Resolução Numérica de Equações Algébricas Não-Lineares – Parte 1

EMC410235 - Programação Científica para Engenharia e Ciência Térmicas

Prof. Rafael F. L. de Cerqueira

2025.2

Objetivos da Aula

- Compreender o que caracteriza uma equação algébrica não-linear.
- Estudar o método de Newton-Raphson e da Bisseção como ferramentas de resolução.
- Utilizar a biblioteca `scipy.optimize` para encontrar raízes numéricas.
- Comparar métodos manuais com soluções automatizadas (`fsolve`).
- Resolver e interpretar exemplos simples de aplicação.

Motivação

- Equações algébricas não-lineares aparecem com frequência em problemas de engenharia:
  - Troca de calor com radiação:

$$q = h(T_s - T_\infty) + \epsilon\sigma(T_s^4 - T_{\mathrm{amb}}^4)$$

  - Fator de atrito em escoamentos internos: equação de Colebrook

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\varepsilon/D}{3{,}7} + \frac{2{,}51}{Re\sqrt{f}}\right)$$

  - Eficiência de aletas

$$\frac{\tanh(mL)}{mL} = \eta_f$$

- Muitas vezes, essas equações não admitem solução analítica.
- Precisamos de métodos numéricos para encontrar soluções aproximadas.

Exemplo Motivador

- Considere a equação:

$$f(x) = x^3 - x - 2$$

- Essa é uma equação algébrica não-linear simples.
- Vamos buscar a raiz tal que $f(x) = 0$.
- Podemos visualizar essa raiz observando o gráfico de $f(x)$.

Vamos plotar?

Método de Newton-Raphson – Derivação

- Objetivo: encontrar a raiz de $f(x) = 0$
- Suponha uma aproximação inicial x_n
- Expandindo $f(x)$ em série de Taylor de primeira ordem ao redor de x_n:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n)$$

- Queremos $f(x_{n+1}) = 0 \Rightarrow$

$$0 \approx f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

- Isolando x_{n+1}:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Método de Newton-Raphson

- Método iterativo baseado em aproximação por tangentes:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- A ideia é usar a reta tangente à curva $f(x)$ para encontrar aproximações sucessivas da raiz.
- Requer:
  - Expressão para a derivada $f'(x)$
  - Um “chute” inicial x_0
- Convergência rápida se x_0 estiver perto da raiz.

Código: Método de Newton-Raphson

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

def df(x):
 return 3*x**2 - 1

def newton(x0, tol=1e-6, max_iter=20):
 for i in range(max_iter):
  x1 = x0 - f(x0)/df(x0)
  if abs(x1 - x0) < tol:
   return x1
  x0 = x1
 return None
```

```
raiz = newton(1.5)
print("Raiz:", raiz)
```

Método da Bisseção – Ideia Geral

- Requer um intervalo $[a, b]$ tal que $f(a) \cdot f(b) < 0$
- Baseia-se na continuidade de $f(x)$ e na existência de raiz no intervalo
- Iterativamente divide o intervalo pela metade:

$$x_{\text{médio}} = \frac{a+b}{2}$$

- Escolhe o subintervalo onde há mudança de sinal
- Processo se repete até atingir a tolerância desejada
- Convergência garantida, porém lenta

Código: Método da Bisseção

```
def bissecao(f, a, b, tol=1e-6):
 if f(a)*f(b) >= 0:
  return None
 while (b - a)/2 > tol:
  c = (a + b)/2
  if f(c) == 0:
   return c
 elif f(a)*f(c) < 0:
  b = c
 else:
  a = c
 return (a + b)/2
```

```
raiz = bissecao(f, 1, 2)
print("Raiz:", raiz)
```

Usando fsolve do SciPy

- `fsolve` é uma função da biblioteca `scipy.optimize` para encontrar raízes de forma automática.
- Requer apenas a função e um chute inicial.

```
from scipy.optimize import fsolve

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

raiz = fsolve(f, x0=1.5)
print("Raiz:", raiz)
```

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html
- Internamente, usa uma versão modificada do método de Newton.

Exemplo Proposto

- Resolver numericamente a equação:

$$e^{-x} \cdot \sin x = \cos x$$

- Reescrevendo:

$$f(x) = e^{-x} \sin x - \cos x = 0$$

- Essa função combina oscilação e decaimento exponencial.
- Agora vamos:
  1. Visualizar a função com `matplotlib`
  2. Usar `fsolve` para encontrar uma raiz
  3. Testar diferentes chutes iniciais

Sistema com Duas Equações Não Lineares

- Considere o seguinte sistema:

$$\begin{cases} x^2 + \sin(y) - 1 = 0 \\ x \cdot y - 0{,}5 = 0 \end{cases}$$

- Sistema não-linear:
  - A primeira equação contém uma função trigonométrica ($\sin(y)$).
  - A segunda equação envolve o produto entre incógnitas.
- A solução analítica não é trivial.
- Vamos usar `fsolve` do `SciPy`.

Resolvendo com fsolve

```python
from scipy.optimize import fsolve
import numpy as np

def sistema(vars):
  x, y = vars
  eq_1 = x**2 + np.sin(y) - 1
  eq_2 = x * y - 0.5
  return [eq_1, eq_2]

raiz, info, ier, msg = fsolve(sistema, [0.5, 1], full_output=True)


print("x =", raiz[0])
print("y =", raiz[1])
```

Saída Estendida do `fsolve`

Ao usar `fsolve` com `full_output=True`, temos:

- **sol**: vetor com a solução aproximada encontrada
- **info**: dicionário com informações sobre a iteração
- **ier**: código de status da iteração
- **msg**: mensagem textual de diagnóstico

Saída do exemplo anterior:

- `sol = [0.5296, 0.8578]`
- `ier = 5`
- `msg =` *"The iteration is not making good progress..."*
- `info['fvec'] = [0.0369, -0.0456]` $\Rightarrow$ as equações não foram zeradas

Interpretando a Saída do `fsolve`

- **Código de status** `ier = 5` indica:
  - A iteração não está progredindo bem.
  - Provavelmente estamos longe da raiz ou em região mal condicionada.
- **info['fvec']** indica o valor das equações no ponto encontrado:

$$f(x, y) = \begin{bmatrix} 0.0369 \\ -0.0456 \end{bmatrix} \quad \Rightarrow \quad \text{Sistema não está satisfeito}$$

- **Soluções possíveis:**
  - Melhorar chute inicial
  - Plotar o sistema para entender o comportamento
  - Usar outro método (`scipy.optimize.root`)

Código com root e chute eficiente

```
from scipy.optimize import root
import numpy as np


# Chute inicial
x0 = [0.5, 2.7]

sol = root(sistema, x0, method='hybr',
options={'xtol': 1e-12, 'maxiter': 10000})

print("Sucesso?", sol.suçess)
print("x =", sol.x[0])
print("y =", sol.x[1])
print("f(x, y) =", sol.fun)
print("Mensagem:", sol.message)
```

Solução Encontrada com `root`

- Utilizando um novo **chute inicial**:

$$x_0 = 0.5, \quad y_0 = 2.7$$

- Configuração do solver:
  - Método: `hybr`
  - Tolerância: `xtol = 1e-12`
  - Máximo de iterações: `maxiter = 10000`
- Solução obtida:

$$x \approx 0{,}2580, \quad y \approx 1{,}9378$$

$$f(x, y) = \begin{bmatrix} 0.0 \\ 1.1 \times 10^{-16} \end{bmatrix}$$

- Mensagem do solver: *"The solution converged."*

Quer saber quais opções estão disponíveis?

- O SciPy permite listar as opções disponíveis para cada método com o comando:

```
from scipy.optimize import show_options

show_options(solver='root', method='hybr', disp=True)
```

- Substitua `method='hybr'` por outros métodos como: `'lm'`, `'broyden1'`, `'df-sane'`, etc.
- Útil para descobrir quais parâmetros podem ser passados via `options={}`.

Exemplo 1: Trocador de Calor contracorrente

Considere um trocador de calor do tipo **anular**, utilizado para resfriar óleo com água em escoamento **contracorrente**. As condições operacionais e geométricas são:

Água (fluido frio):

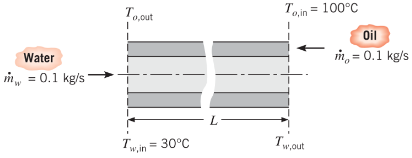
- Temperatura de entrada: $T_{c,i} = 30^\circ C$

Óleo (fluido quente):

- Temperatura de entrada: $T_{h,i} = 100^\circ C$

Trocador de calor (anular):

- Comprimento: $L = 40$ m
- Diâmetro interno do tubo externo: $D_o = 38{,}1$ mm
- Diâmetro externo do tubo interno: $D_i = 25{,}4$ mm
- Coeficiente global: $U = 55$ W/m^2·K



Properties	Water	Oil
ρ (kg/m^3)	1000	800
c_p (J/kg · K)	4200	1900
ν (m^2/s)	7×10^{-7}	1×10^{-5}
k (W/m · K)	0.64	0.134
Pr	4.7	140

Exemplo 2: Efeito da descarga na vazão do chuveiro

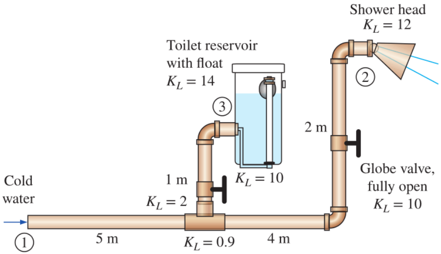
O encanamento do banheiro de um edifício é composto por tubos de cobre de diâmetro interno igual a 1,5cm conforme mostrado ao lado.

(a) Sabendo que a pressão manométrica na entrada do sistema é de 200kPa durante o banho, e que o reservatório da descarga está cheio (sem escoamento nesse ramo), determine a vazão de água que passa pelo chuveiro.

(b) Determine o efeito da descarga do vaso sanitário sobre a vazão do chuveiro.

Informações adicionais:

- Diâmetro interno das tubulações: 1,5cm
- Tubulação Cobre: $\varepsilon = 1,5\ \mu$m



https://colab.research.google.com/drive/1kzRTgymxN8VUSX8ZwBZSoXivDQhEtfnK?usp=sharing

Exemplo 2: Efeito da descarga na vazão do chuveiro

- Correlação de Churchill:

$$f = 8\left[\left(\frac{8}{\mathrm{Re}}\right)^{12} + (A+B)^{-1.5}\right]^{1/12}$$

- onde:

$$A = \left\{-2.457\ln\left[\left(\frac{7}{\mathrm{Re}}\right)^{0.9} + 0.27\frac{\varepsilon}{D}\right]\right\}^{16}$$

$$B = \left(\frac{37530}{\mathrm{Re}}\right)^{16}$$