0744
0.5	0.625	0.5862	-0.0744	0.5770	-0.0011

Como $f(0.5770) < \varepsilon$, nossa raiz aproximada é 0.5770.

Implementação do Método da Falsa Posição

Um algoritmo para o método da falsa posição, proposto por Campos(2007), pode ser visto no Algoritmo 13. Esse algoritmo foi implementado na linguagem Python e o resultado é o Programa 6.7.

Programa 6.7: Método da Falsa Posição

```
1  def metodo_falsa_posicao(f,va, vb, Toler, IterMax):
2      """
3      :param f: funcao
4      :param va: valor inicial do intervalo
5      :param vb: valor final do intervalo
6      :param Toler: tolerancia
7      :param IterMax: numero maximo de iteracoes
8      :return: raiz, iter(numero de iteracoes gastas),
9          rebis_table(tabela de calculo), msg_erro, CondErro,
10         sendo
11         CondErro = 0 se a raiz foi encontrada e
12         CondErro = 1 em caso contrario
13     """
14     a = va
15     b = vb
16     f_a = f(a)
17     f_b = f(b)
18     cond_erro = 0
19     msg_erro = ""
20     if (f_a * f_b > 0):
21         msg_erro ="Funcao nao muda de sinal
22             nos extremos do intervalor dado"
23         cond_erro = 1
24     if(not(cond_erro)):
25         delta_x = abs(a*f_b-b*f_a)/(f_b-f_a)
26         iter = 1
27         k = math.ceil(math.log(((b - a) / Toler), 2) - 1)
28         rebis = []
```

```
29
30          while (iter <= IterMax):
31              x = (a*f_b - b*f_a) / (f_b-f_a)
32              f_x = f(x)
33
34              rebis.append([iter,a, f_a, b, f_b, x, f_x, delta_x])
35              if ((delta_x <= Toler and abs(f_x) <= Toler)
36                  or iter >= IterMax):
37                  break
38              if ((f_a * f_x) > 0):
39                  a = x
40                  f_a = f_x
41              else:
42                  b = x
43                  f_b = f_x
44              delta_x = delta_x / (f_b-f_a)
45              iter += 1
46          rebis_table = pd.DataFrame(rebis,
47              columns=["iter", "a", "f(a)", "b", "f(b)",
48                  "x", "f(x)", "delta_x"])
49
50          # teste de convergencia
51          if (delta_x <= Toler and abs(f_x) <= Toler):
52              cond_erro = 0
53          else:
54              cond_erro = 1
55      else:
56          iter = 0
57          x = 0
```

```
58          rebis_table = []
59      return x, iter, rebis_table, msg_erro, cond_erro
```

6.3.3 Método do Ponto Fixo

A importância desse método está mais nos conceitos que são introduzidos em seu estudo que em sua eficiência computacional.

Seja $f(x)$ uma função contínua em $[a, b]$, intervalo que contém uma raiz da equação $f(x) = 0$.

O MPF consiste em transformar essa equação em uma equação equivalente $x = \varphi(x)$ e a partir de uma aproximação inicial x_0, gerar uma sequência x_k de aproximações para ξ pela relação $x_{k+1} = \varphi(x_k)$, pois a função $\varphi(x)$ é tal que $f(\xi) = 0$ se, e somente se, $\varphi(\xi) = \xi$. Transformamos assim o problema de encontrar um zero de $f(x)$ no problema de encontrar um ponto fixo de $\varphi(x)$.

Uma função $\varphi(x)$ que satisfaz a condição acima é chamada de *função de iteração* para a equação $f(x) = 0$.

$$f(x) = 0 \Leftrightarrow x = \varphi(x)$$

$\varphi(x)$ gera uma sequência de soluções aproximadas.

No caso do método do ponto fixo, a atualização da aproximação é dada pela função $\varphi(x)$.

$$x_{k+1} = \varphi(x_k)$$

Forma geral de $\varphi(x)$:

$$\varphi(x) = x + A(x)f(x) \ , \ A(x) \neq 0 \tag{6.2}$$

Teorema: $\varphi(x) = x$ se e somente se $f(x) = 0$

Prova:

$(\Rightarrow)$ $\varphi(x) = x$

$\varphi(x) = x + A(x)f(x) \therefore \varphi(x) = \varphi(x) + A(x)f(x)$

$A(x) \cdot f(x) = 0 \quad A(x) \neq 0 \Rightarrow f(x) = 0$

$(\Leftarrow)$ $f(x) = 0$

$\varphi(x) = x + A(x) \cdot f(x)$

$\varphi(x) = x$

Graficamente, temos,

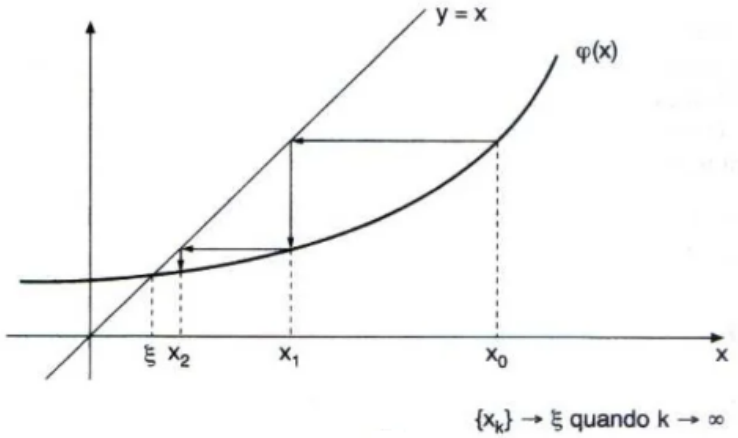


Figura 6.4: Esquema de iteração do Método do Ponto Fixo.

Mas como escolher $\varphi(x)$?

Exemplo: Seja $f(x) = x^2 + x - 6$. Para a equação $f(x) = 0$ temos várias funções de iteração.

Para $x^2 + x - 6 = 0$

podemos ter:

a) $\varphi_1(x) = 6 - x^2$
b) $\varphi_2(x) = \pm\sqrt{6 - x}$
c) $\varphi_3(x) = \frac{6}{x} - 1$
d) $\varphi_4(x) = \frac{6}{x+1}$

A forma geral das funções de iteração $\varphi(x)$ é $\varphi(x) = x + A(x)f(x)$, com a condição que em ξ, ponto fixo de $\varphi(x)$, se tenha $A(\xi) \neq 0$.

Condições de convergência do método do ponto fixo

i) $\varphi(x)$ e $\varphi'(x)$ são contínuas no intervalo de busca I

ii) $\left|\varphi'(x)\right| \leqslant M < 1 \quad \forall\, x \in I$. Então, quanto mais próximo M for de zero, mais rápido a função converge.

iii) $x_0 \in I$, onde o chute inicial (X_0) não é dado pelo MPF, mas sim por um valor do intervalo I.

Exemplo: Usando o método do ponto fixo, calcule as raízes de $f(x)$, no intervalo dado.

$$f(x) = x^3 - 9x + 5, \text{ no intervalo, } [0, 3]$$

Solução:

$f(x) = x^3 - 9x + 5$

$I = [0.5; 1]$

$\varepsilon = 10^{-2}$

$$x^3 - 9x + 5 = 0$$
$$-9x = -x^3 - 5 \quad \times(-1)$$
$$x = \tfrac{x^3+5}{9} \Rightarrow \varphi(x)$$

ITER	$\bar{x}$ $\varphi(x) = (x^3+5)/9$	$f(\bar{x})$ $f(x) = x^3 - 9x + 5$
0	$x_0 = 0.75$	-1.3281
1	$(0.75^3 + 5)/9 = 0.6024$	-0.2032
2	$(0.6024^3 + 5)/9 = 0.5798$	-0.0236
3	$(0.5798^3 + 5)/9 = 0.5772$	-0.00259

Como $f(0.5772) < \varepsilon$, nossa raiz aproximada é 0.5772, calculada pelo método do ponto fixo.

O Programa 6.8 implementa o método do Ponto Fixo em Python.

Programa 6.8: Método do Ponto Fixo

```
1 def ponto_fixo(f,g,va, vb, Toler, IterMax):
2     """
3     Metodo do ponto fixo para encontrar raizes de equacoes.
4     :param f: funcao f(x)
```

```
5       :param g: funcao g(x) tal que x = g(x)
6       :param va: valor inicial
7       :param vb: valor final
8       :param Toler: tolerancia de convergencia
9       :param IterMax: numero maximo de iteracoes
10      :return: raiz aproximada, numero de iteracoes,
11          codigo de erro (0 = convergiu, 1 = nao convergiu)
12      """
13      x0 = (va + vb)/2 # comeca pelo ponto medio do intervalo
14      x = x0
15      cond_erro = 0
16      tab = []
17      iter = 1
18      msg_erro = ""
19      while(iter <= IterMax):
20          try:
21              fx = f(x)
22              x1 = g(x)
23          except Exception as e:
24              msg_erro = "O metodo nao converge com as " \
25              "informacoes fornecidas.{}".format(e)
26              cond_erro = 1
27              break
28          distancia = abs(x1 - x)
29          tab.append([iter, x, fx, distancia])
30          if (abs(distancia) <= Toler or abs(fx) <= Toler):
31              raiz = x1
32              msg_erro = ''
33              cond_erro = 0
```

```
34              break
35          else:
36              cond_erro = 1
37          x = x1
38          iter = iter + 1
39      tabela = pd.DataFrame(tab,
40          columns=["iter", "x", "f(x)", "erro"])
41      if(cond_erro):
42          raiz = 0
43          if(msg_erro == ""):
44              msg_erro = "Erro. O metodo nao convergiu."
45          tabela = []
46
47      return raiz, iter, tabela, msg_erro, cond_erro
```

6.3.4 Método de Newton-Raphson

O método de Newton-Raphson é uma melhoria no método do ponto fixo. Partimos da hipótese que $\varphi'(x) = 0$.

Hipótese: $\varphi'(x) = 0$

O que muda?

$$\varphi(x) = x + A(x) \cdot f(x)$$

$$\varphi'(x) = 1 + A'(x) \cdot f(x) + A(x) \cdot f'(x)$$

Como, por hipótese, $\varphi'(x) = 0$

$$\varphi'(x) = 1 + A(x) \cdot f'(x) = 0$$

Então: $A(x) = -\frac{1}{f'(x)}$

$$\varphi(x) = x + \left(-\frac{1}{f'(x)}\right) \cdot f(x)$$

$$\boxed{\varphi(x) = x - \frac{f(x)}{f'(x)}} \rightarrow \text{Função de iteração do método de Newton-Raphson}$$

Uma outra forma de se obter essa função de iteração do método é por meio da análise da equação da reta tangente.

$$
\begin{aligned}
0 &= y - f(x_k) = f'(x_k)(x - x_k) \\
y &= f(x_k) + f'(x_k)(x - x_k) = 0 \\
f'(x_k)(x - x_k) &= -f(x_k)
\end{aligned}
$$

$$x - x_k = -\frac{f(x_k)}{f'(x_k)}$$

$$x = x_k - \frac{f(x_k)}{f'(x_k)}$$

Graficamente:

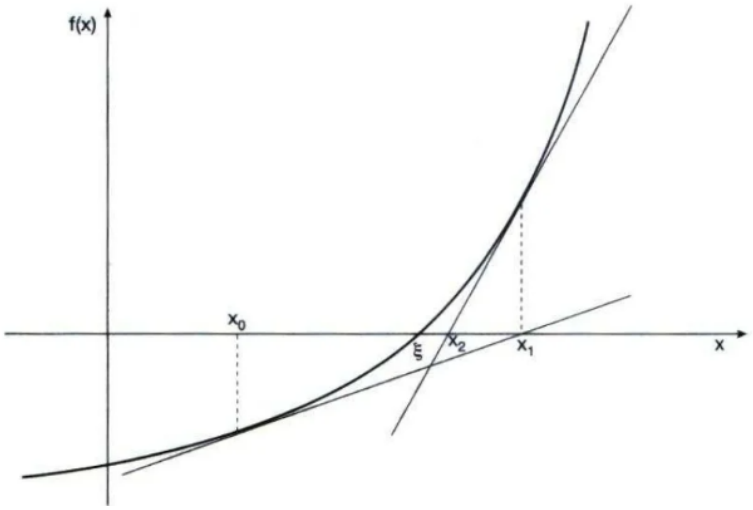


Figura 6.5: Esquema de iteração do Método de Newton-Raphson.

Condições de convergência:

i) $f(x)$, $f'(x)$ e $f''(x)$ são contínuas em I;

ii) $f'(x) \neq 0$;

iii) x_0 está suficientemente próximo da raiz.

Exemplo: Usando o método de Newton-Raphson, calcule as raízes de $f(x) = x^3 - 9x + 5$, no intervalo $[0, 3]$.

Solução:

$f(x) = x^3 - 9x + 5,\ I = [0.5; 1]$

$a = 0.5$

$b = 1$

$\varepsilon = 10^{-2}$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

ITER	$\bar{x}$	$f(\bar{x})$
0	$x_0 = 0.75$	-1.3281
1	$x_1 = 0.56837$	0.06823
2	$x_2 = 0.57687$	0.000123

Como $f(0.57687) < \varepsilon$, então a raiz aproximada pelo método de Newton-Raphson é 0.57687.

Algoritmo do Método de Newton-Raphson

Um algoritmo para o método de Newton-Raphson, proposto por Campos(2007), pode ser visto no Algoritmo 14. Esse algoritmo foi implementado na linguagem Python e o resultado é o Programa 6.9.

Programa 6.9: Método de Newton-Raphson

```
1  def newton_raphson(f,x0, Toler, IterMax):
2      """
3      :param f: funcao
4      :param x0: valor inicial
5      :param Toler: tolerancia
6      :param IterMax: numero maximo de iteraoees
7      :return: raiz, numero de iteracoes gastas e CondErro,
8          sendo
9          CondErro = 0 se a raiz foi encontrada e
10         CondErro = 1 em caso contrario
11     """
12     Fx = f(x0) # avaliar a funcao em x0
13     # avaliar a derivada em x0
```

```
14      DFx = misc.derivative(f,x0,dx=1e-4)
15      x = x0
16      Iter = 1
17      rebis = []
18      rebis.append([Iter, x, DFx, Fx, 0.0])
19      while(1):
20          DeltaX = -Fx/DFx
21          x = x + DeltaX
22          Fx = f(x) # avalia a funcao em x
23          # avalia a derivada da funcao em x
24          DFx = misc.derivative(f, x,dx=1e-4)
25          Iter += 1
26          rebis.append([Iter, x, DFx, Fx, DeltaX])
27          if((abs(DeltaX) <= Toler and abs(Fx) <= Toler)
28              or (DFx == 0) or(Iter >= IterMax)):
29              break
30      raiz = x
31      rebis_table = pd.DataFrame(rebis,
32          columns=["iter", "x", "f'(x)",
33              "f(x)", "delta_x"])
34      # Teste de convergencia
35      if((abs(DeltaX) <= Toler) and (abs(Fx)<=Toler)):
36          CondErro = 0
37          msg_erro = ""
38      else:
39          CondErro = 1
40          msg_erro = "Erro. O metodo nao convergiu."
41          rebis_table = []
42      return raiz, Iter, rebis_table, msg_erro, CondErro
```

6.3.5 Método da Secante

É uma modificação do método de Newton-Raphson. A vantagem é que eliminamos a necessidade de calcular a derivada da função. A desvantagem desse método é que, geralmente, possui uma convergência mais lenta.

A atualização da raiz no método de Newton-Raphson é dada pela fórmula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

No método da secante, procura-se substituir a $f^{'(x)}$ do método de Newton pela reta secante.

Graficamente:

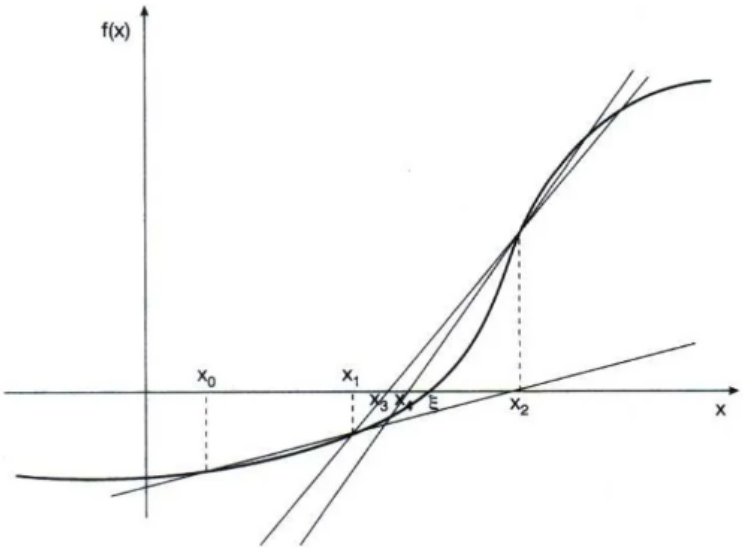


Figura 6.6: Esquema de iteração do Método de Secante.

Reta secante:

$$f^{'}(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Substituindo essa fórmula na equação de Newton-Raphson, temos:

$x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1}}{x_k - x_{k-1}}}$

$$\boxed{x_{k+1} = x_k - f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}} \rightarrow \text{Equação do método da secante}$$

Exemplo: Usando o método da secante, calcule as raízes de $f(x) = x^3 - 9x + 5$, no intervalo $[0, 3]$.

Solução:

$f(x) = x^3 - 9x + 5$, $I = [0.5; 1]$

$a = 0.5$

$b = 1$

$\varepsilon = 10^{-2}$

$$x_{k+1} = x_k - f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

ITER	$\bar{x}$	$f(\bar{x})$
0	$x_0 = 0.5$	0.625
1	$x_1 = 1$	-3
2	$x_2 = 0.5862$	-0.0744
3	$x_3 = 0.57567$	0.00969

Podemos observar que no método da secante, para se obter o x_{k+1} precisamos dos 2 últimos valores anteriores (x_k e x_{k-1}). Portanto, no tabelamento acima, as iterações 0 e 1 representam esses 2 valores iniciais, os quais usaram os extremos do intervalo pesquisado, $x_0 = 0.5$ e $x_1 = 1.0$

Como $f(0.57567) < \varepsilon$, então a raiz aproximada pelo método da secante é 0.57567.

$$x_2 = x_1 - \frac{f(x_1) \cdot (x_1 - x_0)}{f(x_1) - f(x_0)}$$

$$x_2 = 1 - \frac{(-3) \cdot (1 - 0.5)}{(-3) - (0.625)}$$

$$x_2 = 0.5862$$

$$x_3 = x_2 - \frac{f(x_2) \cdot (x_2 - x_1)}{f(x_2) - f(x_1)}$$

$$x_3 = 0.5862 - \frac{(-0.0744) \cdot (0.5862 - 1)}{(-0.0744) - (-3)}$$

$$x_3 = 0.57567$$

Algoritmo do Método da Secante

No Algoritmo 15 vemos o algoritmo do método da secante, conforme Campos (2007). Esse algoritmo foi implementado na linguagem Python e o resultado é o Programa 6.10.

Programa 6.10: Método da Secante

```
1  import numpy as np
2  import pandas as pd
3
4  def secante(f, va,vb, Toler, IterMax):
5      """
6      :param f: funcao
7      :param va: limite inferior
8      :param vb: limite superior
9      :param Toler: tolerancia
10     :param IterMax: numero maximo de iteracoes
11     :return: raiz,
12         Iter (numero de iteracoes gastas),
13         CondErro (condicao de erro, sendo
14             CondErro = 0 se a raiz foi encontrada
15             CondErro = 1 em caso contrario
16     """
17     a = va
18     b = vb
19     #avaliar a funcao em a e b
20     Fa = f(a)
21     Fb = f(b)
22     if(abs(Fa) < abs(Fb)):
23         t = a
24         a = b
25         b = t
26         t = Fa
27         Fa = Fb
28         Fb = t
29     Iter = 1
```

```
30      x = b
31      Fx = Fb
32      rebis = []
33      while(True):
34          DeltaX = -Fx*((b-a)/(Fb-Fa))#-Fx/(Fb-Fa))*(b-a)
35          x = x + DeltaX
36          Fx = f(x) #avaliar a funcao em x
37          rebis.append([Iter,a,Fa,b,Fb,x,Fx,DeltaX])
38
39          if(((abs(DeltaX) <= Toler) and (abs(Fx)<=Toler))
40              or (Iter>=IterMax)):
41              break
42          a = b
43          Fa = Fb
44          b = x
45          Fb = Fx
46          Iter +=1
47
48      raiz = x
49      rebis_table = pd.DataFrame(rebis,
50          columns=["Iter","a","f(a)","b","f(b)",
51              "x","f(x)","delta_x"])
52      #teste de convergencia
53      if((abs(DeltaX) <= Toler) and (abs(Fx) <= Toler)):
54          CondErro = 0
55          msg_erro = ""
56      else:
57          CondErro = 1
58          msg_erro = "Erro. O metodo nao convergiu."
```

```
59          rebis_table = []
60      return raiz, Iter, rebis_table, msg_erro, CondErro
```

6.4 Exercícios

1. Calcule os limites e o número de raízes reais de $P(x) = 2x^3 - 4x^2 - 6x + 5 = 0$.

2. Implemente em Python o método do ponto fixo para encontrar raiz real dado um intervalo, uma tolerância e um número máximo de iterações.

3. Usando o método da bisseção com $\varepsilon \leq 0.001$, calcule pelo menos uma raiz de cada equação abaixo. Utilize o Programa 6.6 para validar seus resultados.

   (a) $f(x) = e^{2x} - 2x^3 - 5 = 0$

   (b) $f(x) = 2x^3 - 5x^2 - x + 3 = 0$

   (c) $f(x) = 5x^2 + log_{10}(x + 1) - 2 = 0$

4. Ache pelo menos uma raiz de cada equação a seguir com $\varepsilon \leq 10^{-4}$ usando os métodos: secante e falsa posição. Faça a comparação do número de iterações gasto por cada método. Utilize os Programas 6.10 e 6.7 para validar seus resultados.

   (a) $f(x) = 2x^3 - 5x^2 - 10x + 20 = 0$

   (b) $f(x) = 5log_{10}(x) + 3x^4 - 7 = 0$

   (c) $f(x) = 2^x + cos(x)x^2 = 0$

5. Ache pelo menos uma raiz positiva de cada equação a seguir com $\varepsilon \leq 10^{-5}$ pelo método de Newton-Raphson. Utilize o Programa 6.9 para validar seus resultados.

   (a) $f(x) = 4x^3 + x + cos(x) - 10 = 0$

   (b) $f(x) = x^4 - 2x^3 + 2x - 1 = 0$

   (c) $f(x) = (x - 2)(e^{x-2} - 1) = 0$

6. Usando os Programas 6.6, 6.7, 6.9 e 6.10 compare o desempenho desses métodos para o cálculo de uma raiz das equações a seguir com $Toler = 10^{-10}$ e $IterMax = 500$.

   (a) $f(x) = 5x^3 - 2x^2 + 8x - 10 = 0$, $\xi \in [0, 2]$

   (b) $f(x) = 2x^3 + 5x^2 - sen(x) - 30 = 0$, $\xi \in [1, 4]$

   (c) $f(x) = e^{-x^2} cos(x) = 0$, $\xi \in [-1, 2]$

   (d) $f(x) = (x + 1)(x - 1)(x - 3)^5 = 0$, $\xi \in [2, 5]$

   (e) $f(x) = (x + 2)^3 \sqrt{x^2 + 1} = 0$, $\xi \in [-3, 0]$

Capítulo 7

Solução Numérica de Equações Diferenciais Ordinárias

7.1 Problema do valor inicial e de contorno

7.1.1 Introdução

Equações diferenciais aparecem com grande frequência em modelos que descrevem quantitativamente fenômenos em diversas áreas, como por exemplo, problemas biológicos, economia, química, mecânica de fluidos, fluxo de calor etc.

Exemplo:

Considere o circuito mostrado na Figura 7.1. A caixa representa um diodo de Esaki com a função característica $f(v)$ representando a corrente como função da tensão v. As Leis de Kirchoff aplicadas a esse circuito nos fornecem a seguinte entre a corrente i e a tensão v, como mostrado no sistema da Equação 7.1.

$$\begin{cases} L\frac{di}{dt} & = E - R_i - v = I(i,v) \\ -C\frac{dv}{dt} & = f(v) - i = V(i,v) \end{cases} \tag{7.1}$$

As equações do Exemplo são chamadas *equações diferenciais*, uma vez que envolvem derivada de funções.

Se uma equação diferencial tem apenas uma variável independente, então ela é uma *equação diferencial ordinária* (EDO). Se a equação diferencial envolve mais de uma variável

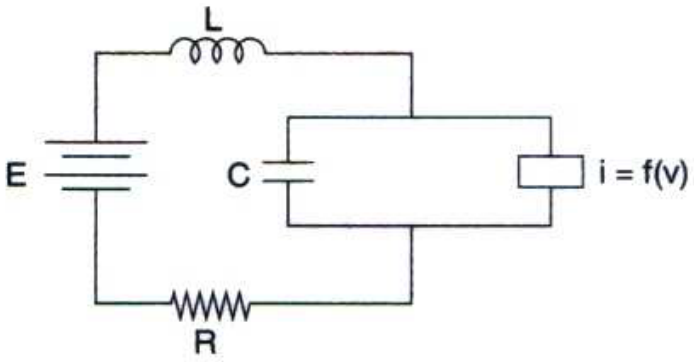


Figura 7.1: Exemplo de aplicação de EDO.

independente, então ela é uma *equação diferencial parcial* (EDP), como:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \text{ com } u \equiv u(x, y) \text{ e } \frac{\partial^2 u}{\partial .^2}$$

Uma solução de uma EDO é uma função da variável independente que satisfaça a equação. Assim:

i) $\frac{dy}{dx} = \dot{y} = y$, tem $y(x) = ax^2$, $a \in \mathbb{R}$ como solução.

ii) $u''' = 0$ é satisfeita para $u(x) = p_2(x)$ onde $p_2(x)$ é qualquer polinômio de grau 2.

Uma equação diferencial possui uma família de soluções e não apenas uma, como mostrado por exemplo na Figura 7.2.

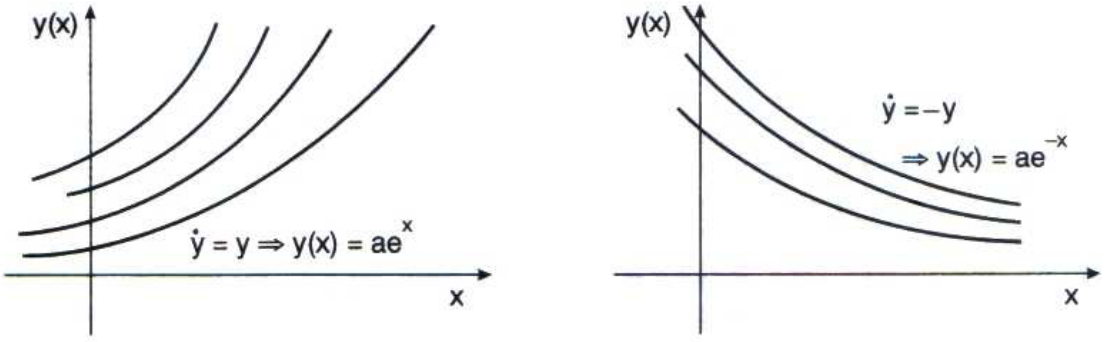


Figura 7.2: Exemplo de famílias de soluções de equações diferenciais.

A ordem de uma ED é a mais alta ordem de derivação que aparecer na equação.

Uma equação diferencial ordinária é dita *linear* se a função e suas derivadas aparecem linearmente na equação. Assim, $xy' = x - y$ é linear.

$y'' + (1 - y^2)y' + y = 0$ e $u'' + e^{-u} = f(x)$ são *não lineares*.

Para individualizar uma solução temos que importar condições suplementares. Em geral, uma equação de ordem m requer m condições adicionais a fim de ter uma única solução. Essas condições podem ser de qualquer tipo, por exemplo: $y(0) = 1$, $y'(4) = 5$, $y(2) + 5y'(3) = 6$, $\int_0^1 sen\, xy(x)dx = 0$, $\lim_{x\to\infty} y(x) = k$.

Se, dada uma equação de ordem m, a função, assim como suas derivadas até ordem $m-1$, são especificadas em um mesmo ponto, então temos um *problema de valor inicial*, PVI, como por exemplo:

a) $\begin{cases} y'(x) = y \\ y(0) = 1 \end{cases}$

b) $\begin{cases} y''' + (x+1)y'' + \cos xy' - (x^2-1)y = \quad x^2 + y^2 sen(x+y) \\ \quad y(0) = 1.1, y'(0) = 2.2, y''(0) = 3.3 \end{cases}$

Se em equações diferenciais ordinárias de ordem m, $m \geqslant 2$, as condições fornecidas para busca de solução única não são todas dadas num mesmo ponto, então temos um *problema de valor de contorno*, PVC.

Ao contrário do que ocorre com os PVI, é comum que problemas de contorno não tenham unicidade de solução. Por exemplo, para todo $\alpha \in \mathbb{R}, y(x) = \alpha(1 + x)$ é a solução do PVC:

$$\begin{cases} y'' = 0 \\ y(-1) = 0 \\ y(1) - 2y'(1) = 0 \end{cases}$$

7.1.2 Problema de valor inicial

A motivação para estudarmos métodos numéricos para aproximar soluções de PVI é a dificuldade de se encontrar, analiticamente, as soluções da equação.

Dado o PVI: $\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$

Se para calcular y_j usamos apenas y_{j-1}, teremos um *método de passo simples* ou *passo um*. Porém, se usarmos mais valores, teremos um *método de passo múltiplo*.

Se estamos trabalhando com PVI de primeira ordem, temos uma aproximação inicial $y(x_0)$ para a solução. Já para os métodos de passo múltiplo, temos que ter estratégias para obtermos as aproximações iniciais necessárias.

Outras características dos métodos de passo simples são:

i) em geral, temos que calcular o valor de $f(x, y)$ e suas derivadas em muitos pontos;

ii) temos dificuldades em estimar o erro.

7.1.3 Método de Euler

Esse método está dentro do conjunto de métodos de passo simples, ou passo um. É o método da série de Taylor de ordem 1.

Dado o PVI:

$$y' = f(x, y), y(x_0) = y_0$$

podemos calcular uma solução aproximada com o método de Euler, o qual consiste em: como conhecemos x_0 e $y_0 = y(x_0)$, então sabemos calcular $y'(x_0) = f(x_0, y_0)$. Assim, a reta que passa por (x_0, y_0) com coeficiente angular $y'(x_0)$, que chamaremos $r_0(x)$, é:

$$r_0(x) = y(x_0) + (x - x_0)y'(x_0)$$

Escolhido $h = x_{k+1} - x_k$, $y(x_1) \cong y_1 = r_0(x_1) = y_0 + hy'(x_0)$, ou seja,

$$y_1 = y_0 + hf(x_0, y_0)$$

O raciocínio é repetido com (x_1, y_1) e $y_2 = y_1 + hf(x_1, y_1)$ e assim, sucessivamente, o método de Euler nos fornece:

$$y_{k+1} = y_k + hf(x_k, y_k),\ k = 0, 1, 2, \ldots$$

Graficamente:

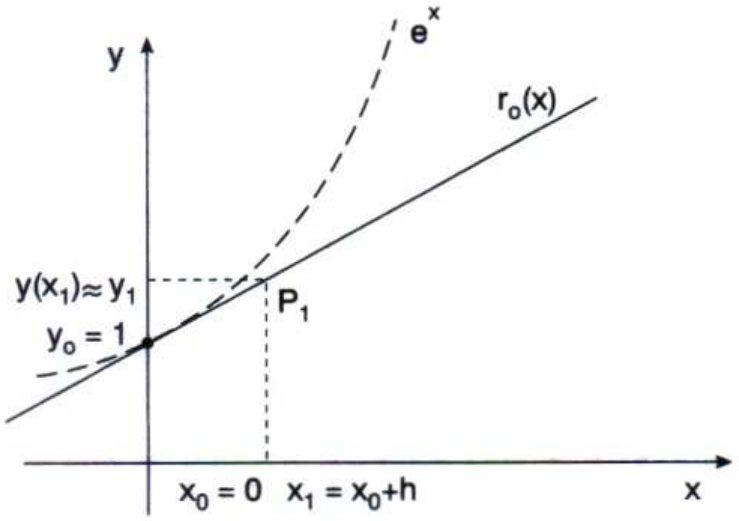


Figura 7.3: Esquema gráfico do método de Euler.

Então, resumidamente, para o método de Euler temos:

$$y' = f(x, y)$$

$$x_{k+1} = x_k + h$$

$$y_{k+1} = y_k + hf(x_k, y_k)$$

Exemplo 1:

Dado o PVI: $\begin{cases} xy' = x - y \\ y(2) = 2 \end{cases}$

Calcule $y(2.2)$ pelo método de Euler, considerando $h = 0.1$.

1º) Determinar y'.

$$y' = \frac{x - y}{x}$$

2º) Retirar as informações da condição inicial.

$$x_0 = 2 \qquad y_0 = y(x_0) = y(2) = 2$$

3º) Com os valores de x_0 e y_0, podemos calcular os próximos.

$$x_1 = x_0 + h$$
$$x_1 = 2 + 0.1$$
$$\boxed{x_1 = 2.1}$$

$$y_1 = y(x_1) = y(2.1)$$
$$y_1 = y_0 + hf(x_0, y_0) \qquad f(x_0, y_0) = f(2, 2)$$
$$y_1 = 2 + 0.1 \cdot 0 \qquad f(2, 2) = \frac{2-2}{2} = 0$$
$$\boxed{y_1 = 2}$$

$$x_2 = x_1 + h$$
$$x_2 = 2.1 + 0.1$$
$$\boxed{x_2 = 2.2}$$

$$y_2 = y(x_2) = y(2.2)$$
$$y_2 = y_1 + hf(x_1, y_1) \qquad f(x_1, y_1) = f(2.1, 2)$$
$$y_2 = 2 + 0.1 \cdot 0.0476 \qquad f(2.1, 2) = \frac{2.1-2}{2.1} = 0.0476$$
$$\boxed{y_2 = y(2.2) = 2.00476}$$

Então, $\boxed{y(2.2) = 2.00476}$

Algoritmo do método e implementação em Python

O algoritmo apresentado aqui (Algoritmo 16) é o definido por CAMPOS (2007). O Programa 7.1 implementa esse algoritmo em Python.

Programa 7.1: Implementação em Python do Algoritmo de Euler

```
1  import numpy as np
2  import pandas as pd
3
4  def fxy(x,y):
5      return x/y - y/x
6
7  def euler(a,b,m,x0,y0):
8      """
9      :param fxy: funcao em (x,y)->y'
10     :param a: limite inferior
11     :param b: limite superior
12     :param m: numero de subintervalos
13     :param y0: valor inicial
```

```
14      :return: h e tabela solucao do PVI
15      """
16
17      h = (b-a)/m
18      x = x0
19      y = y0
20      VetX = []
21      VetY = []
22      #avaliar f(x,y) em x = x0 e y = y0
23      Fxy = fxy(x,y)
24      VetX.append(x)
25      VetY.append(y)
26      tab = np.zeros([m+1,4], dtype=float)
27      tab[0] = [0, x, y, Fxy]
28      for i in range(1,m+1):
29          x = a + i * h
30          y = y + h * Fxy
31          #avaliar f(x,y) em x = xi e y = yi
32          Fxy = fxy(x,y)
33          tab[i] =[i,x,y,Fxy]
34          VetX.append(x)
35          VetY.append(y)
36      table = pd.DataFrame(tab, columns=["i", "x", "y", "Fxy"])
37      return h, table
38
39  if __name__ == "__main__":
40      h, tabela = euler(1,2,10,1,2)
41      print(tabela)
```

Exemplo 2: Vamos usar este exemplo para resolver usando Euler (1ª ordem), Euler Aperfeiçoado (2ª ordem) e Runge Kutta (3ª ordem).

Dado o PVI: $\begin{cases} (y^2 - x^2)dx + xydy = 0 \\ y(1) = 2 \end{cases}$

Aproxime $y(1.2)$ usando $h = 0.1$.

A solução analítica desse problema é $y(1.2) = 1.7750$.

ITER (k)	0	1	2
x	1	1.1	1.2
y	2	?	?

Antes de começarmos a resolver, precisamos achar o y'.

$$y' = f(x, y) = \frac{dy}{dx}$$

$$xydy = (x^2 - y^2)dx$$

$$\frac{dy}{dx} = \frac{x^2 - y^2}{xy} = \frac{x}{y} - \frac{y}{x} = f(x, y)$$

$$y' = \frac{x}{y} - \frac{y}{x}$$

Euler (1ª ordem): $y_{k+1} = y_k + hy'_k$

1º passo: $x_0 = 1, y_0 = 2, h = 0.1$

$y_1 = y_0 + hy'_0$

$y_1 = 2 + 0.1\left(\frac{x_0}{y_0} - \frac{y_0}{x_0}\right)$

$y_1 = 2 + 0.1\left(\frac{1}{2} - \frac{2}{1}\right)$

$y_1 = 1.85$

2º passo: $x_1 = 1.1, y_1 = 1.85, h = 0.1$

$y_2 = y_1 + hy'_1$

$y_2 = 1.85 + 0.1\left(\frac{x_1}{y_1} - \frac{y_1}{x_1}\right)$

$y_2 = 1.85 + 0.1\left(\frac{1.1}{1.85} - \frac{1.85}{1.1}\right)$

$y_2 = 1.7413$

Podemos verificar, a partir da solução analítica desse PVI ($y(1.2) = 1.7750$), que a solução aproximada encontrada utilizando o método de Euler está "não muito boa".

Vamos calcular os erros absoluto e relativo para essa aproximação.

$\mathrm{E}_{abs} = valor\_real - valor\_aprox$

$E_{abs} = 1.7750 - 1.7413$

$E_{abs} = 0.0337$

$E_{rel} = \frac{E_{abs}}{valor\_real}$

$E_{rel} = \frac{0.0337}{1.7750}$

$E_{rel} = 1.89\%$

7.1.4 Método de Euler Aperfeiçoado - Runge Kutta 2ª ordem

Podemos observar a partir da solução real do problema exemplo, que a nossa solução aproximada pelo método de Euler pode ser melhorada. Temos igualdade somente da primeira casa decimal ao comparar a solução analítica com a aproximada.

O método de Euler Aperfeiçoado, ou Runge Kutta de 2ª ordem, pode ser resumido pela equação:

$y_{k+1} = y_k + \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y^*_{k+1})\right]$

onde,

$y^*_{k+1} = y_k + hy'_k$ ou $y^*_{k+1} = y_k + hf(x_k, y_k)$

Usando o método de Euler aperfeiçoado, calcularemos o valor aproximado de $y(1.2)$ do Exemplo 2.

1º passo: $k = 0$

$y_k^* = y_0 + 0.1y_0'$

$y_1^* = 2 + 0.1\left[\frac{1}{2} - \frac{2}{1}\right]$

$y_1^* = 1.85$

$y_1 = y_0 + \frac{0.1}{2}\left[f(x_0, y_0) + f(x_1, y_1^*]\right.$

$y_1 = 2 + \frac{0.1}{2}\left[\left(\frac{1}{2} - \frac{2}{1}\right) + \left(\frac{1.1}{1.85} - \frac{1.85}{1.1}\right)\right]$

$y_1 = 1.8706$

2º passo: $k = 1$

$y_2^* = y_1 + 0.1y_1'$

$y_1^* = 1.8706 + 0.1\left[\frac{1.1}{1.8706} - \frac{1.8706}{1.1}\right]$

$y_1^* = 1.7594$

$y_2 = y_1 + \frac{0.1}{2}\left[f(x_1, y_1) + f(x_2, y_2^*]\right.$

$y_2 = 1.8706 + \frac{0.1}{2}\left[\left(\frac{1.1}{1.8706} - \frac{1.8706}{1.1}\right) + \left(\frac{1.2}{1.7594} - \frac{1.7594}{1.2}\right)\right]$

$y_2 = 1.7758$

A partir da solução analítica desse PVI ($y(1.2) = 1.7750$), a solução aproximada, usando o Método de Euler Aperfeiçoado, melhorou. Vamos confirmar com o cálculo dos erros.

$\mathrm{E}_{abs} = valor\_real - valor\_aprox$

$E_{abs} = 1.7750 - 1.7758$

$E_{abs} = 0.0008$

$E_{rel} = \frac{E_{abs}}{valor\_real}$

$E_{rel} = \frac{0.0008}{1.7750}$

$E_{rel} = 0,045\%$

Implementação do método em Python

O Programa 7.2 implementa o método de Runge-Kutta de 2ª ordem (Euler Aperfeiçoado), em Python.

Programa 7.2: Implementação do Método de Runge-Kutta de 2ª ordem

```
1 import numpy as np
2 import pandas as pd
3
4 def fxy(x,y):
5     return x/y - y/x
6
7 def runge_kutta_2(x0, y0, a, b, n):
8     # Calcular o tamanho do passo
9     h = (b - a) / n
```

```
10
11      # Vetores para armazenar os valores de x e y
12      x = np.zeros(n + 1)
13      y = np.zeros(n + 1)
14
15      # Definir os valores iniciais
16      x[0] = x0
17      y[0] = y0
18
19      # Loop do Runge Kutta de 2a ordem
20      # Loop para calcular os valores de y em cada passo
21      for i in range(n):
22          # Calcular k1 e k2
23          k1 = fxy(x[i], y[i])
24          k2 = fxy(x[i] + h/2, y[i] + (h*k1)/2)
25
26          # Calcular o pr ximo valor de y
27          y[i+1] = y[i] + h * k2
28
29          # Atualizar o valor de x
30          x[i+1] = x[i] + h
31
32      table_result = pd.DataFrame([x,y]).
33          pivot_table(columns=["x","y"])
34
35      return h, table_result
36
37  if __name__ == "__main__":
38      h, tabela = runge_kutta_2(1,2,1,2,10)
```

```
39      print(tabela)
```

Ainda podemos melhorar essa aproximação. A seguir apresentamos o Método de Runge Kutta de 3ª ordem.

7.1.5 Método de Runge Kutta de 3ª ordem

Esse método pode ser resumido na seguinte equação.

$y_{k+1} = y_k + \ [2k_1 + 3k_2 + 4k_3]$

onde,

$k_1 = hf\left(x_k, y_k\right)$

$k_2 = hf\left(x_k + \frac{h}{2}, y_k + \frac{k_1}{2}\right)$

$k_3 = hf\left(x_k + \frac{3h}{4}, y_k + \frac{3k_2}{4}\right)$

Observação: Não confundir o k das equações auxiliares com o k da iteração (passo).

1º passo: $k = 0,\ x_0 = 1,\ y_0 = 2$

Cálculos de k_1, k_2 e k_3:

$k_1 = 0.1\left[\frac{1}{2} - \frac{2}{1}\right] \quad \therefore \quad k_1 = -0.15$

$k_2 = 0.1f\left(1 + \frac{0.1}{2}, 2 + \frac{-0.15}{2}\right)$

$k_2 = 0.1f\left(1.05, 1.925\right)$

$k_2 = 0.1\left[\frac{1.05}{1.925} - \frac{1.925}{1.05}\right] \quad \therefore \quad k_2 = -0.1288$

$k_3 = 0.1f\left(1 + \frac{3}{4}\cdot 0.1, 2 + \frac{3}{4}\cdot(-0.1288)\right)$

$k_3 = 0.1f(1.075, 1.9034) \quad \therefore \quad k_3 = -0.1206$

Cálculo de y_1:

$y_1 = y_0 + \frac{1}{9}\left[2k_1 + 3k_2 + 4k_3\right]$

$y_1 = 2 + \frac{1}{9}\left[2\cdot(-0.15) + 3\cdot(-0.1288) + 4\cdot(-0.1206)\right]$

$y_1 = 1.8701$

2º passo: $k = 1,\ x_1 = 1.1,\ y_1 = 1.8701$

Cálculos de k_1, k_2 e k_3:

$k_1 = hf\left(x_1 + \frac{h}{2}, y_1 + \frac{k_1}{2}\right) \quad \therefore \quad k_1 = -0.1112$

$k_2 = hf\left(x_1 + \frac{h}{2}, y_1 + \frac{k_1}{2}\right)$

$k_2 = 0.1\cdot f\left(1.1 + \frac{0.1}{2}, 1.8701 + \frac{(-0.1112)}{2}\right)$

$$k_2 = 0.1 \cdot f(1.15, 1.8145) \quad \therefore \quad k_2 = -0.0944$$

$$k_3 = h \cdot f\left(x_1 + \tfrac{3h}{4}, y_1 + \tfrac{3k_2}{4}\right)$$

$$k_3 = 0.1 \cdot f\left[\left(1.1 + \tfrac{3 \cdot 0.1}{4}\right), \left(1.8701 + \tfrac{3 \cdot (-0.0944)}{4}\right)\right]$$

$$k_3 = 0.1 \cdot f(1.175, 1.7993) \quad \therefore \quad k_3 = -0.0878$$

Cálculo de y_2:

$$y_2 = 1.8701 + \ [2(-0.1112) + 3(-0.0944) + 4(-0.0878)]$$

$$y_2 = 1.7749 \quad \approx \quad \text{solução exata.}$$

$$\mathrm{E}_{abs} = valor\_real - valor\_aprox \qquad E_{rel} = \tfrac{E_{abs}}{valor\_real}$$

$$E_{abs} = 1.7750 - 1.7749 \qquad E_{rel} = \tfrac{0.0001}{1.7750}$$

$$E_{abs} = 0.0001 \qquad E_{rel} = 0,0056\%$$

Implementação do método em Python

O Programa 7.3 implementa o método de Runge-Kutta de 3ª ordem em Python.

Programa 7.3: Implementação do Método de Runge-Kutta de 3ª ordem

```
1  import numpy as np
2  import pandas as pd
3
4  def fxy(x,y):
5      return x/y - y/x
6
7  def runge_kutta_2(x0, y0, a, b, n):
8      """
9      Implementa o metodo de Runge-Kutta de
10     3a ordem para resolver uma EDO.
11
12     Entrada:
```

```
13          fxy: A funcao que define a EDO (f(x, y)).
14          x0: Valor inicial de x.
15          y0: Valor inicial de y.
16          a: Valor inicial do intervalo
17          b: Valor final do intervalo
18          n: Numero de subdivisoes.
19
20      Saida:
21          Tamanho do passo (h) e a tabela de resultados (x, y)
22          com as solucoes aproximadas.
23      """
24      h = (b - a) / n
25      #results = [[x0, y0]]
26      # Vetores para armazenar os valores de x e y
27      x = np.zeros(n + 1)
28      y = np.zeros(n + 1)
29      x[0]=x0
30      y[0]=y0
31
32      for i in range(n):
33          k1 = h * fxy(x[i], y[i])
34          k2 = h * fxy(x[i] + h/2, y[i] + (k1/2))
35          k3 = h * fxy(x[i] + (3*h)/4, y[i] + (3*k2/4))
36
37          y[i+1] = y[i] + (1/9)*(2*k1+3*k2+4*k3)
38
39          x[i+1] = x[i] + h
40
41
```

```
42      table_result = pd.DataFrame(
43          [x,y]).pivot_table(columns=["x","y"])
44
45      return h, table_result
46
47  if __name__ == "__main__":
48      h, tabela = runge_kutta_3(1,2,1,2,10)
49      print(tabela)
```

7.1.6 Exercícios

1. Seja o PVI: $y' = y$, $y(0) = 1$. Trabalhando com quatro casas decimais, use o método de Euler para aproximar $y(0.04)$ com $\xi \leqslant 5 \times 10^{-4}$.

2. Calcular a solução do PVI: $y' = x - 2y + 1$, com $y(0) = 1$, no intervalo $[0, 1]$, com $m = 10$ subintervalos, usando os Programas 7.1, 7.2 e 7.3.

Referências

CAMPOS, Frederico Ferreira. *Algoritmos numéricos*. 2. ed. Rio de Janeiro: LTC, 2007. 428 p., il.

DIVINO, Bruno. *Python* - Uma Introdução à Linguagem. Alura. Atualizado em: 15/10/2021. Disponível em: <https://www.alura.com.br/artigos/python-uma-introducao-a-linguagem> Acesso em: dezembro, 2022.

BURDEN, Richard L.; FAIRES, J. Douglas; BURDEN, Annette M.. *Análise Numérica*. Tradução All Tasks; Revisão técnica: Helena Maria Ávila de Castro. 3. ed. São Paulo: Cengage Learning, 2015.

RUGGIERO, Márcia A. Gomes; LOPES, Vera Lúcia da Rocha. *Cálculo numérico*: aspectos teóricos e computacionais. 2. ed. São Paulo: Pearson Makron Books, 1996. xvi, 406 p., il.

Apêndice A - Algoritmos

Algoritmo 1 Substituições Sucessivas.
Fonte: Campos(2007).

```
// Objetivo:  Resolver o sistema triangular inferior Lx = c
Entrada: n, L, c // ordem, matriz triangular inferior e vetor independente
Saida: x  // solução do sistema triangular inferior
x(1) ← c(1)/L(1,1)
for i ← 2 to n do
    Soma ← 0
    for j ← 1 to i − 1 do
        Soma ← Soma + L(i,j) * x(j)
    end
    x(i) ← (c(i) − Soma)/L(i,i)
end
```

Algoritmo 2 Substituições Retroativas.
Fonte: Campos (2007).

```
// Objetivo:  Resolver o sistema tringular superior Ux = d
Entrada: n, U, d // ordem, matriz triangular superior e vetor independente
Saida: x  // solução do sistema triangular superior
x(n) ← d(n)/U(n,n)
for i ← (n − 1) to 1 step -1 do
    Soma ← 0
    for j ← n to i do
        Soma ← Soma + L(i,j) * x(j)
    end
    x(i) ← (c(i) − Soma)/L(i,i)
end
```

Algoritmo 3 Eliminação de Gauss.
Fonte: Ruggiero e Lopes (1996).

```
// Fase de eliminação
for k ← 1 to n − 1 do
    for i ← k + 1 to n do
        m ← a(i, k)/a(k, k)
        a(k) ← 0
        for j ← k + 1 to n do
            a(i, j) ← a(i, j) − m ∗ a(k, j)
            b(j) ← b(i) − m ∗ b(k)
        end
    end
end
// Fase de resolução
x(n) ← b(n)/a(n, n)
for k ← n − 1 to n do
    s ← s + a(k, j) ∗ x(j)
    x(k) ← (b(k) − s)/a(k, k)
end
```

Algoritmo 4 Técnica Iterativa de Jacobi
Fonte: Burden et al. (2005).

```
for i ← 1 to n do
    x_old_i ← b_i/a_ii
end
it ← 1
while it ≤ maxiter do
    for i ← 1 to n do
        x_i ← 1/a_ii [ − Σ_{j=1j}^{n} (a_ij x_old_j) + b_i ]
        if x − x_old ≤ eps then
            Saida: (x_1, ..., x_n)
        end
        it ← it + 1
        for i ← 1 to n do
            x_old_i ← x_i
        end
    end
end
Saida: 'Número maximo de iterações excedido. O procedimento não foi bem sucedido.'
```

Algoritmo 5 Método Iterativo de Gauss-Seidel.
Fonte: Burden et al. (2005).

for $i \leftarrow 1$ **to** n **do**
  $x_{old_i} \leftarrow b_i/a_{ii}$
end
$it \leftarrow 1$
while $it \leq maxiter$ **do**
  for $i \leftarrow 1$ **to** n **do**
    $x_i \leftarrow \frac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^{n} (a_{ij} x_{old_j}) + b_i \right]$
    if $x - x_{old} < eps$ **then**
      Saida: $(x_1, \ldots, x_n)$
    end
    $it \leftarrow it + 1$
    for $i \leftarrow 1$ **to** n **do**
      $x_{old_i} \leftarrow x_i$
    end
  end
end
Saida: 'Número maximo de iterações excedido. O procedimento não foi bem sucedido.'

Algoritmo 6 Algoritmo de regressão linear múltipla e polinomial via equações normais. Fonte: CAMPOS (2002).

```
// Objetivo: Calcular parametros de quadrados minimos de modelo linear
   multiplo via equacoes normais
Entrada: n, v, p, x, y // numero de pontos, numero de variaveis, numero de
   parametros, variaveis explicativas, variaveis respostas
Saida: b, r2, sigma2, condErro // coef. de regressao, coef. de determinacao,
   variancia residual, cond de erro
if v ≥ 1 e v + 1 <> p then
    condErro ← 1;
    abandone;
end
condErro ← 0;
vp1 ← v + 1;
pm1 ← p − 1;
// inclusao de uma coluna de 1's relativa a b0
for i ← 1 to n do
    for j ← vp1 to 2 step −2 do
        x(i, j) ← x(i, j − 1)
    end
    x(i, 1) ← 1
end
// Se regressao polinomial, entao gera potencias de x
if v = 1 e p > 2 then
    for j = 2 to pm1 do
        jp1 ← j + 1
        for i = 1 to n do
            x(i, jp1) ←x(i,2)^j
        end
    end
end
```

Algoritmo 7 Algoritmo de regressão linear múltipla e polinomial via equações normais (Continuação).

```
// equacoes normais
for i ← 1 to p do
    for j ← 1 to p do
        Soma ← 0
        for k ← 1 to n do
            Soma ← Soma + x(k,i) * x(k,j)
        end
        Sxx(i,j) = Soma // matriz dos coeficientes
    end
    Soma ← 0
    for k ← 1 to n do
        Soma ← Soma + x(k,i) * y(k)
    end
    Sxy(i) ← Soma
end
[L, Det, condErro] ← Cholesky(p, Sxx) // decomposição de Cholesky
t = Substituicoes_Sucessivas(p, L, Sxy)
for i ← 1 to p do
    for j ← 1 to i do
        U(j,i) ← L(i,j) // U = transposta de L
    end
end
b = Substituicoes_Retroativas(p, U, t) // coeficientes
D ← 0; Sy2 ← 0
for i ← 1 to n do
    u ← 0;
    for j ← 1 to p do
        u ← u + b(j) * x(i,j)
    end
    d ← y(i) − u; D ← D + d^2; Sy2 ← Sy2 + y(i)^2
end
// Cálculo de R2
r2 ← 1 − D/(Sy2 − Sxy(1)^2/n) // coeficiente de determinação
// Cálculo da variância residual
sigma2 ← D/(n − p) // variância residual
Fim do algoritmo
```

Algoritmo 8 Algoritmo de Horner.

```
// Objetivo:  Avaliar um polinômio de grau n no ponto a
// Parâmetros de entrada:  n (grau), c (coeficientes), a (ponto a ser
   avaliado), onde c é tal que P(x) = c(1)x^n + c(2)x^{n-1} + ··· + c(n)x + c(n+1)
// Parâmetros de saída:  y (ordenada P(a) )
y ← c(1)
for i ← 2 to n+1 do
  y ← y * a + c(i)
end
```

Algoritmo 9 Algoritmo LimitesRaizes.
Fonte: CAMPOS(2007).

```
// Objetivo:  Achar os limites das raízes reais de uma equação polinomial
// Parâmetros de entrada:  n (grau do polinômio), c (coeficientes, sendo
   P(x) = c(1)x^n + c(2)x^(n-1) + ... + c(n)x + c(n+1))
// Parâmetros de saída:  L (limites inferior e superior das raízes positivas
   e negativas, respectivamente)
if  c(1) = 0 then
  escreva "coeficiente c(1) nulo";
  abandone;
end
t ← n;
c(t+1) ← 0;
// se c(n+1) for nulo, então o polinômio é deflacionado
repeat
  if c(t) != 0 then
    interrompa;
  end
  t ← t − 1;
until 1;
// Cálculo dos quatro limites das raízes reais
for 1 ← 2 to n+1 do
  y ← y * a + c(i)
end
for i ← 1 to 4 do
  // inversão da ordem dos coeficientes
  if i == 2 or i == 4 then
    for j ← 1 to t/2 do
      Aux ← c(j); c(j) ← c(t − j + 1); c(t − j + 1) ← Aux;
    end
  end
  ...
```

...

```
else
    // reinversão da ordem e troca de sinais dos coeficientes
    if i == 3 then
        for j ← 1 to t/2 do
            Aux ← c(j); c(j) ← c(t − j + 1); c(t − j + 1) ← Aux
        end
        for j ← t − 1 to 1 step −2 do
            c(j) ← −c(j)
        end
    end
end
// se c(1) for negativo, então é trocado o sinal de todos os coeficientes
if c(1) < 0 then
    for j ← 1 to t do
        c(j) ← −c(j)
    end
end
// cálculo de k, o maior índice dos coeficientes negativos
k ← 2 repeat
    if c(k) < 0 or k > t then
        interrompa;
    end
    k ← k + 1
until 1;
// Cálculo de B, o maior coeficiente negativo em módulo
if k <= t then
    B ← 0 for j ← 2 to t do
        if c(j) < 0 e abs(c(j)) > B then
            B ← abs(c(j))
        end
    end
    // Limite das raízes positivas de P(x) = 0 e das equações auxiliares
    L(i) ← 1 + sqrt[k − 1]B/c(1)
end
else
    L(i) ← 10^100
end
end
// limite das raízes positivas e negativas de P(x) = 0
Aux ← L(1); L(1) ← 1/L(2); L(2) ← Aux; L(3) ← −L(3);L(4) ← −1/L(4)
```

Algoritmo 11 Algoritmo TrocaSinal.
Fonte: CAMPOS(2007).

```
// Objetivo:  Achar um intervalo [a,b] onde uma função troca de sinal
// Parâmetros de entrada:  z(ponto a partir do qual o intervalo será gerado)
// Parâmetros de saída:  a(limite inferior), b(limite superior) e CondErro
   (condição de erro), sendo CondErro = 0 se f(a)f(b) <=0 e CondErro = se
   f(a)f(b) >0
if z = 0 then
  a ← −0.05; b ← 0.05;
end
else
  a ← 0.95 ∗ z; b ← 1.05 ∗ z;
end
Iter ← 0; Aureo ← 2/(sqrt(5) − 1)
// avaliar a função em a e b
Fa ← f(a); Fb ← f(b);
Escreva Iter, a, b, Fa, Fb
repeat
  if Fa ∗ Fb <= 0 ou Iter >= 20 then
    interrompa;
  end
  Iter ← Iter + 1 if abs(Fa) < abs(Fb) then
    a ← a − Aureo ∗ (b − a)
    // avaliar a função em a
    Fa ← f(a)
  end
  else
    b ← b + Aureo ∗ (b − a)
    // avaliar a função em b
    Fb ← f(b)
  end
  Escreva Iter, a, b, Fa, Fb
until 1;
if Fa ∗ Fb <= 0 then
  CondErro ← 0
end
else
  CondErro ← 1
end
```

Algoritmo 12 Algoritmo da Bisseção.
Fonte: CAMPOS(2007).

```
// Objetivo:  Calcular a raiz de uma equação pelo método da bisseção
// Parâmetros de entrada:  a (limite inferior), b (limite superior), Toler
   (tolerância) e IterMax (número máximo de iterações)
// Parâmetros de saída:  Raiz (raiz), Iter (número de iteraões gastas) e
   CondErro (condição de erro, sendo CondErro = 0 se a raiz foi encontrada e
   CondErro = 1 em caso contrário)
// avaliar a função em a e b
Fa ← f(a); Fb ← f(b);
if Fa * Fb > 0 then
    Escreva 'Função não muda de sinal nos extremos do intervalo dado';
    abandone;
end
DeltaX ← abs(b − a)/2; Iter ← 0;
repeat
    // avaliar a função em x
    x ← (a + b)/2; Fx ← f(x);
    Escreva Iter, a, Fa, b, Fb, x, Fx, DeltaX
    if (DeltaX <= Toler e abs(Fx) <= Toler) ou Iter >= IterMax then
        interrompa;
    end
    if Fa*Fx > 0 then
        a ← x; Fa ← Fx;
    end
    else
        b ← x;
    end
    Deltax ← DeltaX/2; Iter ← Iter + 1;
until 1;
Raiz ← x // Teste de convergência
if DeltaX <= Toler e abs(Fx) <= Toler then
    CondErro ← 0
end
else
    CondErro ← 1
end
```

Algoritmo 13 Algoritmo da Falsa Posição.
Fonte: CAMPOS(2007).

```
// Objetivo:  Calcular a raiz de uma equação pelo método da falsa posição
// Parâmetros de entrada:  a (limite inferior), b (limite superior), Toler
   (tolerância) e IterMax (número máximo de iterações)
// Parâmetros de saída:  Raiz (raiz), Iter (número de iterações gastas) e
   CondErro (condição de erro, sendo CondErro = 0 se a raiz foi encontrada e
   CondErro = 1 em caso contrário)
// avaliar a função em a e b
Fa ← f(a); Fb ← f(b);
if Fa * Fb > 0 then
  | Escreva 'Função não muda de sinal nos extremos do intervalo dado';
  | abandone;
end
DeltaX ← abs(b − a)/2; Iter ← 0;
if f(a) > 0 then
  | t ← a; a ← b; b ← Fa; Fa ← Fb; Fb ← t
end
Iter ← 0; x ← b; Fx ← Fb
repeat
  | DeltaX ← −Fx/(Fb − Fa) * (b − a)
  | // avaliar a função em x
  | x ← x + DeltaX; Fx ← f(x);
  | Escreva Iter, a, Fa, b, Fb, x, Fx, DeltaX
  | if abs(DeltaX <= Toler e abs(Fx) <= Toler) ou Iter >= IterMax then
  |   | interrompa;
  | end
  | if Fx < 0 then
  |   | a ← x; Fa ← Fx;
  | end
  | else
  |   | b ← x; Fb ← Fx
  | end
  | Iter ← Iter + 1
until 1;
Raiz ← x // Teste de convergência
if abs(DeltaX) <= Toler e abs(Fx) <= Toler then
  | CondErro ← 0
end
else
  | CondErro ← 1
end
```

Algoritmo 14 Algoritmo Newton.
Fonte: CAMPOS(2007).

```
// Objetivo:  Calcular a raiz de uma equação pelo método de Newton
// Parâmetros de entrada:  x0 (valor inicial), Toler (tolerância) e IterMax
   (número máximo de iterações)
// Parâmetros de saída:  Raiz (raiz), Iter (número de iteracoes gastas) e
   CondErro (condição de erro, sendo CondErro = 0 se a raiz foi encontrada e
   CondErro = 1 em caso contrário)
// avaliar a função e sua derivada em x0
Fx ← f(x0); DFx ← f'(x0); x ← x0; Iter ← 0
Escreva Iter, x DFx, Fx
repeat
    DeltaX ← −Fx/DFx; x ← x + DeltaX
    // avaliar a função e sua derivada em x
    Fx ← f(x0); DFx ← f'(x0)
    Iter ← Iter + 1
    Escreva Iter, x, DFx, Fx, DeltaX
    if abs(DeltaX <= Toler e abs(Fx) <= Toler) ou Iter >= IterMax then
        interrompa;
    end
until 1;
Raiz ← x // Teste de convergência
if abs(DeltaX) <= Toler e abs(Fx) <= Toler then
    CondErro ← 0
end
else
    CondErro ← 1
end
```

Algoritmo 15 Algoritmo da Secante.
Fonte: CAMPOS(2007).

```
// Objetivo:  Calcular a raiz de uma equação pelo método da secante
// Parâmetros de entrada:  a (limite inferior), b (limite superior), Toler
   (tolerância) e IterMax (número máximo de iterações)
// Parâmetros de saída:  Raiz (raiz), Iter (número de iterações gastas) e
   CondErro
// avaliar a função em a e b
Fa ← f(a); Fb ← f(b);
if Fa * Fb > 0 then
 | Escreva 'Função não muda de sinal nos extremos do intervalo dado'; abandone;
end
if abs(Fa) < abs(Fb) then
 | t ← a; a ← b; b ← t; t ← Fa; Fa ← Fb; Fb ← t
end
Iter ← 0; x ← b; Fx ← Fb
repeat
 | DeltaX ← −Fx/(Fb − Fa) * (b − a)
 | // avaliar a função em x
 | x ← x + DeltaX; Fx ← f(x);
 | Escreva Iter, a, Fa, b, Fb, x, Fx, DeltaX
 | if abs(DeltaX <= Toler e abs(Fx) <= Toler) ou Iter >= IterMax then
 |  | interrompa;
 | end
 | a ← b; Fa ← Fb; b ← x; Fb ← Fx; Iter ← Iter + 1
until 1;
Raiz ← x // Teste de convergência
if abs(DeltaX) <= Toler e abs(Fx) <= Toler then
 | CondErro ← 0
end
else
 | CondErro ← 1
end
```

Algoritmo 16 Algoritmo de Euler.
Fonte: CAMPOS(2007).

```
// Objetivo:  Resolver um PVI pelo metodo de Euler
// parametros de entrada:  a (inf), b (sup), m (num subintervalos), y0
   (valor inicial)
// parametros de saida:  VetX ; VetY (abscissas e solucao do PVI)
```

$h \leftarrow (b-a)/m; x \leftarrow a; y \leftarrow y0$

```
// avaliar f(x; y) em x = x0 e y = y0
```

$Fxy = f(x; y)$
$VetX(1) \leftarrow x; VetY(1) = y;$
for $i = 1$ *to* m **do**
  $x \leftarrow a + i * h; y = y + h * Fxy$
  `// avaliar f(x; y) em x = xi e y = yi`
  $Fxy \leftarrow f(x; y)$
  Escreva i, x, y, Fxy
  $VetX(i+1) \leftarrow x; VetY(i+1) \leftarrow y;$
end
return $VetX, VetY$

Métodos Numéricos Computacionais

Com Implementações em Python

Métodos Numéricos Computacionais
Com Implementações em Python

Sobre o livro:

Este e-book apresenta, de forma clara e objetiva, os principais métodos numéricos aplicados à engenharia, acompanhados de algoritmos em pseudocódigo e implementações completas em Python 3. A obra aborda erros numéricos, solução de sistemas lineares, interpolação, ajuste de curvas, regressão múltipla, integração numérica e resolução de EDO's, integrando teoria e prática de maneira acessível.

Inclui, ainda, um capítulo introdutório de Python e uma calculadora online com todos os métodos estudados, permitindo ao leitor experimentar, visualizar e validar resultados de forma interativa.

Um material didático essencial para estudantes e profissionais que desejam compreender e aplicar métodos numéricos com eficiência no contexto computacional moderno.

Sobre a autora:

Rosana Massahud é graduada em Ciência da Computação e mestre em Engenharia de Sistemas, com ênfase em Modelagem Matemática de Sistemas Biológicos. Professora do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) – Campus Nepomuceno, atua nas áreas de Banco de Dados, Análise de Dados, Modelagem Computacional e Desenvolvimento de Sistemas.

Dedica-se ao ensino, à pesquisa e ao desenvolvimento de materiais didáticos que aproximam a prática computacional das necessidades acadêmicas e profissionais, com foco na formação crítica e aplicada dos estudantes.

ISBN: 978-65-87888-46-0