ICMC-USP

Notas de Aula
SME0305, SME0306 & SME0602

Métodos Numéricos e Computacionais

com Aplicações para Engenharia e Progamação em `python`

Roberto F. Ausas

June 26, 2024

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

Prefacio

Este documento está organizado por capítulos, cada um dos quais apresentando os principais métodos do cálculo numérico e computacional para os cursos de engenharia. Para cada tema, se apresentam os principais conceitos teóricos por trás dos métodos numéricos, algumas aplicações de interesse para a engenharia, a sua implementação computacional e, finalmente, problemas e atividades que o aluno precisará desenvolver, com as quais, este será avaliado ao longo do curso.

Que tópicos este curso apresenta

Nas disciplinas de cálculo numérico para engenharia são estudados os seguintes temas, que temos dividido em capítulos (a depender do curso específico, alguns tópicos poderão não estar inclusos):

Capítulo |01| Introdução à programação em `python`;
Capítulo |02| Redes em `python`: Sistemas Lineares 1;
Capítulo |03| Problema de Calor e Iterações na Engenharia: Sistemas Lineares 2;
Capítulo |04| Aproximação de Autovalores e Autovetores;
Capítulo |05| Aproximação de Funções (sme0306 e sme0602);
Capítulo |05| Sistemas Sobredeterminados e Analise de Dados (sme0305);
Capítulo |06| Problemas Não Lineares (sme0306 e sme0602);
Capítulo |07| Resolução Numérica de EDOs (sme0306 e sme0602);

RFA

Contents

PROGRAMAÇÃO EM `python` 1

1.1 Preludio

Nas primeiras aulas precisamos introduzir a principal ferramenta de trabalho que será utilizada nas disciplinas de Métodos Numéricos e Computacionais. Depois de tudo, a ideia do cálculo numérico é resolver problemas complexos de Engenharia, que não poderiam ser resolvidos manualmente, sem o auxilio de um computador.

De maneira muito geral, as linguagens de programação, podem ser divididas em duas categorias:

- **Linguagens compiladas:** Dentre as primeiras temos linguagens tais como `C`, `C++` e `Fortran`. A escrita de código neste tipo de linguagens de programação requer um domínio maior da sintaxe e das funcionalidades da linguagen. Como contrapartida, este tipo de linguagens produzem códigos que são muito rápidos pois tem sido optimizadas ao longo dos anos, e por tanto são usadas amplamente na Engenharia. De fato, a maioria dos códigos de cálculo usados na indústria (*Open Source* ou comerciais) estão feitos com elas. Ao **compilar** o código e gerar um arquivo binário optimizado, é possivel tirar o maior proveito do poder de processamento de um computador.

- **Linguagens interpretadas:** Por outra parte, as linguagens interpretadas, como `python` e `Matlab`, são relativamente simples e intuitivas, pela sua flexibilidade na sintaxe, tornando o processo de desenvolvimento mais rápido e ágil. De maneira grosseira, o código vai sendo executado a medida que se **interpreta**. Porém, se não são tomados alguns recaudos no desenho do código, a performance computacional delas pode estar bastante aquém do necessário para resolver problemas de grande porte. Um exemplo prototípico disto, é quando utilizamos uma estrutura de repetição (como um `for`) no qual realizamos um grande número de operações em cada passo. Ao longo de curso iremos chamando a atenção sobre isto, tentando introduzir boas práticas de programação.

Como comparação, vejamos o exemplo de um código simples para criar um array com números de ponto flutuante de dupla precisão, popular ele com os números $0, \ldots, N$ e imprimir o resultado na tela, usando a linguagem compilada `C` (acima) e a linguagem interpretada `python` (embaixo).

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  int i, N = 10;
  double *array;
  array=(double *) malloc(N*sizeof(double));
  for(i=0; i < N; i++) {
       array[i] = (double) i;
       printf("%lf\n", array[i]);
     }
  free(array);
  return 0;
}
```

```python
import numpy as np
N = 10
array = np.arange(N)
print(array)
```

Olhando para o exemplo, já vemos que uma linguagem interpretada como `python` se torna mais prática para um curso de cálculo numérico, pois o objetivo é entender os métodos numéricos, os algoritmos e como resolver problemas práticos no computador, sem gastar muito tempo no desenvolvimento de código.

Neste curso iremos adotar a linguagem `python`, a qual tem-se tornado bastante popular nos últimos anos. Para facilitar a implementação dos diferentes métodos numéricos, contamos com algumas bibliotecas específicas que facilitarão o trabalho:

- `numpy`: Para manipulação eficiente de matrizes e vetores, operações de algebra linear computacional e vários métodos numéricos.
- `scipy`: Para métodos numéricos mais avançados ou específicos, não cobertos pela anterior.
- `matplotlib`: Para plotagens e geração de gráficos em 2D e 3D.

Aviso importante

O material de estudo para este capítulo serão os `jupyter notebooks` desenvolvidos pelo professor na sala de aula, que foram disponibilizadas no repositório da disciplina. Outras fontes de informação a ser consideradas são os sites das bibliotecas que iremos utilizar, tais como `https://numpy.org` e `https://matplotlib.org/`, assim como consultas dirigidas ao professor em forma presencial ou por e-mail (`rfausas@icmc.usp.br`).

1.2 Lista 1 de exercícios

A lista de exercícios a ser apresentada em avaliação oral por sorteio é dada na sequência:

Exercícios

1. Fazer em python uma função que:

   - Pega dois vetores randômicos **a** e **b** de dimensão n, e dois escalares randômicos α e β e calcula um vetor **c** tal que

     $$\mathbf{c} = \alpha\mathbf{a} + \beta\mathbf{b}$$

   - Pega uma matriz randômica **A** de $n \times n$ e calcula a sua m-essima potência

     $$\mathbf{A}^m = \underbrace{\mathbf{A} \ldots \mathbf{A}}_{m \text{ vezes}}$$

     (tomar valores de $m = 2, 3, 4$).

   Em todos os casos medir o tempo necessário para realizar as operações para diferentes dimensões n. Plotar o tempo de cálculo como função da dimensão n usando a escala linear padrão e a escala `loglog`. No segundo ponto, colocar no mesmo gráfico os resultados para os diferentes valores de m. Tirar conclusões.

   Nota: Para que os resultados sejam interessantes, no primeiro ponto, tomar valores de $n = 10^5,\ 10^6, \ldots, 10^9$. Já, no segundo ponto, tomar valores de $n = 1000, 2000, 3000, 4000$.

2. Considerar uma sequência de numéros gerada da seguinte forma:

   $$x_n = a\, x_{n-1}\,(1 - x_{n-1}), \quad n = 1, 2, \ldots, N$$

   Considerar $N = 5000$, $x_0 = 0.1$ e diferentes valores de a entre 0 e 4 (p.e., $a = 1,\ 2,\ 3.8$ e 4). Calcular a média e a variança da sequência:

   $$\bar{x} = \frac{1}{N}\sum_{i=0}^{N} x_i, \quad \sigma = \frac{1}{N-1}\sum_{i=0}^{N}(x_i - \bar{x})^2$$

   Programar-lo na mão e usando funções de `numpy` já prontas Comparar os resultados.

3. No problema 2, plotar a sequéncia de valores obtida em cada caso considerando os diferentes valores de a pedidos. Fazer os gráficos usando legendas, labels, e outros atributos que achar interessante, para melhor ilustrar os resultados.

4. Continuando com a sequência do problema 3, fazer um código que gera o diagrama de bifurcações, que é um gráfico que mostra os valores que assume a sequência, como função dos valores de a. O resultado deveria ser algo do tipo mostrado na figura ao lado, que no eixo horizontal tem os valores de a usados para gerar a sequência e no eixo vertical todos os possíveis valores que toma a sequência para o correspondente valor de a. Se sugere usar pontinhos bem pequenos para gerar o gráfico. Explicar os resultados se auxiliando com os gráficos

do exercício anterior.

5. Considerar uma rede de distribuição com a mostrada na figura ao lado. Esta pode ser um exemplo de uma rede elétrica ou hidráulica e é basicamente o que se chama um *grafo*). Notar que em geral ela estará caracterizada por um certo número de *nós* (ou uniões), um certo nuúmero de *arestas* e alguma informação sobre a conectividade entre pontos.

   - Fazer uma estrutura de dados que sirva para descrever essa rede. Idealmente, a estrutura deveria incluir algum tipo de matriz ou *array* que indica como os nós e as arestas estão relacionados. Adicionalmente, a estrutura deve conter um array para descrever as coordenadas (x, y) de cada nó. Construir um exemplo e plotar a rede.
   - Fazer uma função que deleta um nó e todas as arestas que emanam dele.
   - Fazer uma outra função em `python` que permita apagar (ou *deletar*) uma aresta da rede. Notar que se a aresta possui um nó que náo pertence a nenhuma outra aresta, esse nó também precisa ser deletado.
   - Finalmente, fazer uma função que insere uma nova aresta na rede para conectar dois pontos já existentes.

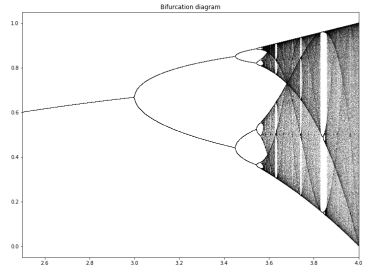


Figure 1.1: Diagrama de bifurcações.

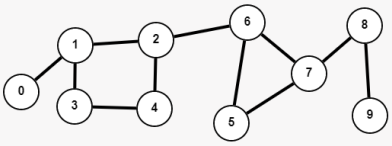


Figure 1.2: Exemplo de uma rede.

CÁLCULO DE REDES EM python: Resolução de Sistemas Lineares 1

2

2.1 Preludio

No presente capítulo iremos mostrar como se formula o problema de distribuição numa rede elétrica ou hidráulica, ou em geral, qualquer quantidade ou informação que se propaga por uma rede de maneira conservativa. De maneira geral temos os seguintes:

Objetivos

- Entender a modelagem de distintos sistemas físicos tais como redes hidráulicas, elétricas ou inclusive treliças, do ponto de vista da Algebra Linear;
- Aprender como resolver tais problemas de maneira sistemática no computador;

Há muitos sistemas físicos que podem ser modelizados como uma rede na qual se propaga uma certa quantidade física. Na tabela da sequência mostramos três exemplos clássicos que aparecem na engenharia:

Sistema	Nó	Variável nodal	Aresta	Variável de aresta	Lei de conservação
Elétrico	Conexão	Voltagem	Resistencia	Corrente	Carga elétrica
Hidráulico	União	Pressão	Cano	Vazão	Massa
Estrutural	Articulação	Deslocamento	Barra	Tensão	Momento

2.1.1 Exemplo de resolução de uma rede hidráulica

Nesta parte vamos trabalhar com redes hidráulicas. Para isto, vamos considerar o exemplo de uma rede simples mostrado na figura. Posteriormente, conseguiremos generalizar isto para resolver qualquer rede.

Para o exemplo particular temos: $n_c = 7$ canos e $n_v = 5$ nós:

O objetivo é determinar:

- As pressões em cada nó: $\mathbf{p} = [p_1 \dots p_5]^\intercal$
- As vazões em cada cano: $\mathbf{Q} = [Q_1 \dots Q_7]^\intercal$.

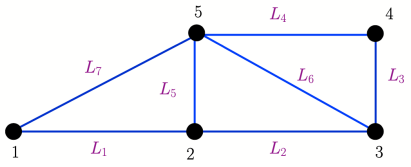


Figure 2.1: Exemplo de uma rede simples.

Conceitos necessários:

- Comportamento de um cano → **Lei constitutiva**

- Condições de acoplamento → **Conservação de massa e Continuidade**

Lei constitutiva: Comportamento de um cano

Para cada cano, $k = 1, \ldots, n_c$, que conecta os nós i e j, vamos considerar a lei de perda de pressão em função da vazão:

$$\Delta p_k = p_i^k - p_j^k = \rho_k L_k Q_k$$

$$\Rightarrow \quad Q_k = \frac{1}{\rho_k L_k} \Delta p_k \;=\; C_k \, (p_i^k - p_j^k)$$

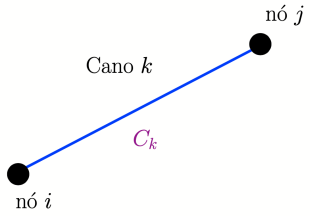


Figure 2.2: Um só cano.

- $C_k = \frac{1}{\rho_k L_k}$
- ρ_k é uma constante de resistência
- L_k é o comprimento de cada cano.
- A vazão é positiva quando vai de i a j.

Condições de acoplamento

- **Unicidade da pressão**: Para qualquer nó $i = 1, \ldots, n_v$,

$$p_i^k = p_i$$

  para todo nó k que possui o nó i.

- **Conservação da massa**:

$$\sum_{\forall\, k \text{ que possui o nó } i} Q_k{}^{(i)} \;=\; 0, \quad i = 1, 2, \ldots, n_v$$

Desta forma, teremos uma equação por cada nó da rede. Para o caso particular do exemplo:

$$\begin{aligned}
\text{Nó } 1 &: \sum_k Q_k^{(1)} = C_1(p_1 - p_2) + C_7(p_1 - p_5) = 0 \\
\text{Nó } 2 &: \sum_k Q_k^{(2)} = C_1(p_2 - p_1) + C_5(p_2 - p_5) + C_2(p_2 - p_3) = 0 \\
\text{Nó } 3 &: \sum_k Q_k^{(3)} = C_2(p_3 - p_2) + C_6(p_3 - p_5) + C_3(p_3 - p_4) = 0 \\
\text{Nó } 4 &: \sum_k Q_k^{(4)} = C_3(p_4 - p_3) + C_4(p_4 - p_5) = 0 \\
\text{Nó } 5 &: \sum_k Q_k^{(5)} = C_7(p_5 - p_1) + C_5(p_5 - p_2) + C_6(p_5 - p_3) + C_4(p_5 - p_4) = 0
\end{aligned}$$

i.e., **5 equações e 5 incógnitas**. Agora vamos ordenar o sistema:

$$
\begin{aligned}
(C_1 + C_7)\,p_1 &- C_1\,p_2 &+ 0\,p_3 &+ 0\,p_4 &- C_7\,p_5 &= 0\\
-C_1\,p_1 &+ (C_1 + C_2 + C_5)\,p_2 &- C_2\,p_3 &+ 0\,p_4 &- C_5\,p_5 &= 0\\
0\,p_1 &- C_2\,p_2 &+ (C_2 + C_3 + C_6)\,p_3 &- C_3\,p_4 &- C_6\,p_5 &= 0\\
0\,p_1 &+ 0\,p_2 &- C_3\,p_3 &+ (C_3 + C_4)\,p_4 &- C_4\,p_5 &= 0\\
-C_7\,p_1 &- C_5\,p_2 &- C_6\,p_3 &- C_4\,p_4 &+ (C_4 + C_5 + C_6 + C_7)\,p_5 &= 0
\end{aligned}
$$

ou em forma matricial:

$$
\begin{pmatrix}
C_1 + C_7 & -C_1 & 0 & 0 & -C_7\\
-C_1 & C_1 + C_2 + C_5 & -C_2 & 0 & -C_5\\
0 & -C_2 & C_2 + C_3 + C_6 & -C_3 & -C_6\\
0 & 0 & -C_3 & C_3 + C_4 & -C_4\\
-C_7 & -C_5 & -C_6 & -C_4 & C_4 + C_5 + C_6 + C_7
\end{pmatrix}
\cdot
\begin{pmatrix} p_1\\ p_2\\ p_3\\ p_4\\ p_5 \end{pmatrix}
=
\begin{pmatrix} 0\\ 0\\ 0\\ 0\\ 0 \end{pmatrix}
$$

De forma geral, a matriz da rede descreve-se como

$$
A_{ij} = \begin{cases}
\sum\limits_{k \text{ conectados com } i} C_k^{(i)} & \text{se } i = j\\
-C_r & \text{se } i \neq j, \text{ com } i \text{ conectado a } j \text{ por } C_r\\
0 & \text{se não há conexão entre } i \text{ e } j
\end{cases}
$$

- A matriz é simétrica
- A soma dos elementos de qualquer linha/coluna é zero
- A matriz **não é invertível** → **A pressão está indeterminada**. Isto significa que não podemos resolver o sistema de equações até não eliminar essa indeterminação.
- Em geral, um nó estará conectado a um número relativamente pequeno de nós. Isto fará com que a matriz A possua muitos zeros

2.1.2 Conexão da rede

Agora, precisamos conectar a rede para resolver um problema que tenha algum sentido físico.

- Para remover a indeterminação se fixa o valor da pressão em algum ponto. Para o exemplo, vamos conectar o nó 3 à atmosfera, do qual podemos deduzir que:

$$p_3 = 0$$

Porém, agora teremos uma vazão Q_R para determinar.

- Também, vamos injetar uma vazão Q_B no nó 1, conectando uma bomba, e assim resolver um problema que tenha uma solução não trivial.

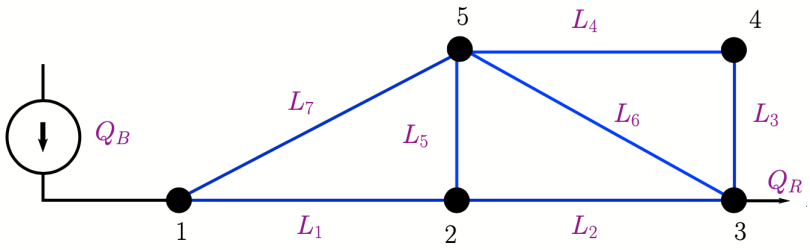


Figure 2.3: Rede conectada a uma bomba e descarregando na atmosfera.

Conexão da rede no nível algébrico

Precisamos incorporar no nível do sistema de equações a informação sobre a conexão da rede que acabamos de mencionar:

- Se estamos injetando no nó 1, então, a sua equação agora é:

$$\sum Q_r^{(1)} = (C_1 + C_7)\, p_1 - C_1\, p_2 + 0\, p_3 + 0\, p_4 - C_7\, p_5 = Q_B$$

- Similarmente, a equação para o nó 3 agora é:

$$\sum Q_r^{(3)} = 0\, p_1 - C_2\, p_2 + (C_2 + C_3 + C_6)\, p_3 - C_3\, p_4 - C_6\, p_5 = Q_R$$

em que Q_R é a **nova** incógnita. i.e., seria melhor colocar-la no lado esquerdo:

$$0\, p_1 - C_2\, p_2 + (C_2 + C_3 + C_6)\, p_3 - C_3\, p_4 - C_6\, p_5 - 1\, Q_R = 0$$

- mas, se já sabemos que $p_3 = 0$, então, temos uma equação adicional:

$$0\, p_1 + 0\, p_2 + 1\, p_3 + 0\, p_4 + 0\, p_5 = 0$$

No entanto, estamos interessados em achar as pressões, então, podemos **ignorar** a equação de balanço do nó 3 e ficar apenas com essa última equação, resultando o novo sistema de equações (ainda de 5×5):

$$\begin{pmatrix} C_1+C_7 & -C_1 & 0 & 0 & -C_7 \\ -C_1 & C_1+C_2+C_5 & -C_2 & 0 & -C_5 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -C_3 & C_3+C_4 & -C_4 \\ -C_7 & -C_5 & -C_6 & -C_4 & C_4+C_5+C_6+C_7 \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{pmatrix} = \begin{pmatrix} Q_B \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Esse sistema matriz vetor, sim pode ser resolvido.

De forma geral podemos descrever a matriz $\widetilde{A}$ e o vetor **b** do sistema como: Seja n_{atm} é o índice do nó em que a pressão é fixada ($p_{n_{atm}} = 0$) e n_B o índice do nó em que a bomba é conectada.

$$\widetilde{A}_{ij} = \begin{cases} A_{ij} & \text{se } i \neq n_{atm} \\ 0 & \text{se } i = n_{atm} \ e \ j \neq n_{atm} \\ 1 & \text{se } i = j = n_{atm} \end{cases}$$

$$b_i = \begin{cases} 0 & \text{se } i \neq n_{atm} \\ Q_B & \text{se } i = n_B \\ 0 & \text{se } i = n_{atm} \end{cases}$$

Ficando um sistema matriz-vetor do seguinte tipo:

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n_{atm}} & \dots & A_{1n_B} & \dots & A_{1n_p} \\ A_{21} & A_{22} & \dots & A_{1n_{atm}} & \dots & A_{1n_B} & \dots & A_{2n_p} \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ 0 & 0 & \dots & 1 & \dots & 0 & \dots & 0 \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ A_{n_B1} & A_{n_B2} & \dots & A_{n_Bn_{atm}} & \dots & A_{n_Bn_B} & \dots & A_{n_Bn_p} \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ . & . & \dots & . & \dots & . & \dots & . \\ A_{n_p1} & A_{n_p2} & \dots & A_{n_pn_{atm}} & \dots & A_{n_pn_B} & \dots & A_{n_pn_p} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ . \\ . \\ . \\ p_{n_{atm}} \\ . \\ . \\ . \\ p_{n_B} \\ . \\ . \\ . \\ p_{n_p} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ . \\ . \\ . \\ Q_B \\ . \\ . \\ . \\ 0 \end{pmatrix}$$

Exemplo numérico

Para colocar números concretos, consideremos uma vazão $Q_B = 3\,m^3/s$ e as seguintes propriedades para os canos da rede: Inserindo estes valores

Cano	ρ_k [bar s / m^4]	L_k [m]	C_k [bar s / m^3]$^{-1}$
1	0.1	5	2
2	0.1	5	2
3	0.1	10	1
4	0.1	5	2
5	0.1	10	1
6	0.1	5	2
7	0.1	5	2

chegamos no sistema matriz-vetor (conferir):

$$\begin{pmatrix} 4 & -2 & 0 & 0 & -2 \\ -2 & 5 & -2 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -2 \\ -2 & -1 & -2 & -2 & 7 \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

E agora o que precisamos fazer é a resolução do sistema, o qual pode ser feito pelo chamado escalonamento da matriz:

$$\left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ -2 & 5 & -2 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -2 & 0 \\ -2 & -1 & -2 & -2 & 7 & 0 \end{array}\right) \xrightarrow{\ell_2 \leftarrow \ell_2 + \frac{1}{2}\ell_1} \left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -2 & 0 \\ -2 & -1 & -2 & -2 & 7 & 0 \end{array}\right) \xrightarrow{\ell_5 \leftarrow \ell_5 + \frac{1}{2}\ell_1}$$

$$\left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -2 & 0 \\ 0 & -2 & -2 & -2 & 6 & \frac{3}{2} \end{array}\right) \xrightarrow{\ell_5 \leftarrow \ell_5 + \frac{1}{2}\ell_2} \left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -2 & 0 \\ 0 & 0 & -3 & -2 & 5 & \frac{9}{4} \end{array}\right) \xrightarrow{\ell_4 \leftarrow \ell_4 + \ell_3}$$

$$\left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & -2 & 0 \\ 0 & 0 & -3 & -2 & 5 & \frac{9}{4} \end{array}\right) \xrightarrow{\ell_5 \leftarrow \ell_5 + 3\ell_3} \left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & -2 & 0 \\ 0 & 0 & 0 & -2 & 5 & \frac{9}{4} \end{array}\right) \xrightarrow{\ell_5 \leftarrow \ell_5 + \frac{2}{3}\ell_4}$$

$$\left(\begin{array}{ccccc|c} 4 & -2 & 0 & 0 & -2 & 3 \\ 0 & 4 & -2 & 0 & -2 & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & -2 & 0 \\ 0 & 0 & 0 & 0 & \frac{11}{3} & \frac{9}{4} \end{array}\right) \xrightarrow{\ell_i \leftarrow \ell_i / a_{ii}} \left(\begin{array}{ccccc|c} 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & \frac{3}{4} \\ 0 & 1 & -\frac{1}{2} & 0 & -\frac{1}{2} & \frac{3}{8} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -\frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{27}{44} \end{array}\right)$$

$$\begin{pmatrix} 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & 1 & -\frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{pmatrix} = \begin{pmatrix} \frac{3}{4} \\ \frac{3}{8} \\ 0 \\ 0 \\ \frac{27}{44} \end{pmatrix}$$

o que finalmente resulta:

$$\begin{array}{rcrcrcrcrcl} 1\,p_1 & - & \frac{1}{2}\,p_2 & + & 0\,p_3 & + & 0\,p_4 & - & \frac{1}{2}\,p_5 & = & \frac{3}{4} \quad \rightarrow \quad p_1 = \frac{123}{88} \\ & & 1\,p_2 & - & \frac{1}{2}\,p_3 & + & 0\,p_4 & - & \frac{1}{2}\,p_5 & = & \frac{3}{8} \quad \rightarrow \quad p_2 = \frac{60}{88} \\ & & & + & 1\,p_3 & + & 0\,p_4 & + & 0\,p_5 & = & 0 \quad \rightarrow \quad p_3 = 0 \\ & & & & & + & 1\,p_4 & - & \frac{2}{3}\,p_5 & = & 0 \quad \rightarrow \quad p_4 = \frac{54}{132} \\ & & & & & & & & 1\,p_5 & = & \frac{27}{44} \quad \rightarrow \quad p_5 = \frac{27}{44} \end{array}$$

Verificação dos resultados

Lembremos que a equação para o nó 3 era:

$$\sum Q_r^{(3)} = 0\,p_1 - C_2\,p_2 + (C_2 + C_3 + C_6)\,p_3 - C_3\,p_4 - C_6\,p_5 = Q_R$$

ou, colocando os valores da tabela

$$\sum Q_r^{(3)} = 0 - 2\,p_2 + 5\,p_3 - 1\,p_4 - 2\,p_5 = Q_R$$

e inserindo as pressões que acabamos de achar:

$$Q_R = \sum Q_r^{(3)} = 0 - 2\,\tfrac{60}{88} + 0 - 1\,\tfrac{54}{132} - 2\,\tfrac{27}{44} = -3$$

ou seja, pelo nó 3 está saindo exatamente o que injectamos no nó 1.

2.2 Redes em python

A matriz de conectividades e montagem da matriz A

Uma forma "prática" de montar a matriz A, utiliza a chamada matriz de conectividades $\texttt{conec} \in \mathbb{N}^{n_c \times 2}$. Esta é uma estrutura muito conveniente para descrever a rede e é um ingrediente chave no método dos Elementos Finitos:

$$\texttt{conec} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 5 \\ 5 & 2 \\ 5 & 3 \\ 5 & 1 \end{pmatrix}$$

Olhando para a matriz **global** da rede, a ideia é ir acumulando nela, a matriz **local** associada a cada cano. Para o cano k, de conductividade C_k, teremos uma matriz de 2×2, que chamaremos a matriz local do cano.

$$\mathbf{C}_{loc_k} = \begin{pmatrix} C_k & -C_k \\ -C_k & C_k \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{Cano 1}} \begin{pmatrix} C_1 & -C_1 & 0 & 0 & 0 \\ -C_1 & C_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{Cano 2}}$$

$$\begin{pmatrix} C_1 & -C_1 & 0 & 0 & 0 \\ -C_1 & C_1 + C_2 & -C_2 & 0 & 0 \\ 0 & -C_2 & C_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{Cano 3}} \begin{pmatrix} C_1 & -C_1 & 0 & 0 & 0 \\ -C_1 & C_1 + C_2 & -C_2 & 0 & 0 \\ 0 & -C_2 & C_2 + C_3 & -C_3 & 0 \\ 0 & 0 & -C_3 & C_3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \ldots$$

$$\ldots \xrightarrow{\text{Cano 7}} \begin{pmatrix} C_1 + C_7 & -C_1 & 0 & 0 & -C_7 \\ -C_1 & C_1 + C_2 + C_5 & -C_2 & 0 & -C_5 \\ 0 & -C_2 & C_2 + C_3 + C_6 & -C_3 & -C_6 \\ 0 & 0 & -C_3 & C_3 + C_4 & -C_4 \\ -C_7 & -C_5 & -C_6 & -C_4 & C_4 + C_5 + C_6 + C_7 \end{pmatrix}$$

Para fazer tal montagem precisaremos programar uma função de `python` que automatize esse processo de montagem de matrizes locais, o que deveria ser algo do tipo:

```
def Assembly(conec, C):
    nv = ...  # numero de nos
    nc = ...  # numero de canos
    A = np.zeros(shape=(nv,nv))
    for k in range(nc):
        n1 = conec[k,0]
        n2 = conec[k,1]
        .
        .

    return A
```

em que `C` é o vetor que possui os valores das condutâncias dos canos da rede. Os detalhes ficarão como exercício para a lista 2.

Montagem do sistema final e Resolução

Uma vez que temos a matriz A do sistema, precisamos montar a matriz $\tilde{A}$ e o vetor de lado direito que é forma o sistema de equações definitivo que precisa ser resolvido para determinar as pressões nos nós da rede, que é o que nos interessa em concreto. Para isto, precisaremos programar uma função de `python` que modifique a linha da matriz na posição `natm` daquele nó que foi conectado à atmosfera e também cria um vetor de lado direito que incorpora a informação da vazão injetada pela bomba `QB` na posição `nB`. A função deveria ser algo do tipo[1]

1: A função `np.linalg.solve` essencialmente faz o escalonamento da matriz para encontrar a solução.

```
def SolveNetwork(conec, C, natm, nB, QB):
    Atilde = Assembly( ... )
    Atilde[natm, :] = ...
    .
    .
    b = np.zeros( ... )
    .
    pressure = np.linalg.solve(Atilde, ... )
    return pressure
```

Os detalhes ficarão como exercício para a lista 2.

Post-processo da solução

Uma vez resolvido o problema, estamos interessados em analizar a solução, para extrair informações de interesse do cálculo:

- Cálculo das vazões nos canos;
- Cálculo da potência consumida pela bomba;
- Visualização e plotagem da solução.

Para o primeiro ponto, uma forma elegante de fazer o cálculo é usar a seguinte fórmula:

$$\mathbf{Q} = \mathbf{K}\,\mathbf{D}\,\mathbf{p}$$

em que $\mathbf{K} \in \mathbb{R}^{n_c \times n_c}$ é a matriz diagonal com as conductâncias dos canos definida por

$$\mathbf{K}_{ij} = \begin{cases} C_i & \text{se } i = j \\ 0 & \text{no resto} \end{cases}$$

e a matriz $\mathbf{D} \in \mathbb{R}^{n_c \times n_v}$ é a matriz definida por:

$$\mathbf{D}_{kj} = \begin{cases} 1 & \text{se } j = \texttt{conec[k,0]} \\ -1 & \text{se } j = \texttt{conec[k,1]} \\ 0 & \text{no resto} \end{cases}$$

Para o segundo ponto, pode-se provar que a potência consumida pela bomba para fazer circular o fluido é dada por:

$$W = \mathbf{p}^\intercal \ (\mathbf{D}^\intercal \, \mathbf{K} \, \mathbf{D}) \ \mathbf{p}$$

isto surge do fato de que a potência consumida pela bomba tem que ser igual a energia dissipada nos canos da rede, i.e.,

$$W = \sum_{k=1}^{n_c} Q_k \Delta p_k$$

em que Q_k é a vazão pelo cano k e Δp_k é a diferença de pressão nos extremos do cano k. Mas, se colocarmos as vazões e as diferenças de pressão em vetores, podemos escrever

$$W = \mathbf{Q} \cdot \mathbf{\Delta} p = \mathbf{Q}^\intercal \, \mathbf{\Delta} p$$

i.e., W é o igual ao produto escalar desses vetores. Agora, notemos que $\mathbf{\Delta} p = \mathbf{D}\, p$ e $\mathbf{Q} = \mathbf{K}\,\mathbf{D}\,\mathbf{p}$, então

$$W = \mathbf{Q}^\intercal \, \mathbf{\Delta} p = (\mathbf{K}\,\mathbf{D}\,\mathbf{p})^\intercal \, \mathbf{\Delta} p = (\mathbf{p}^\intercal \mathbf{D}^\intercal \mathbf{K}^\intercal)(\mathbf{D}\,\mathbf{p}) = \mathbf{p}^\intercal (\mathbf{D}^\intercal \mathbf{K} \mathbf{D})\, \mathbf{p}$$

pois o produto de matrizes é **associativo** e $\mathbf{K} = \mathbf{K}^\intercal$, por ser ela uma matriz diagonal[2].

Finalmente, para visualizar a solução, na lista de exercícios iremos disponibilizar algumas funções de `python` que permitirão mostrar a distribuição de pressões em certo tipo de redes como a mostrada na figura.

2: Notar que o resultado é um número, i.e., uma matriz de 1×1, pois a potência que tem unidades de Energia por unidade de tempo (p.e., [J/s]) é de fato uma quantidade escalar.

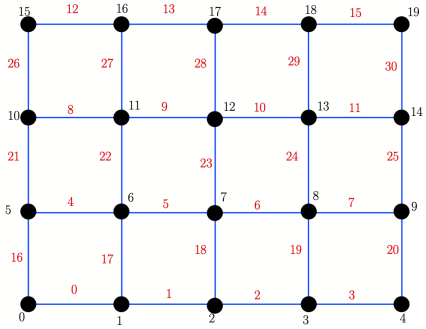


Figure 2.4: Exemplo de uma rede de tipo grade possuido $m = 5$ nós na horizontal e $n = 4$ nós na vertical (i.e., $n_v = m\,n = 20$ nós). Notar que a numeração de nós e arestas já é feita começando em 0.

Aviso importante

O material de estudo para este capítulo serão as notas apresentadas no presente capítulo, os `jupyter notebooks` disponibilizados pelo professor, assim como consultas dirigidas ao professor em forma presencial ou por e-mail (`rfausas@icmc.usp.br`).

2.3 Lista 2 de exercícios

A lista de exercícios a ser apresentada em avaliação oral por sorteio é dada na sequência:

Exercícios

1. Completar a função de `python` que faz a montagem da matriz A. Testar na rede que tem sido usada como exemplo.
2. Completar a função de `python` que cria a matriz $\tilde{A}$ e vetor de lado direito do sistema b de equações definitivo a ser resolvido, resolve o sistema e retorna o vetor de pressões com a solução. Testar na rede que tem sido usada como exemplo. Conferir a resposta com a obtida anteriormente.

3. Fazer uma função de `python` que calcula o vetor de vazões na rede segundo o explicado anteriormente. Explicar como é que o cálculo das matrizes **K** e **D** leva a determinar as vazões.

4. Incluir o cálculo da potência na função que foi programada no exercício anterior.

5. Considerar a rede hidráulica de tipo grade gerada com a função que foi disponibilizada no `jupyter notebook`. Montar um exemplo de resolução no qual a rede possui 10×12 nós, uma vazão $Q_B = 3\,m^3/s$ é injetada no nó do canto inferior esquerdo e a descarga à atmosfera é feita pelo nó do canto superior direito. Aplique todas as funções desenvolvidas até agora e visualize a solução usando a função `PlotPressure` disponibilizada no `jupyter notebook`. Mudar o ponto de descarga e comparar com a solução anterior.

6. Considerar redes hidráulicas de tipo grade de tamanho variável $m \times n$ (p.e., 10×10, 20×20, 40×40, 80×80, etc.) Fazer um código que resolve o sistema para encontrar a solução e mede o tempo de cálculo envolvido. Plotar em escala loglog o tempo como função do número de incógnitas $n_v = m\,n$.

2.4 Cálculos Monte Carlo em Redes

Objetivos

- Realizar cálculos estocásticos para estimar a probabilidade de certos eventos em sistemas complexos, tais como uma rede hidráulica composta de centenas o milhares de arestas.
- Estes cálculos estocásticos são a base dos chamados métodos Monte Carlo, que são uma classe de métodos computacionais baseados em amostragens aleatórias.
- Estes métodos são muitos poderosos e flexíveis para resolver problemas complexos. Muitas vezes são a única forma de resolver-los.

Vejamos um exemplo de um problema complexo que poderiamos resolver usando estes métodos:

Análise de risco

- Um bairro conta com uma rede hidráulica conhecida (ver figura Figure 2.5), alimentada por uma bomba que injeta uma vazão $Q_B = 10\ \mathrm{m}^3/\mathrm{s}$ no nó do canto inferior esquerdo, i.e., o nó `nB = 0` das redes que geramos usando a função `GeraRede(...)`
- No nó `natm = nv - 1` é fixada uma pressão nula.
- A rede possui dois tipos de canos: **grossos**, com "condutância" $C = 20$, e os **finos**, com $C = 2$.
- Anualmente, os canos **finos** (apenas eles) ficam obstruidos com probabilidade p_O. Quando obstruidos, seu valor de "condutância" se reduz a `Centup=0.2`, ou seja, 10 vezes menor do que o valor

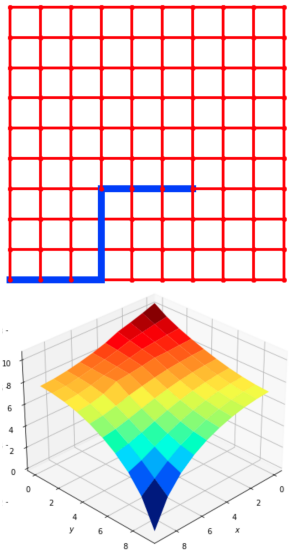


Figure 2.5: Rede nominal que será usada para realizar a analise de risco.

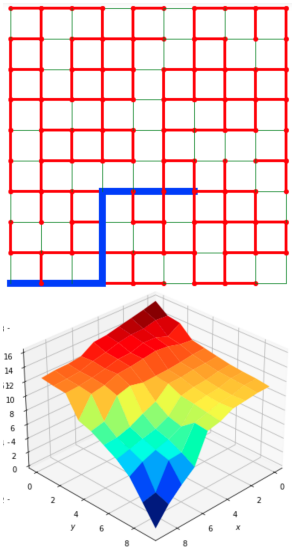


Figure 2.6: Exemplo de uma rede com canos entupido que será usada para realizar a analise de risco.

nominal (ver exemplo de uma rede com obtruções na figura Figure 2.6).

Pergunta

Qual é a probabilidade de que, passado um ano, a pressão máxima na rede seja maior do que 12?

Mas, vejamos que acontece neste problema:

- Seja M o número de canos finos, todos distintos por ocuparem lugares distintos na rede.
- Existem 2^M casos diferentes. Então, supondo, por exemplo, $M = 100$ teremos $2^{100} = 1.27 \times 10^{30}$ casos.
- Cada caso que deve ser analisado corresponde a um vetor de condutâncias da rede diferente.
- Para cada caso, podemos calcular deterministicamente todas as pressões, em particular a máxima:

```
Cnew = RandomFailFinos(C, p0, Centup)
pressure = SolveNetwork(conec, Cnew, nB, QB, natm)
pmax = np.max(pressure)
```

  em que a função `RandomFailFinos` gera uma instancia de possíveis canos entupidos na rede.
- Então, podemos avaliar a probabilidade como:

$$\text{Prob} = \frac{\#\text{ num. de casos que aconteceu o evento}}{\#\text{ num. total de possíveis casos}}$$

- Se formos calcular todos os possíveis casos surge uma **dificuldade:**

Aviso importante

Se o cálculo de um caso particular demora **1 nanosegundo**, para analisar todos demoramos $\mathbf{4 \times 10^{13}}$ **anos**.

Para entender a ideia por trás dos métodos Monte Carlo (MC), e como que eles podem nos auxiliar para responder a pergunta anterior, vamos fazer um exemplo simples:

Cálculo do número π

Vamos estimar el valor do número π da seguinte forma: Se consideramos a região quadrada $[-1, 1] \times [-1, 1]$ e um círculo inscrito, a razão das áreas entre estes é $\pi/4$ (ver figura).

Então, podemos propor um cálculo estocástico que consistirá em jogar pontos dentro dessa região quadrada. Alguns desses pontos irão cair dentro do circulo e outros fora. Como é de se esperar, a razão entre a quantidade de pontos que cai dentro e a quantidade total de pontos jogados, deveria tender justamente a razão entre as áreas de círculo e do quadrado ($\pi/4$).

Se denotarmos por N_c e N ao número de pontos dentro do círculo e ao número total de pontos, respectivamente:

$$\text{Prob} = \frac{\pi}{4} = \lim_{N\to\infty} \frac{\#\text{num. pontos no círculo}}{\#\text{num. total de pontos}} = \lim_{N\to\infty} \frac{N_c}{N}$$

Isto leva ao seguinte roteiro que precisamos executar:

1. Gerar (x_i, y_i), formado por dois números x_i e y_i entre -1 e 1, independentes e com probabilidade uniforme.
2. Calcular θ_i, definida como igual a 1 se $x_i^2 + y_i^2 < 1$ e igual a 0 se não. Seja $\Theta = \{\theta_i\}$ a sequência gerada.
3. Tirar a média
$$\hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} \theta_i$$

o qual se materializa no seguinte código de python:

```
# Computation of pi by stocastic method

N = 10000 # Number of realizations
Nc = 0
for i in range(N):
 x = -1.0 + 2.0*np.random.rand()
 y = -1.0 + 2.0*np.random.rand()
 if (x**2 + y**2 < 1.0):
  Nc = Nc + 1;

Prob = Nc / N
print('The estimate for pi is:', 4*Prob)
```

Os resultados para diferente número de realizações se mostram na figura ao lado e na tabela seguinte para $4 \times p$ (o que deveria tender a π)[3]:

#	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$
1	3.0800	3.1492	3.1450	3.1418	3.1405
2	3.2320	3.1340	3.1396	3.1400	−−
3	3.1240	3.1412	3.1469	3.1376	−−
4	3.0800	3.1660	3.1488	3.1406	−−

Agora que entedemos como funciona um método MC vamos aplicar ele para fazer a analise de risco em redes hidráulicas, que é o tema desenvolvido na Atividade 1.

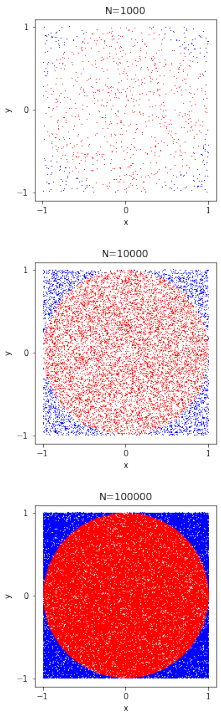


Figure 2.7: Exemplo do cálculo MC.

3: Um dos pontos a destacar é que estamos usando uma distribuição uniforme de probabilidades (`np.random.rand()`) para gerar os pontos randômicos, i.e., a probabilidade é a mesma para qualquer número em $(0, 1)$ e os pontos gerados são "independentes" um dos outros. Isto é um ingrediente essencial do método MC.

2.5 Atividade 1

A primeira atividade a ser desenvolvida em grupo e com relatório que será entregue em data a ser definida, segue na sequência:

Exercícios

1. Fazer uma função que pega as condutâncias da rede e gera uma lista com as novas condutâncias dos valores individuais dos canos que foram entupidos numa determinada realização.

Considerar uma estrutura do tipo:

```
def RandomFailFinos(C, pO, Centup):
    Cnew = np.copy(C)
    nc = ...
    for k in range(nc):
        x = np.random.rand()
        if( ..... ):
            Cnew[k] = ...

    return Cnew
```

2. Realizar o cálculo de probabilidade Prob de que, passado um ano, a pressão máxima na rede seja maior do que 12?
   Para isto:
   - Seguir um roteiro de cálculo similar ao que foi feito no exemplo de cálculo do número π;
   - Realizar entre 5000 e 10000 realizações para cada caso;
   - Considerar diferentes valores de p_O e fazer uma tabela mostrando Prob como função de p_O;
   - Plotar a evolução dessa probabilidade como função do número de realizações;
   - Tirar conclusões.

2.5.1 Exercícios opcionais

Na sequência tem um exercício extra que é opcional. Quem quiser, pode apresentar ele para o professor de maneira individual, valendo como uma das avaliações orais.

BONUS EXTRA

1. Considerar uma rede com os seguintes parâmetros:
   - `n = 8`
   - `m = 9`
   - `QB = 3`
   - `natm = n*m - 1`
   - `nB = 0`
   - As condutâncias dependem de um parâmetro x. Considerar os casos:

$$\begin{aligned} CH &= 2.3 + 0.1\,(x-1)^2, \\ CV &= 1.8 + 0.2\,(x-1)^2 \end{aligned}$$

   e

$$\begin{aligned} CH &= 2.3 + 10\,e^{-(x-5)^2}, \\ CV &= 1.8 + 10\,e^{-(x-5)^2} \end{aligned}$$

   A ideia é plotar em cada caso, a potência dissipada como função do parâmetro x e determinar visualmente para que valor de x a potencia é igual a 6. Barrer valores de x num intervalo

interessante. Aplicar as funções desenvolvidas nos exercícios anteriores.

PROBLEMA DE CALOR e ITERAÇÕES NA ENGENHARIA: Resolução de Sistemas Lineares 2

3

3.1 Preludio

O conceito de processo iterativo é muito frequênte na engenharia para resolver problemas complexos, nos quais não é possível obter uma solução fechada dos problemas numa primeira tentativa. Ele é usado, tanto nas etapas de desenho como de cálculo via modelos físicos. Neste capítulo iremos estudar o problema de transferência de calor por condução em materiais. Para isto,vamos introduzir métodos e algoritmos para resolver o problema de encontrar a distribuição de temperaturas num material, tal como é mostrado na figura ao lado. Neste capítulo temos os seguintes:

Objetivos

- Entender a modelagem do problema de transferência de calor por condução
- Apresentar métodos iterativos **sem matriz** e **com matriz**;
- Implementar os cálculos em python;
- Usar bibliotecas de métodos iterativos disponíveis em `scipy`

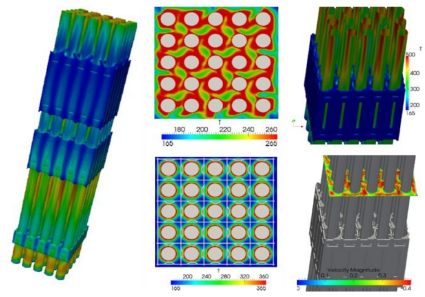


Figure 3.1: Exemplo da distribuição de temperaturas num elemento combustível de uma central nuclear sendo refrigerado por um fluido.

3.2 Transferência de Calor por Condução

Como no problema das redes hidráulicas precisamos introduzir alguns conceitos para poder formular o problema:

- **Lei constitutiva:** a qual relaciona o fluxo de energia, na forma de calor, neste caso devido a um variação ou diferença de temperatura:

$$\mathbf{q}(x,y,t) = -k\,\nabla T(x,y,t)$$

em que (x,y,t) denota o ponto espacial e o tempo t, k é a chamada condutividade térmica, que é um propriedade do material, que assume-se conhecida, T é a distribuição de temperaturas, a qual pretendemos encontrar e o operador ∇ denota o gradiente, que no caso bidimensional é[1]:

$$\nabla T(x,y,t) = \begin{bmatrix} \frac{\partial T}{\partial x} \\ \\ \frac{\partial T}{\partial y} \end{bmatrix}$$

- **Lei de balanço de energia:**

$$\underbrace{\frac{d}{dt}\iiint_V \rho\, c\, T\, dv}_{\text{Taxa de variação da energia interna}} = \underbrace{\oiint_S \mathbf{q}\cdot\mathbf{n}\, ds}_{\text{Calor que ingresa ou sai pela superficie}} + \underbrace{\iiint_V f\, dv}_{\text{Calor gerado internamente no volume}}$$

1: Na prática, o gradiente da função será aproximado por uma expresão do tipo:

$$\nabla T(x,y,t) \approx \frac{T_B - T_A}{|d_{AB}|}\,\frac{\vec{d_{AB}}}{|d_{AB}|}$$

, i.e., a partir da diferença de temperatura entre dois pontos próximos A e B, sendo d_{AB} a distância entre eles.

em que c é a capacidade calorífica do material, ρ é a densidade, f é a fonte geradora de calor (p.e., uma reação nuclear, uma reação química, uma corrente elétrica) e $\mathbf{n}$ é o vetor normal unitário **exterior** à superfície do sólido.

Neste capítulo iremos supor que o problema é estacionario, i.e., a temperatura não mudará com o tempo, e por tanto podemos negligenciar o termo do lado esquerdo, resultando no seguinte problema matemático que precisamos resolver para encontrar a distribuição de temperaturas $T(x, y)$[2]:

$$\oiint_S k \nabla T \cdot \mathbf{n} \, ds = \iiint_V f \, dv \qquad (3.1)$$

2: A equação 3.1 deve valer para qualquer volume arbitrários V limitado por um contorno ou superficie S.

Para termos uma ideia, o coeficiente de condutividade térmica, denotado por k é uma propriedade que pode ter valores bem diferentes dependend do material:

Material	Condutividade k [$\mathrm{W \cdot m^{-1} \cdot K^{-1}}$]
Aluminio	230
Aço	20
Concreto	1.5
Plástico isolante	0.03
Ar	0.02

Em geral, k assume-se constante, porém também pode depender da própria temperatura ou outras variáveis, mas isso torna o problema não linear e muito mais dificil de se resolver. Neste capťulo iremos nos restringir ao caso linear e estudaremos como resolver o sistema de equações que resulta de aplicar a lei de conservação introduzida acima.

Formulação do sistema de equações

Para poder resolver o problema, em geral, precisamos efetuar o que se conhece como **discretização**. Isto consiste em dividir o domínio computacional em parcelas que são chamadas **células**[3]. Isto é ilustrado na figura 3.2, onde um domínio computacional quadrado (figura à esquerda) e a sua discretização em células quadradas (figura à direita) são mostrados. Na figura também indicamos as temperaturas que iremos impor nas bordas do domínio (no topo T_T, embaixo T_B, na esquerda T_L e na direita T_R), as quais devem ser conhecidas para poder efetivamente resolver o problema.

3: Asumiremos que todas as células são quadradas de lado h, então, se $h \to 0$, quanto maior seja o número de celulas, mais acurada será a solução numérica obtida.

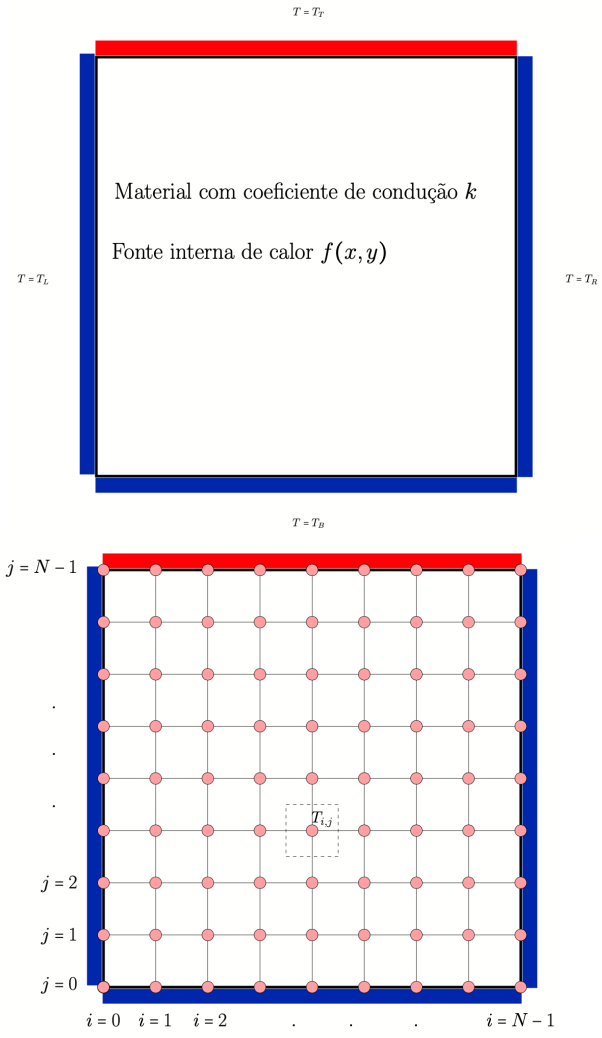


Figure 3.2: Volume e partição em células onde será resolvido o problema de condução de calor.

Olhando para a figura, vemos que temos uma grade quadrada, cujos vértices, indicados pelos pontos rosas, têm sido indexados pelos índices (i, j). Ao redor de cada ponto rosa temos desenhado com linha tracejada, um volume[4] V arbitrário de forma quadrada. A ideia é encontrar as temperaturas $T_{i,j}$ $\forall i, j$.

Para formular o problema, temos que escrever a lei de balanço (3.1) para cada possível volume da discretização. Para isto considerar a figura Figure 3.3. Podemos calcular facilmente o fluxo de calor passando por cada face do contorno do quadrado. Lembrando que a h é a distância entre pontos, temos que uma boa aproximação dos fluxos é dada por:

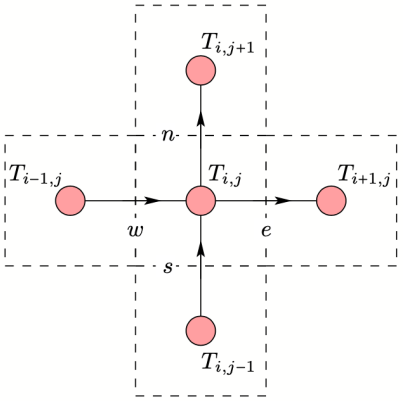


Figure 3.3: Célula (i, j) e seus 4 vizinhos, intercambiando calor a través da sua supeficies, limitada pelos contornos n (*north*), s (*south*), w (*west*) e e (*east*).

$$
\begin{aligned}
\mathbf{q}_e &\approx -k\,\frac{T_{i+1,j} - T_{i,j}}{h}\mathbf{e}_x \\
\mathbf{q}_w &\approx -k\,\frac{T_{i,j} - T_{i-1,j}}{h}\mathbf{e}_x \\
\mathbf{q}_n &\approx -k\,\frac{T_{i,j+1} - T_{i,j}}{h}\mathbf{e}_y \\
\mathbf{q}_s &\approx -k\,\frac{T_{i,j} - T_{i,j-1}}{h}\mathbf{e}_y
\end{aligned}
$$

Agora, somando todos os fluxos e calculando o termo de geração de calor:

$$\oiint_S k\,\nabla T \cdot \mathbf{n}\,ds \approx \sum_{\substack{\text{Faces do}\\ \text{contorno de } V}} \mathbf{q}_f \cdot \mathbf{n}_f\, s_f \approx$$

$$\approx h\left[\frac{k}{h}(T_{i,j} - T_{i+1,j}) + \frac{k}{h}(T_{i,j} - T_{i-1,j}) + \frac{k}{h}(T_{i,j} - T_{i,j+1}) + \frac{k}{h}(T_{i,j} - T_{i,j-1})\right] = \iiint_V f\,dv \approx h^2 f$$

Observação

Para o caso em que $f = 0$ e o k é o mesmo em todos os ponto, a distribuição interna de temperaturas é apenas uma consequência das temperaturas que estivermos impondo nas bordas. Ainda, nesse caso, sendo que lado direito da equação fica igual a zero, teremos que:

$$-T_{i+1,j} - T_{i-1,j} + 4\,T_{i,j} - T_{i,j+1} - T_{i,j-1} = 0$$

O que significa que a temperatura num ponto é a média das temperaturas nos seus vizinhos[5]:

$$T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}}{4}$$

Mas, não conhecemos as temperaturas! Isso é justamente o que queremos calcular.

5: Notar que teremos uma destas equações para cada possível valor de i, j, ou seja, o que temos é de fato um sistema de equações, para determinar as temperaturas $T_{i,j},\ i, j = 1, \ldots, N$.

3.3 Métodos iterativos

A fórmula anterior motiva o seguinte método iterativo para encontrar as temperaturas para todos os pontos no interior do material:

Método iterativo de Jacobi

O método consiste em atualizar a temperatura em todos os pontos internos, com base na temperatura calculada numa iteração anterior, partindo de algum chute inicial arbitrário. O pseudo-código fica:

- Dado um chute inicial $T_{i,j}^{(0)},\ \ i, j = 1, \ldots, N-2,$
- Para $k = 1, \ldots, MAX\_IT$

$$T_{i,j}^{(k)} = \frac{T_{i+1,j}^{(k-1)} + T_{i-1,j}^{(k-1)} + T_{i,j+1}^{(k-1)} + T_{i,j-1}^{(k-1)}}{4}$$

A implementação resulta:

```
# Metodo de Jacobi

N = 11  # Numero de pontos em cada direcao

Told = np.zeros(shape=(N,N))
Tnew = np.zeros(shape=(N,N))

# Temperaturas nas bordas
Told[0,  :] = 0.0 # TL
Told[N-1,:] = 0.0 # TR
Told[:,  0] = 0.0 # TB
Told[:,N-1] = 20.0 # TT

# Loop de iteracoes
Nmax = 10000
Tnew = Told.copy()
for iter in range(Nmax):
    Tnew[1:N-1,1:N-1] = 0.25*(Told[2:N,1:N-1]+Told[0:N-2,1:N-1] +
    \
                              Told[1:N-1,2:N]+Told[1:N-1,0:N-2])

    error = np.linalg.norm(Tnew-Told)
    if(error < 1.0e-8):
        print('Converged in %d iterations\n' %(iter))
        break ;

    Told = Tnew.copy()
```

Notar que nas paredes $T_B = T_L = T_R = 0$ e na parede superior $T_T = 20$. Na figura Figure 3.4

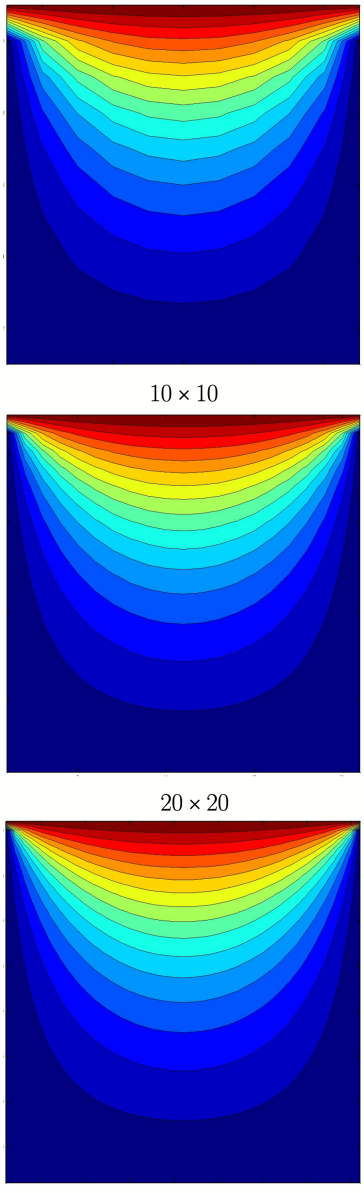


Figure 3.4: Distribuição de temperaturas para obtidas pelo método iterativo de Jacobi para diferentes níveis de discretização.

Método iterativo de Gauss-Seidel

Neste método, a diferença é que atualizamos a temperatura em todos os pontos internos, não apenas com a temperatura da iteração anterior, mas, quando possível, com a temperatura atualizada dos pontos pelos quais já temos passado, partindo de algum chute inicial arbitrário. O pseudo-código fica:

- Dado um chute inicial $T_{i,j}^{(0)}$, $i, j = 1, \ldots, N-2$,
- Para $k = 1, \ldots, MAX\_IT$

$$T_{i,j}^{(k)} = \frac{T_{i+1,j}^{(k-1)} + T_{i-1,j}^{(k)} + T_{i,j+1}^{(k-1)} + T_{i,j-1}^{(k)}}{4}$$

A implementação resulta:

```
# Metodo de Gauss-Seidel

N = 11  # Numero de pontos em cada direcao

Told = np.zeros(shape=(N,N))
Tnew = np.zeros(shape=(N,N))
```

```
# Temperaturas nas bordas
Told[0,  :] = 0.0 # TL
Told[N-1,:] = 0.0 # TR
Told[:,  0] = 0.0 # TB
Told[:,N-1] = 20.0 # TT

# Loop de iteracoes
Nmax = 10000
Tnew = Told.copy()
for iter in range(Nmax):
   for i in range(1,N-1):
       for j in range(1,N-1):
          Tnew[i,j] = 0.25*(Told[i+1,j] + Tnew[i-1,j] +\
                            Told[i,j+1] + Tnew[i,j-1])

    error = np.linalg.norm(Tnew-Told)
    if(error < 1.0e-8):
        print('Converged in %d iterations\n' %(iter))
        break ;

    Told = Tnew.copy()
```

3.4 Métodos iterativos com matriz

O que temos apresentado na seção anterior são métodos iterativos, também chamados métodos de relaxação, numa forma na qual não foi feita a montagem explícita da matriz do sistema. Este tipo de implementação **sem matriz** é possível quando o tamanho do sistema não é muito grande. Imaginemos agora que estivéssemos resolvendo um problema tridimensional, no qual temos uma grade com 100 pontos em cada direção, então nesse caso estariamos lidando com

$$100 \times 100 \times 100 = 10^6 \text{ incógnitas}$$

Neste tipo de situações, que são bem frequentes na engenharia, precisamos métodos mais sofisticados, e em geral, será necessário montar a matriz do sistema. Para isto, iremos primeiramente definir um único índice para cada ponto (ver figure 3.5).

Na numeração natural, a fórmula para construir o índice global sería: $I = i + (j-1)\,N$, porém, como em python a numeração começa em 0 e o índice I chega até $N^2 - 1$, usaremos a função de `python`

```
def ij2n (i, j, N):
      return i + j*N
```

Agora, lembremos que a equação para o nó i, j, no caso em que o coeficiente de condutividade k é constante e o termo fonte $f = 0$:

$$-T_{i+1,j} - T_{i-1,j} + 4\,T_{i,j} - T_{i,j+1} - T_{i,j-1} = 0, \quad i, j = 1, \ldots, N-2$$

O que precisamos saber é a qual índice global de temperaturas corresponde cada um dos termos na equação, i.e.,

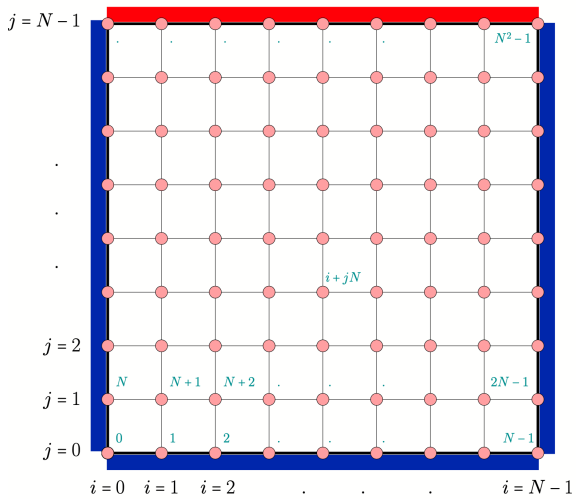


Figure 3.5: Domínio computacional indexando os pontos com um único índice.

```
Ic = ij2n(i,   j,   N)
Ie = ij2n(i+1, j,   N)
Iw = ij2n(i-1, j,   N)
In = ij2n(i,   j+1, N)
Is = ij2n(i  , j-1, N)
```

Então, a equação resulta:

$$-T_{I_e} - T_{I_w} + 4T_{I_c} - T_{I_n} - T_{I_s} = 0, \quad I_c = 0, \ldots, N^2 - 1$$

O código para montar a matriz é:

```
def MatAssembly(N):
    nunk = N*N;
    A = np.zeros(shape=(nunk,nunk))
    for i in range(1,N-1):
        for j in range(1,N-1):
            Ic = ij2n(i,   j,   N)
            Ie = ij2n(i+1, j,   N)
            Iw = ij2n(i-1, j,   N)
            In = ij2n(i,   j+1, N)
            Is = ij2n(i,   j-1, N)
            A[Ic,[Ic,Ie,Iw,In,Is]] = [4.0, -1.0, -1.0, -1.0, -1.0]
    return A
```

Por exemplo, para uma grade de 4×4 (`A = MatAssTemp(4)`) resulta:

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.]
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Todos as linhas com zeros correspondem aos pontos que estão sobre as bordas, cujas equação veremos como contruir em breve.

Temperatura nas bordas

Está faltando colocar as condições de temperatura fixa nas paredes. Para isto vamos fazer o mesmo que faziamos com as redes hidráulicas: Já que conhecemos a temperatura nos nós que estão sobre as paredes, vamos escrever uma equação trivial para eles. Por exemplo, se o nó i, j com indice global Ic, está na parede direita escreveríamos:

$$0\,T_0 \;+\; 0\,T_1 \;+\; \ldots \;+\; 1\,T_{Ic} \;+\; \ldots \;+\; 0\,T_{N^2-1} = T_R$$

Para fixar as ideias, vamos supor que a matriz do sistema é de 5×5 e que queremos impor a temperatura no ponto 3 com o valor T_R, i.e.,

$$T_2 = T_R$$

O resultado seria:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & 1 & 0 & 0 \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ T_R \\ b_3 \\ b_4 \end{pmatrix}$$

O código para montar o sistema completo de equações e junto com as condições de contorno resulta:

```
def BuildSystem(N, TL, TR, TB, TT):

    A = MatAssembly(N)
    Atilde = A.copy()

    nunk = N**2
    b = np.zeros(shape=(nunk,1)) # right-hand-side

    k = np.array(range(0,N)) # Auxiliary array
    Iden = np.identity(nunk) # Auxiliary matrix

    Ic = ij2n(0,k,N)
    Atilde[Ic,:], b[Ic] = Iden[Ic,:], TL # Tleft

    Ic = ij2n(N-1,k,N)
    Atilde[Ic,:], b[Ic] = Iden[Ic,:], TR # Tright

    Ic = ij2n(k,0,N)
    Atilde[Ic,:], b[Ic] = Iden[Ic,:], TB # Tbottom
```

```
    Ic = ij2n(k,N-1,N)
    Atilde[Ic,:], b[Ic] = Iden[Ic,:], TT # Ttop

    return Atilde, b
```

Para o mesmo exemplo de uma grade de 4×4 a matriz do sistema fica:

```
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  4. -1.  0.  0. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]]
```

Notar que basicamente, o que estamos fazendo é inserir nessas linhas, as linhas correspondentes da matriz identidade $\mathbb{I}_{N^2 \times N^2}$. Embora esta forma de colocar as condições de contorno é muito prática, ela tem uma desvantagem: **A matriz do sistema não resulta simetrica!**

Há métodos diretos, tal como o método de Cholesky[6] e métodos iterativos, tais como o método dos **gradientes** ou dos gradientes conjugados que precisam de uma matriz simétrica[7]. Neste caso, precisamos impor as condições de borda sem perder a simetria. Do ponto de vista algébrico seria (para o exemplo anterior):

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ T_R \\ b_3 \\ b_4 \end{pmatrix}$$

O que podemos fazer para preservar a simetria do sistema, é passar essa incógnita para o lado direito do sistema. Se estamos no ponto 3 por exemplo:

$$\begin{pmatrix} a_{00} & a_{01} & 0 & a_{03} & a_{04} \\ a_{10} & a_{11} & 0 & a_{13} & a_{14} \\ 0 & 0 & 1 & 0 & 0 \\ a_{30} & a_{31} & 0 & a_{33} & a_{34} \\ a_{40} & a_{41} & 0 & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} b_0 - a_{02}\, T_R \\ b_1 - a_{12}\, T_R \\ T_R \\ b_3 - a_{32}\, T_R \\ b_4 - a_{42}\, T_R \end{pmatrix}$$

6: O método de Cholesky é um tipo de método de escalonamento que serve apenas para matrizes simétricas. Neste método se calcula uma matriz triangular superior H tal que

$$A = H^\intercal H$$

Isto se conhece também como uma fatoração.

7: Também, notar que numa matriz simétrica temos a vantagem de que apenas precisaríamos armazenar a metade da matriz, o que implica uma economia de memoria, que pode ser importante para sistemas de grande porte.

3.5 Métodos iterativos gerais para sistemas de equações

Ao final das contas, lembremos que o nosso objetivo é resolver o problema:

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

Claramente, poderíamos resolver ele pelo método de escalonamento, porém, se quisermos usar um método iterativo, que pode ser mais vantajoso do ponto de vista de custo computacional ou de consumo de memória. Então, primeiramente vamos definir o que se chama residual do problema como:

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

Notar que na solução do problema verifica-se que $\mathbf{r} = 0$. Porém, num ponto qualquer $\mathbf{x}$ em geral, $\mathbf{r} = A\mathbf{x} - b \neq 0$.

Tendo definido isso, um método iterativo geral para resolver sistemas lineares basicamente fez um percurso para tentar encontrar a solução do problema partindo de um chute inicial $\mathbf{x}^{(0)}$, a cada passo do método o método determina uma direção $\mathbf{d}$ na qual se deslocar até ficar o suficientemente próximo da solução. O método segue o roteiro apresentado no pseudo-código da sequência.

Método Iterativo

Dados $\mathbf{x}^{(0)}$, TOL, MAX_IT, $k = 0$

Enquanto $\|\mathbf{r}^{(k)}\| > TOL$ e $k < MAX\_IT$

1. Resolver $M\,\mathbf{d}^{(k+1)} = -\mathbf{r}^{(k)}$
2. Determinar o escalar β_{k+1}
3. Avançar: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \beta_{k+1}\,\mathbf{d}^{(k+1)}$
4. Calcular residual: $\mathbf{r}^{(k+1)} = A\,\mathbf{x}^{(k+1)} - \mathbf{b}$
5. Incrementar k

Fim

De fato:

- Método de Jacobi, $M = \text{diag}(A)$
- Método de Gauss-Seidel $M = \text{triu(A)}$[8]

8: No método de Gauss-Seidel pode-se ver que a matriz M é a parte triangular superior da matriz que em python se extrau usando np.triu(A).

com $\beta_k = 1 \forall k$. Em métodos mais avançados resulta mais complicado identificar as matrizes e o parâmetro β_k precisa ser calculado com alguma fórmula, mas não entraremos nos detalhes para manter a simplicidade.

Também, notar que se $M = A$ então o algoritmo converge numa iteração, pois, se denotamos por $\mathbf{x}^{(\star)}$ à solução do sistema

$$d^1 = -A^{-1}\mathbf{r}^{(0)} = A^{-1}(b - A\mathbf{x}^{(0)}) = A^{-1}b - A^{-1}\,A\mathbf{x}^{(0)} = \mathbf{x}^{(\star)} - \mathbf{x}^{(0)}$$

i.e., a primeira direção será justamente o vetor que vai desde o chute inicial arbitrário $\mathbf{x}^{(0)}$ que tenhamos escolhido até a solução do sistema $\mathbf{x}^{(\star)}$.

Embora seja interessante implementar o método, em geral, se recomenda usar bibliotecas de cálculo. Dentre os métodos disponíveis em `scipy` que são bastante usados, temos:

- **Método dos Gradientes conjugados (CG):**
  `xsol_cg, info = scipy.linalg.cg(Atilde, b, tol=1e-8, M=P, callback=cg_counter())`
- **Método dos Residuos mínimos generalizados (GMRES):**
  `xsol_gmres, info = scipy.linalg.gmres(Atilde, b, tol=1e-8, M=P, callback=gmres_counter())`

Os quais podem operar com matrizes densas ou em formato esparso (ver a jupyter notebook disponibilizada para mais detalhes).

Para os métodos funcionar eficientemente, é necessário utilizar pre-condicionadores, que podem ser pensados como uma aproximação da inversa da matriz. Por este motivo, o uso deste tipo de resolutores torna-se delicado.

Aviso importante

O material de estudo para este capítulo serão as notas apresentadas no presente capítulo, os `jupyter notebooks` disponibilizados pelo professor, assim como consultas dirigidas ao professor em forma presencial ou por e-mail (`rfausas@icmc.usp.br`).

3.6 Lista 3 de exercícios

A lista de exercícios a ser apresentada em avaliação oral por sorteio é dada na sequência:

Exercícios

1. Rodar os códigos do método de Jacobi e Gauss-Seidel e fazer uma tabela para calcular o número de iterações necessárias para atingir a convergência numa tolerância de 10^{-5} e 10^{-8}. Considerando discretizações com $N = 11$, $N = 21$ e $N = 41$. Tirar conclusões.
2. Implementar o método iterativo geral que aparece no quadro laranja acima e testar ele para os casos de método de Jacobi e Gauss-Seidel. Comparar com os resultados anteriores.
3. Modificar a função que fixa as condições de contorno de forma tal que se mantenha a simetria de matriz. Testar aplicando o método iterativo geral usado anteriormente.
4. Considerar a figura que mostra um domínio computacional que possui uma inclusão de forma circular com raio $R = 0.2$. A ideia é fixar a temperatura de todos os pontos que ficam dentro do círculo a uma temperatura $T_{\text{incl}} = 20$. As paredes externas terão uma temperatura igual a 0. Modificar as funções fornecidas para incorporar esta informação no sistema. Testar com o método iterativo geral condiserando diferentes discretizações e mostrar

a solução em cada caso. Notar que para determinar quais pontos caim no circulo precisa-se calcular as suas coordenadas $(x_i, y_j) = (i\,h, j\,h)$.

5. Testar os métodos iterativos disponíveis no `scipy` para os casos do exercício anterior seguindo os exemplos mostrados no `notebook`.

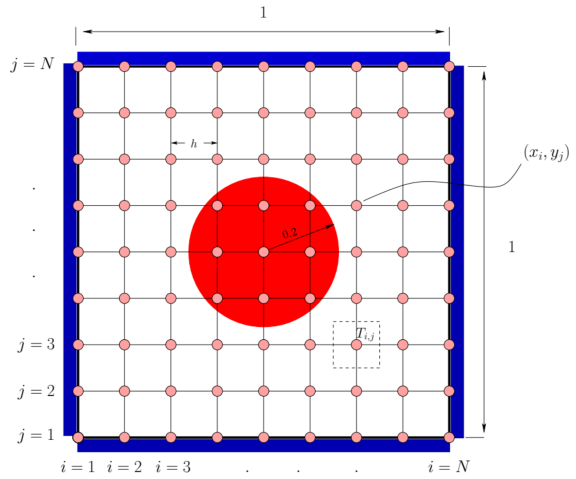


Figure 3.6: Domínio computacional com uma inclusão na qual a temperatura será fixada a um valor $T_{\text{incl}} = 20$. Todas as paredes externas terão uma temperatura de 0.

CÁLCULO NUMÉRICO DE AUTOVALORES E AUTOVETORES NA ENGENHARIA

4

4.1 Preludio

Existem vários problemas na engenharia e na matemática aplicada em que precisamos calcular os autovalores e autovetores de uma matriz. Alguns exemplos são mostrados nas figuras. Precisamos estudar o que há por traz dos métodos numéricos para calcular autovalores e autovetores de matrizes.

Objetivos

- Lembrar algumas noções de algebra linear sobre o problema de autovalores;
- Introduzir o **Método de Francis** para o cálculo do espectro completo de autovalores, o qual é baseado na fatoração QR;
- Implementar os cálculos em python e usar bibliotecas de métodos iterativos disponíveis em `scipy`;
- Entender a modelagem do problema de vibrações de membranas.

4.2 O problema de autovalores e autovetores

Seja uma matriz $A \in \mathbb{R}^{n\times n}$, um escalar $\lambda \in \mathbb{R}$ é autovalor de A se existir um $\mathbf{v} \in \mathbb{R}^n$ (não nulo*) tal que

$$A\,\mathbf{v} = \lambda \mathbf{v}$$

ou seja, procuramos aqueles $\mathbf{v} \neq 0$ tais que $A\,\mathbf{v}$ é um multiplo escalar do próprio $\mathbf{v}$. Por tanto, se λ é autovalor de A, existe $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{v} \neq 0$ tal que

$$(A - \lambda\,\mathbb{I})\,\mathbf{v} = 0$$

em que $\mathbb{I}$ é a matriz identidade de $n \times n$. De Álgebra Linear, sabemos que neste caso, o que deve estar acontecendo é que a matriz $(A - \lambda\,\mathbb{I})$ é não invertível (ou seja, singular) e por tanto ela deve ter determinante identicamente nulo, i.e.,

$$P(\lambda) = \det(A - \lambda\,\mathbb{I}) = 0$$

Este é um polinômio na variável λ, conhecido como polinômio característico.

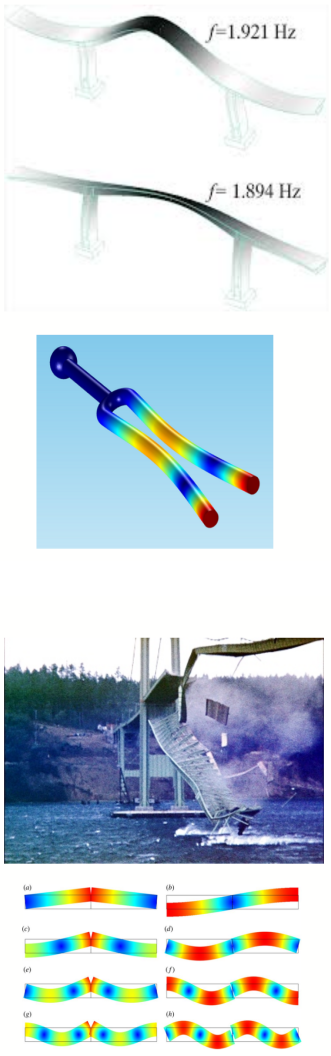


Figure 4.1: Exemplos de problemas que envolvem cálculo de autovalores.

* Notar que o caso de $\mathbf{v} = 0$ não interessa, pois a equação é satisfeita trivialmente

Exemplo 1

Considerar o seguinte exemplo:

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

Então, calculando o polinômio característico

$$P(\lambda) = \det\left(\begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 2-\lambda & 2 \\ 5 & -1-\lambda \end{bmatrix}\right)$$

Calculando o determinante, chegamos em que:

$$P(\lambda) = \lambda^2 - \lambda - 12$$

cujas raizes (i.e., os λ tais que $P(\lambda) = 0$, são $\lambda_1 = -3$ e $\lambda_2 = 4$, que são os autovalores de A.

Em particular, se a matriz A é simétrica, i.e., $A^\intercal = A$, então podemos garantir que ela possui exatamente n autovalores[1].

1: O caso das matrizes simétricas é muito importante, pois ele aparece frequentemente na engenharia.

De fato, se $\mathbf{v}$ and $\mathbf{w}$ são autovetores correspondentes a autovalores distintos λ e μ de uma matriz simétrica, i.e.

$$A\,\mathbf{v} = \lambda\,\mathbf{v}$$
$$A\,\mathbf{w} = \mu\,\mathbf{w}$$

Então, $\mathbf{v}$ e $\mathbf{w}$ são ortogonais:

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^T\,\mathbf{w} = 0$$

Exemplo 2

Considerar o seguinte exemplo:

$$A = \begin{bmatrix} 3 & 2 & 4 \\ 2 & 0 & 2 \\ 4 & 2 & 3 \end{bmatrix}$$

Então, calculando o polinômio característico

$$P(\lambda) = \det\left(\begin{bmatrix} 3 & 2 & 4 \\ 2 & 0 & 2 \\ 4 & 2 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right)$$

Chegamos em que:

$$P(\lambda) = (\lambda+1)^2\,(\lambda-8)$$

cujas raizes são $\lambda_1 = -1$, $\lambda_2 = -1$ e $\lambda_3 = 8$. Já, os autovetores correspondentes a eles são:

$$\mathbf{v}_{(1)} = [-0.4941, -0.47202, 0.73011]^\intercal$$

$$\mathbf{v}_{(2)} = [-0.55805, 0.81614, 0.14998]^\intercal,$$

$$\mathbf{v}_{(3)} = [0.6667, 0.3334, 0.6667]^\intercal,$$

onde é facil notar que os produtos escalares $\mathbf{v}_{(1)} \cdot \mathbf{v}_{(2)} = 0$ e $\mathbf{v}_{(1)} \cdot \mathbf{v}_{(3)} = 0$ e $\mathbf{v}_{(2)} \cdot \mathbf{v}_{(3)} = 0$, ou seja, são ortogonais.

Na sequência introduzimos um conceito muito importante:

Matrizes semelhantes

Duas matrizes A e B dizem-se semelhantes, se existe uma matrix Q não singular tal que:

$$A = Q\,B\,Q^{-1}$$

ou, de forma equivalente

$$B = Q^{-1}\,A\,Q$$

Então, se $(\lambda, \mathbf{v})$ é par Autovalor-Autovetor de A, $(\lambda, Q^{-1}\mathbf{v})$ será par Autovalor-Autovetor de B.

Outro conceito importante é o de matriz diagonalizável:

Matriz diagonalizável

Uma matriz diz-se diagonalizável se ela for semelhante a uma matriz diagonal, ou seja, existe Q tal que

$$A = Q\,D\,Q^{-1}$$

em que D é uma matriz diagonal, i.e.,

$$D = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

e pode-se demonstrar que se $A \in \mathbb{R}^{n\times n}$ for **simétrica**, então, existirá uma matriz Q ortogonal (i.e., $Q^{-1} = Q^T$), tal que

$$Q^T\,A\,Q = D = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

em que os $\lambda_i,\ i = 1, \ldots, n$ são os autovalores de A.

Notemos que as colunas de Q são de fato os autovetores de A: Para ver isto, tomamos um vetor da base canónica, i.e., o vetor $\mathbf{e}_{(i)}$ está feito de zeros exceto na sua compenente i que vale 1:

$$\mathbf{e}_{(i)} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \\ . \\ . \\ . \\ 0 \end{bmatrix}$$

Agora fazemos $(Q^T\, A\, Q)\, \mathbf{e}_{(i)} = D\, \mathbf{e}_{(i)} = \lambda_i\, \mathbf{e}_{(i)}$

Então:

$$(A\, Q\,)\, \mathbf{e}_{(i)} = Q\, (\lambda_i\, \mathbf{e}_{(i)}) = \lambda_i\, (Q\, \mathbf{e}_{(i)})$$

Agora chamemos $\mathbf{v}_{(i)} = Q\, \mathbf{e}_{(i)}$

$$A\, \mathbf{v}_{(i)} = \lambda_i\, \mathbf{v}_{(i)}$$

Isto significa que $\mathbf{v}_{(i)} = Q\, \mathbf{e}_{(i)}$ é o autovetor associado ao autovalor λ_i da matriz A. Mas, acontece que $Q\, \mathbf{e}_{(i)}$ é a coluna número i da matriz Q, i.e.,

$$Q = \begin{bmatrix} \uparrow & \uparrow & \ldots & \uparrow \\ \mathbf{v}_{(1)} & \mathbf{v}_{(2)} & \ldots & \mathbf{v}_{(n)} \\ \downarrow & \downarrow & \ldots & \downarrow \end{bmatrix}$$

Isto significa, que a matriz Q é feita de "pendurar" os autovetores de matriz A.

Observação importante

O visto antes, nos diz que, se conseguirmos um método para diagonalizar uma matriz, então, teremos um método para calcular **todos** os seus autovalores e autovetores.

4.3 Método de Francis

Na sequência apresentamos um método iterativo que permite diagonalizar uma matriz. O método é baseado na decomposição ou fatoração QR da matriz, para o qual temos:

Teorema da fatoração QR

Toda matriz $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ possui uma fatoração

$$A = Q\,R$$

onde $Q \in \mathbb{R}^{m\times m}$ é ortogonal e $R \in \mathbb{R}^{m\times n}$ é triangular (trapezoidal) superior, com $r_{ii} \geq 0\ \forall i$.

$$A = QR = \begin{pmatrix} q_{11} & q_{12} & \dots & \dots & q_{1m} \\ q_{21} & q_{22} & \dots & \dots & q_{2m} \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ q_{m1} & q_{m2} & \dots & \dots & q_{mm} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ 0 & r_{22} & r_{23} & \dots & r_{2n} \\ 0 & 0 & r_{33} & \dots & r_{3n} \\ 0 & 0 & . & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & 0 & \dots & r_{nn} \\ - & - & - & - & - \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Se em particular a matriz for quadrada $(m = n)$, então fica:

$$A = QR = \begin{pmatrix} q_{11} & q_{12} & \dots & \dots & q_{1n} \\ q_{21} & q_{22} & \dots & \dots & q_{2n} \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ . & . & \dots & \dots & . \\ q_{n1} & q_{n2} & \dots & \dots & q_{nn} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ 0 & r_{22} & r_{23} & \dots & r_{2n} \\ 0 & 0 & r_{33} & \dots & r_{3n} \\ 0 & 0 & 0 & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & . & \dots & . \\ 0 & 0 & 0 & \dots & r_{nn} \end{pmatrix}$$

Não iremos demonstrar o teorema anterior, o qual pode ser feito aplicando um processo conhecido como ortogonalização de vetores. Para o que resta, apenas precisamos saber que o teorema é valido, e portanto, que qualquer matriz poder ser escrita como o produto de uma matriz Q ortogonal e uma R triangular superior, isto é suficiente para entender como funcionar o seguinte método iterativo:

Algoritmo iterativo para cálculo de autovalores (Francis)

O processo iterativo de cálculo funciona da seguinte forma. A ideia é gerar uma sequência de matrizes $\{A_1, A_2, \ldots, A_k, \ldots\}$ tal que:

- Definir a primeira matriz da sequência $A_1 = A$
- Calcular a fatoração QR de $A_1 \Rightarrow A_1 = Q_1 R_1$
- Calcular $A_2 = R_1 Q_1$
- Calcular a fatoração QR de $A_2 \Rightarrow A_2 = Q_2 R_2$
- Calcular $A_3 = R_2 Q_2$
- .
- .
- .
- Calcular $A_{k-1} = R_{k-2} Q_{k-2}$
- Calcular a fatoração QR de $A_{k-1} \Rightarrow A_{k-1} = Q_{k-1} R_{k-1}$
- Calcular $A_k = R_{k-1} Q_{k-1}$
- .
- .

e assim por diante, até atingir um certo criterio de parada.

Pode-se provar duas coisas:

- as matrizes A e A_k, possuem os mesmos autovalores;
- a matriz A_k converge a uma matriz diagonal, na qual os autovalores ficam em evidência.

Para ver a primeira propriedade, numa determinada iteração temos que $A_{k-1} = Q_{k-1} R_{k-1} \Rightarrow R_{k-1} = Q_{k-1}^{\intercal} A_{k-1}$, e a sua vez definimos $A_k = R_{k-1} Q_{k-1}$, então

$$A_k = Q_{k-1}^{\intercal} A_{k-1} Q_{k-1}$$

Agora procedemos recursivamente:

$$\begin{aligned}
A_k &= Q_{k-1}^{\intercal} A_{k-1} Q_{k-1} \\
&= Q_{k-1}^{\intercal} (Q_{k-2}^{\intercal} A_{k-2} Q_{k-2}) Q_{k-1} \\
&= Q_{k-1}^{\intercal} (Q_{k-2}^{\intercal} (Q_{k-3}^{\intercal} A_{k-3} Q_{k-3}) Q_{k-2}) Q_{k-1} \\
&\;\;\vdots \\
&= Q_{k-1}^{\intercal} Q_{k-2}^{\intercal} Q_{k-3}^{\intercal} \cdots Q_1^{\intercal} A_1 Q_1 \ldots Q_{k-3} Q_{k-2} Q_{k-1} \\
&= (Q_1 \ldots Q_{k-3} Q_{k-2} Q_{k-1})^{\intercal} A (Q_1 \ldots Q_{k-3} Q_{k-2} Q_{k-1}) \\
&= V^{\intercal} A V
\end{aligned}$$

em que $V = Q_1 \ldots Q_{k-3} Q_{k-2} Q_{k-1}$, ou seja, as matrizes A_k e A são semelhantes à través da matriz V e por tanto possuem os mesmos autovalores, e como visto anteriormente, os autovetores serão as colunas da matriz V. Isto se traduz no seguinte algoritmo de diagonalização:

Algoritmo de Francis

Dada $A,\ MAX\_ITER,\ TOL$
Inicializar $k = 1$ e $A_1 = A, V_1 = \mathbb{I}$
Enquanto $\varepsilon > TOL$ e $k < MAX\_IT$

1. Calcular a fatoração $A_k = Q_k\, R_k$
2. Definir $A_{k+1} = R_k\, Q_k$
3. $V_{k+1} = V_k\, Q_k$
4. Calcular o erro: $\varepsilon = \max \|A_k^{ij}\|, i, j = 1, \ldots, n,\ i \neq j$
5. Incrementar k

Problema de autovalores generalizado

Uma variante do clássico problema de autovalores, é o chamado problema generalizado, o qual consiste em encontrar os vetores não nulos $\mathbf{v}$ e escalares λ tais que:

$$A\,\mathbf{v} = \lambda\, M\,\mathbf{v}$$

em que M é uma matriz simétrica definida positiva[2] Em muitos exemplos, M será uma matriz diagonal com coeficientes positivos (na diagonal). Neste caso, se verifica a ortogonalidade dos autovetore de A com respeito ao produto escalar **generalizado**, i.e.,

2: Uma matriz $M \in \mathbb{R}^{n\times n}$ diz-se definida positiva se $\mathbf{x}^\intercal M\mathbf{x} > 0\ \ \forall \mathbf{x} \neq 0$.

$$\mathbf{v}_{(i)}^\intercal\, M\, \mathbf{v}_{(j)} = \delta_{ij}$$

em que $\delta_{ij} = 0$ se $i \neq j$ e $\delta_{ij} = 1$ se $i = j$.

Cálculo de autovalores em `python`

Em `python` temos disponível dois métodos para resolver problemas de autovalores:

- **Matrizes densas**: O método baseado na fatoração QR que acabamos de apresentar, o qual disponibiliza todos os autovalores e autovetores da matriz:

  ```
  Lam, Q = scipy.linalg.eigh(A, M)
  ```

  A versão para matrizes não simétricas é `scipy.linalg.eig`.

- **Matrizes esparsas**: Um método que não temos apresentado, o qual é baseado no chamado método das potências. Este método está pensando para calcular apenas um certo número de autovalores (p.e., os n maiores ou os n menores). Isto é muitas vezes suficiente. Por exemplo, no cálculo das frequências de oscilação de estruturas ou de circuitos, em geral, só interessam os chamados modos fundamentais.

```
Lam, Q = scipy.sparse.linalg.eigsh(K, k=20, M=M, which='SM')
```

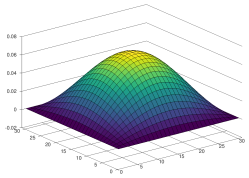
em que `k` indica o número de autovalores desejado e `'SM'` denota *Smallest in magnitude*, outras opções sendo `'LM'` (*Largest in magnitude*).
A versão para matrizes não simétricas é `scipy.sparse.linalg.eigs`.

Em ambos os casos, se a matriz `M` não é fornecida, assume-se o problema estándard de autovalores (i.e., $M = \mathbb{I}$).

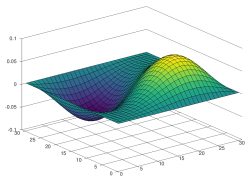
4.4 Vibração de membranas em python

Neste parte iremos estudar uma aplicação do cálculo de autovalores à problema de oscilação de membranas em tensão, isto é, um corpo quase-bidimensional que ocupa uma região do plano $x - y$ e que está submetido a uma tensão

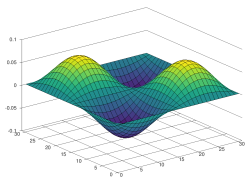
Outro exemplo é uma "membrana unidimensional" (uma corda tensa), como mostrado na figura ao lado.



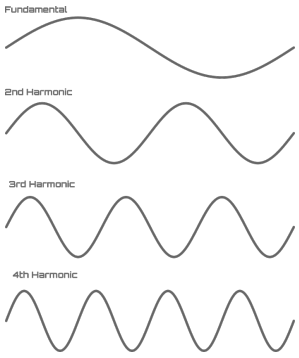
A dinâmica da membrana surge do principio de conservação do momento linear que é basicamente a equação de Newton para uma membrana representada por um dominio Ω do plano. Então, o problema de equilibrio dinâmico, pode ser formulado como uma equação a derivadas parciais em que a incógnita é o deslocamento vertical $w(x, y, t)$ no ponto $(x, y) \in \Omega$ ao tempo t



$$\begin{cases} \rho\, e\, \dfrac{\partial^2 w}{\partial t^2} - \sigma\, \nabla^2 w = f(x, y, t)\,, & (x, y) \in \Omega \subset \mathbb{R}^2 \\ w(x, y, t) = 0, & (x, y) \in \partial\Omega \\ w(x, y, 0) = u_0(x, y) & (x, y) \in \Omega \\ \dfrac{\partial w}{\partial t}(x, y, 0) = v_0(x, y) & (x, y) \in \Omega \end{cases}$$



em que conhecemos:



- $\partial\Omega$ denota a borda do dominio Ω;
- $\rho(x, y)$ é a densidade do material da membrana;
- $e(x, y)$ é a espessura da membrana;
- σ é a tensão membranal (em N/m);
- $f(x, y, t)$ é a força vertical aplicada (em N/m^2);
- ∇^2 é o operador Laplaciano (para modelizar as forças internas), o qual em duas dimensões é dado por:

$$\nabla^2 w = \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}$$

As duas últimas equações representam as condições iniciais, para a posição e a velocidade da mambrana.

Para resolver este problema, em geral, precisamos introduzir uma discretização do domínio Ω, assim como foi feito no problema de transferência de calor. Vamos considerar um domínio retangular e uma grade regular tal como mostrado na figura:

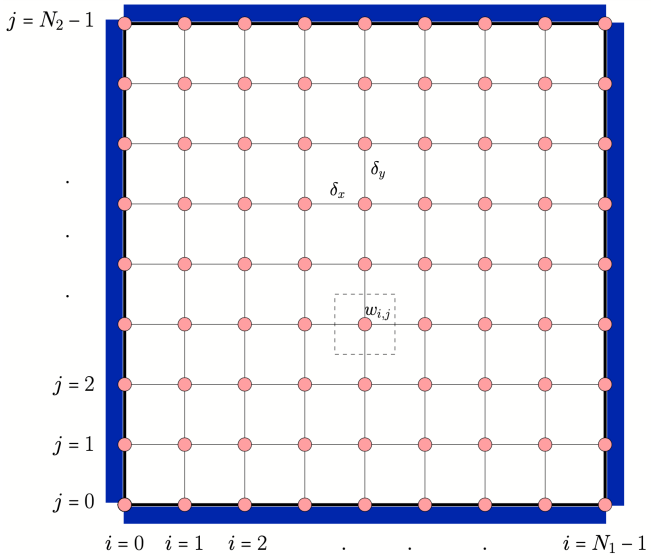


Figure 4.2: Domínio discretizado para o problema de oscilação de membranas.

A ideia é escrever a equação para cada parcela de membrana identificada pelos indices i, j

$$\rho_{ij}\, e_{ij}\, \frac{d^2 w_{ij}}{dt^2} - \sigma\, \nabla^2_{ij} w = f_{ij}(t)$$

O ponto agora é como expressar o Laplaciano no caso discreto. Uma possibilidade é dada pelo operador de diferenças de segundo grau

$$\nabla^2_{ij} w = \frac{w_{i+1,j} + w_{i-1,j} + w_{i,j+1} + w_{i,j-1} - 4w_{ij}}{\delta^2}$$

resultando na equação para o ponto i, j

$$\rho_{ij}\, e_{ij}\, \frac{d^2 w_{ij}}{dt^2} - \sigma\, \frac{w_{i+1,j} + w_{i-1,j} + w_{i,j+1} + w_{i,j-1} - 4w_{ij}}{\delta^2} = f_{ij}(t)$$

Agora transformamos os "arrays" em vetores, por exemplo:

$$W_k = w_{ij} \qquad \Leftrightarrow \qquad \texttt{k = i + j*N1}$$

onde N_1 é o número de nós em x, dando

$$\rho_k\, e_k\, \frac{d^2 w_k}{dt^2} - \sigma\, \frac{w_{k_e} + w_{k_w} + w_{k_n} + w_{k_s} - 4w_k}{\delta^2} = f_k(t)$$

em que k_e (east), k_w (west), k_n (north), k_s (south) são os indices globais dos pontos vizinhos do ponto k. Notar que os w_k ainda são funções do tempo. Em notação vetorial podemos escrever

$$M\, \frac{d^2 \mathbf{w}(t)}{dt^2} + K\mathbf{w}(t) = F(t)$$

em que M é chamada de matriz de massas e K de matriz de rigidez da membrana e vem dadas por:

$$M = \begin{bmatrix} \rho_1 e_1 & 0 & \dots & 0 & \dots & 0 \\ 0 & \rho_2 e_2 & \dots & 0 & \dots & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & \dots & \rho_k e_k & \dots & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & \dots & 0 & \dots & \rho_n e_n \end{bmatrix}$$

$$K = \frac{\sigma}{\delta^2} \begin{bmatrix} . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & -1 & \dots & -1 & 4 & -1 & \dots & -1 & . & . \\ . & . & -1 & \dots & -1 & 4 & -1 & \dots & -1 & . \\ . & . & . & -1 & \dots & -1 & 4 & -1 & \dots & -1 & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \end{bmatrix}$$

que é essencialmente a mesma matriz pentadiagonal que temos visto em outras ocasiões. A implementação em python pode ser feita como mostrado no código da sequência[3].

3: Notar que na implementação da matriz K estamos usando o comando `scipy.sparse.diags`, que já cria uma matriz pentadiagonal em formato esparso.

```
def BuildMatrizesEigen(N1, N2, sigma, rho, e, delta):
    nunk = N1*N2

    # Stiffness matrix K: Build it as a sparse matrix
    d1 = 4.0*np.ones(nunk)
    d2 = -np.ones(nunk-1)
    d3 = -np.ones(nunk-N1)
    K = (sigma/delta**2)*scipy.sparse.diags([d3, d2, d1, d2, d3],
              [-N1, -1, 0, 1, N1], format='csr')

    # Force the eigenvalues associated to boundary points
    # to be a big number as compared to fundamental modes
    big_number = 10000
    Iden = big_number*scipy.sparse.identity(nunk, format='csr')

    # Lados verticais
    for k in range(0,N2):
        Ic = ij2n(0,k,N1) # Left
        K[Ic,:], K[:,Ic] = Iden[Ic,:], Iden[:,Ic]

        Ic = ij2n(N1-1,k,N1) # Right
        K[Ic,:], K[:,Ic] = Iden[Ic,:], Iden[:,Ic]

    # Lados horizontais
    for k in range(0,N1):
        Ic = ij2n(k,0,N1) # Bottom
        K[Ic,:], K[:,Ic] = Iden[Ic,:], Iden[:,Ic]

        Ic = ij2n(k,N2-1,N1) # Top
        K[Ic,:], K[:,Ic] = Iden[Ic,:], Iden[:,Ic]

    # Mass matrix: Simple case, multiple of identity
    M = rho*e*scipy.sparse.identity(nunk, format='csr')
```

```
    return K, M
```

A questão é, onde que entram os autovalores e autovetores para resolver este problema. A resposta é dada pelo seguinte teorema:

Evolução do sistema

Considerando o caso de escilações livres $f = 0$ (sem forças aplicadas). A evolução do sistema

$$M\,\mathbf{w}'' + K\,\mathbf{w} = 0 \tag{4.1}$$

a partir de uma condição inicial arbitrária para o deslocamente $\mathbf{w}(0) = U$ e para a velocidade $\mathbf{w}'(0) = V$ é:

$$\mathbf{w}(t) = \sum_{k=1}^{n} \Phi^{(k)}\, c_k\, \sin(\omega_k t + \phi_k) \tag{4.2}$$

em que os vetores $\Phi^{(k)}$ e as frequências ω_k são solução do problema de autovalores

$$K\,\Phi^{(k)} = \omega_k^2\, M\,\Phi^{(k)}$$

- Os vetores $\Phi^{(k)}$ são os **modos naturais** da estrutura, e os ω_k são as **frequências naturais** dela.
- Os coeficientes c_k e ϕ_k, de amplitude e fase, são definidos pela condição inicial.

4.5 Lista 4 de exercícios

Exercícios

1. Implementar o método de Francis. Para isto, utilizar a função `scipy.linalg.qr` a qual retorna as matrizes da fatoração QR. Testar o método em matrizes randômicas simétricas de dimensão crescente (p.e. $n = 10, 50, 100, 200$). Medir o tempo de cálculo como função de n.

2. Pensar um método para construir uma matriz simétrica cheia de dimensão n e que possua espectro conhecido (p.e., os autovalores poderiam ser os números $1, 2, 3, \ldots, n$. Agora, usar a função `scipy.linalg.eigh` e calcular os autovalores, para conferir que tenham os valor esperado.

3. Mostrar que a 4.2 é efetivamente solução do problema de evolução 4.1.

4. No problema anterior, calcular os coeficiente c_k e ϕ_k a partir de uma condição inicial U_0 para a posição e V_0 para a velocidade.

5. Introduzir as modificações necessárias no código que gera as matrizes do problema de autovalores para resolver o problema de encontrar os modos de oscilação de uma membrana de forma

circular de raio $R = 0.5$. Considerar $\rho = e = \sigma = 1$.

6. Introduzir as modificações necessárias no código para implementar uma densidade que uma função da posição dada por:

$$\rho(x, y) = 1 + 0.75\cos(4\pi x)\cos(2\pi y)$$

Considerar uma membrana retangular de lado $L_1 = 1$ e lado $L_2 = 0.5$. A espesura pode tomar sendo $e = 1$.

4.6 Atividade 2

A segunda atividade a ser desenvolvida em grupo e com relatório que será entregue em data a ser definida, segue na sequência:

Exercícios

1. Considerar uma membrana quadrada de lados $L_1 = L_2 = 1$, com espesura $e = 1$, densidade $\rho = 1$ e tensão $\sigma = 1$. Calcular as primeiras 4 frequências de oscilação da membrana como função do tamanho da grade. Tomar $N_1 = N_2 = 11, 21, 31, 41, 51, 61, 81, 101$.
   Usar a função `scipy.sparse.linalg.eigsh`, especificando $k = 4$. Mostrar os resultados em forma de gráfico e tabela.

2. Repetir o exercício anterior, mas agora para uma membrana de forma triangular.

APROXIMAÇÕES NUMÉRICAS NA ENGENHARIA 5

5.1 Preludio

Neste capítulo lidamos com diferentes métodos de aproximação de dados que aparecem frequentemente na engenharia e na matemática aplicada, e que servem para tentar fazer sentido ou representar dados discretos originados da realização de experimentos/medições. Ao falar de dados vindos de experimentos, não necessariamente estamos nos referindo a experimentos físicos. Os dados que iremos analisar poderiam vir de experimentos numéricos, i.e., dados produzidos por software ou códigos de cálculo que já estão disponíveis. Os métodos de aproximação servem para poder poder extrair informações úteis destes dados. A teoria por trás deles é relativamente simples e pode ser consultada em qualquer livro de Cálculo Numérico. Por este motivo, não iremos dar muitos detalhes no presente texto. A ideia é introduzir alguns métodos clássicos e mostrar a sua utilização em `python`. Em particular serão apresentados os tópicos na sequência:

Objetivos

- Interpolação de dados e funções;
- Aproximações de quadrados mínimos para dados e funções
- Cálculo numérico de integrais definidas;
- Aproximações para as derivadas de funções

5.2 Interpolação de Dados e Funções

O conceito de interpolação de dados é muito simples. Consideremos um conjunto de pontos que foram obtidos de algum experimento (físico ou sintético)

$$\{(x_i, y_i)\}_{i=0}^{n}$$

em que os $\{x_i\}_{i=0}^{n}$ são chamados de abscissas. e os $\{y_i\}_{i=0}^{n}$ são as ordenadas, ou em definitiva, os dados medidos. O objetivo do problema de interpolação é construir algum tipo de função $v(x)$ que seja simples o suficiente e que sirva para aproximar os dados, respeitando que:

$$v(x_i) = y_i, \quad i = 0, 1, \ldots, n$$

Então, dizemos que a função $v(x)$ **interpola** os dados. A ideia é ilustrada na figura 5.1.

Notar que o interessante disto é que $v(x)$ não apenas fornece o valor da função nos pontos x_i originais, senão, que também podemos avaliar ela em qualquer ponto intermediário $x \neq x_i$ no qual não temos informação disponível vinda do experimento que deu origem a esses dados.

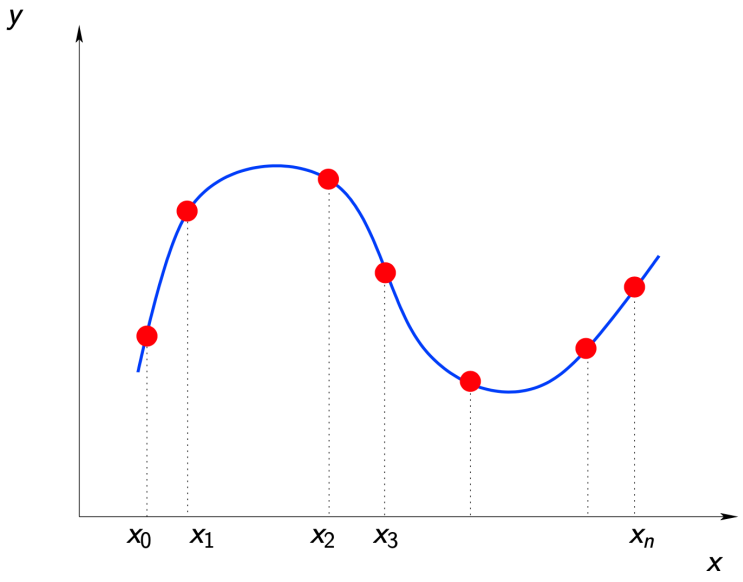


Figure 5.1: Problema de interpolação.

Mas a questão que surge é como construir essa aproximação. Para ilustrar isto, na figura 5.2 mostramos dois exemplos de função $v(x)$ que interpolam os mesmos dados e como podemos apreciar a aproximação fornecida em pontos $x \neq x_i$ é bem distinta. Em geral, a ideia é construir

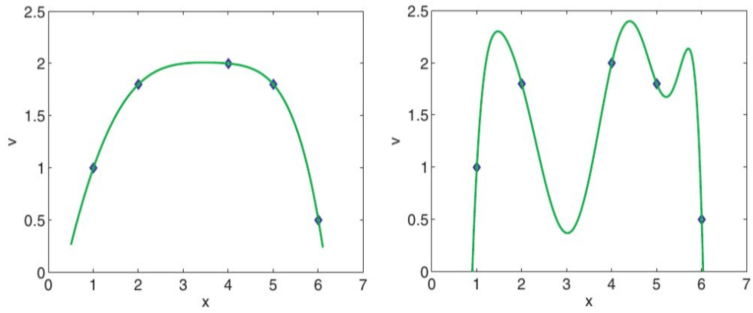


Figure 5.2: Exemplo de duas funções $v(x)$ que interpolam os mesmos dados.

$v(x)$ como uma combinação linear de funções $\phi(x)$ da seguinte forma:

$$v(x) = \sum_{j=0}^{n} c_j \, \phi_j(x) = c_0 \, \phi_0(x) + c_1 \, \phi_1(x) + \cdots + c_n \, \phi_n(x)$$

Existem diferentes escolhas para as funções $\phi(x)$. Dois exemplos clássicossão

- Interpolação polinomial: $\phi_j(x) = x^j$
- Interpolação trigonométrica: $\phi_j(x) = \cos(2\pi j x)$

Nesta disciplina, vamos nos focar no caso da interpolação polinomial, que é o mais simples.

Para começar, vamos pegar os dados e escrever o seguinte sistema de equações que surge de pedir que valham as condições de interpolação, i.e.,

$$v(x_i) = y_i, \quad i = 0, 1, \ldots, n$$

$$
\begin{array}{ccccccccc}
c_0\phi_0(x_0) & + & c_1\phi_1(x_0) & + & \ldots & + & c_n\phi_n(x_0) & = & y_0 \\
c_0\phi_0(x_1) & + & c_1\phi_1(x_1) & + & \ldots & + & c_n\phi_n(x_1) & = & y_1 \\
\vdots & & \vdots & & & & \vdots & & \vdots \\
c_0\phi_0(x_n) & + & c_1\phi_1(x_n) & + & \ldots & + & c_n\phi_n(x_n) & = & y_n
\end{array}
$$

Temos $n+1$ coeficientes a determinar, que são os c_i's (as incógnitas) e temos $n+1$ equações. Em forma matricial seria:

$$
\underbrace{\begin{pmatrix}
\phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \ldots & \phi_n(x_0) \\
\phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_n(x_1) \\
\phi_0(x_2) & \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_n(x_2) \\
\vdots & \vdots & \vdots & \vdots & \\
\vdots & \vdots & \vdots & \vdots & \\
\vdots & \vdots & \vdots & \vdots & \\
\phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \ldots & \phi_n(x_n)
\end{pmatrix}}_{A}
\underbrace{\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_n \end{pmatrix}}_{\mathbf{c}}
=
\underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}}
$$

$$
A\,\mathbf{c} = \mathbf{y}
$$

Esse sistema precisa ser resolvido por algum dos métodos estudados.

Exemplo

Considerar os dados que aparecem na sequência:

x_i	y_i
2	14
6	24
4	25
7	15

- Vamos empregar a base de funções (monômios): $\{1, x, x^2, x^3\}$
- Então a função de interpolação será:

$$
v(x) = c_0\,1 + c_1\,x + c_2\,x^2 + c_3\,x^3
$$

Aplicando o visto antes, obtemos o sistema:

$$
\begin{pmatrix}
1 & x_0 & x_0^2 & x_0^3 \\
1 & x_1 & x_1^2 & x_1^3 \\
1 & x_2 & x_2^2 & x_2^3 \\
1 & x_3 & x_3^2 & x_3^3
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}
=
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}
$$

E utilizando os dados concretos:

$$
\begin{pmatrix}
1 & 2 & 4 & 8 \\
1 & 6 & 36 & 216 \\
1 & 4 & 16 & 64 \\
1 & 7 & 49 & 343
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}
=
\begin{pmatrix} 14 \\ 24 \\ 25 \\ 15 \end{pmatrix}
$$

Para resolver o sistema, podemos usar por exemplo:

```
c = numpy.linalg.solve(A,y)
```

dando:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3.8 \\ 2.767 \\ 1.7 \\ -0.267 \end{pmatrix}$$

o que define o polinômio que interpola os dados fornecidos.

No exemplo anterior:

- A matriz V é chamada matriz de **Vandermonde**
- O determinante de V é:

$$\det(V) = \prod_{0 \le i < j \le n} (x_i - x_j)$$

- Se todos os $\{x_i\}_{i=0}^{n}$ **são distintos**, V é não singular, e o sistema

$$V\,\mathbf{c} = \mathbf{y}$$

tem solução unica $\Rightarrow$ o polinômio achado é único

Implementação em `python`

Em `python` podemos fazer implementar o problema de interpolação de duas formas:

1. Calculando a matriz de Vandermonde e resolvendo o sistema linear associado:

```
# Compute Vandermonde matrix
V = numpy.vander(x, N, increasing=True)

# Compute coefficients of polynomial
c = numpy.linalg.solve(V,y)
```

   Se o `N` não for fornecido então assume-se que `N=len(x)`. A opção `increasing=True` irá gerar a matriz como foi apresentado no exemplo, e nesse caso os coeficientes virão na ordem **ascendente** das potências quando resolvido o sistema, porém, se não especificado, o valor padrão é `increasing=False`, então nesse caso os coeficientes virão na ordem **descendente** das potências quando resolvido o sistema.

2. Também podemos usar uma função que já realiza o cálculo diretamente:

```
# Coefficients of interpol. polynom.
c = numpy.polyfit(x, y, grau)
```

   em que `x` é o vetor com as abcissas e `y` é o vetor dos dados ou ordenadas[1]. É preciso notar que esta função retorna os coeficientes

1: Notar que o `grau` do polinómio tem que ser um a menos que o número de pontos, pois um polinômio de grau n é definido por $n+1$ coeficientes passando exatamente por $n+1$ pontos.

de um polinômio.

$$p(x) = c_0 + c_1\,x + c_2\,x^2 + \cdots + c_n\,x^n, \quad i = 0, 1, \ldots, n$$

por um vetor com os coeficientes escritos na ordem descendente das potencias, i.e.,

$$\mathbf{c} = [c_n, c_{n-1}, \ldots, c_1, c_0]$$

Uma outra função que é de utilidade é função `polyval` que fornece uma forma prática de avaliar um polinômio num conjunto de pontos arbitrário:

```
# Eval the polynom. at a set of points
yeval = np.polyval(c, xeval)
```

em que `c` é o vetor de valores retornado pela função `polyfit`, ou seja, que tem os coeficientes na ordem descendente das potencias.

5.2.1 Interpolação linear por partes

Considerar a função

$$R(x) = \frac{1}{1 + 25\,x^2}$$

e o polinômio interpolante que passa por 11 pontos, como mostrado na figura **??**:

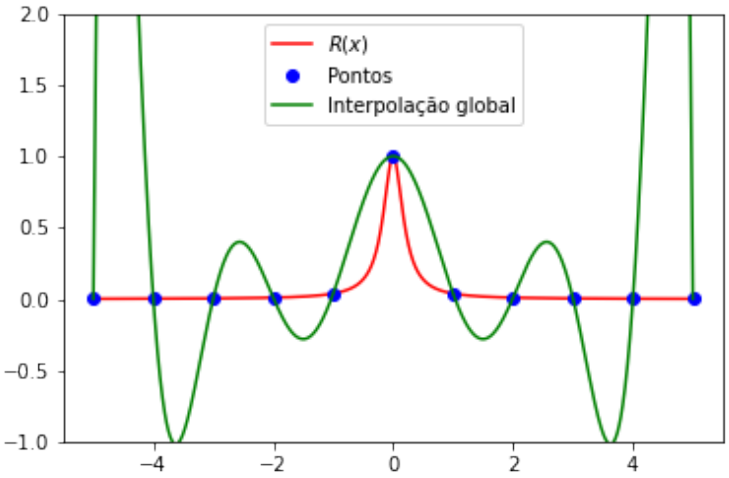


Figure 5.3: Interpolação global da função de Runge $R(x)$.

O fenômeno das oscilações acentuadas é tipico no problema de interpolação global. Uma solução para isto é adotar uma interpolação por partes. O caso mais simples e robusto, corresponde à interpolação linear por partes, tal como ilustrado na figura 5.4. A ideia é simples e consiste em dividir o intervalo e trabalho em partes e em cada parte calcular um polinômio (tipicamente, linear, quadrático ou cúbico) em cada parte. A implementação em `python` mostra-se na sequência.

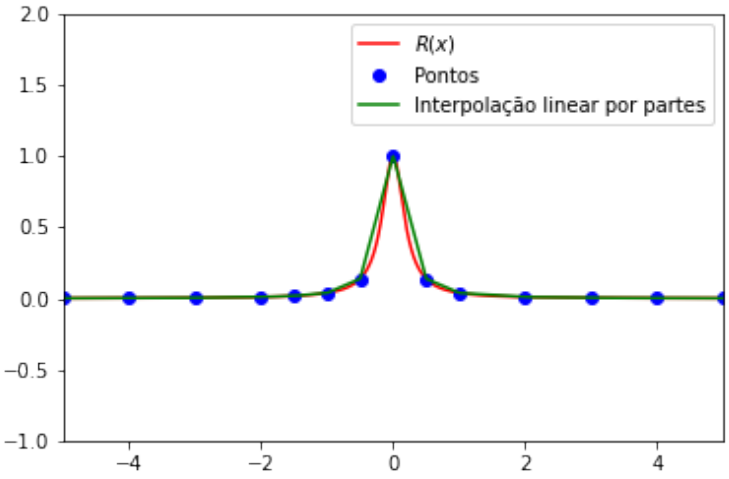


Figure 5.4: Interpolação linear por partes da função de Runge $R(x)$.

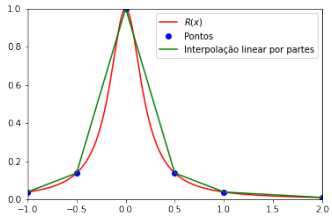


Figure 5.5: Detalhe da figura 5.4.

```
from scipy.interpolate import interp1d

def R(x):
    return 1.0/(1.0 + 25.0*x**2)

# Interpolating points
xi = np.array([-5, -4, -3, -2, -1.5, -1, -0.5, 0, 0.5, 1,
   2, 3, 4, 5])
yi = R(xi)

# Define a set of points to evaluate the functions
xeval = np.linspace(-5, 5, 2000)
yeval = R(xeval)

# Compute the piecewise liner polynomial
ylin = interp1d(xi, yi, kind='linear')

# Plot everything
plt.plot(xi, yi, 'ob',
         xeval, yeval, '-r',
         xeval, ylin(xeval), '-g')
```

5.3 Método dos Mínimos Quadrados e Regressão Linear

Vamos supor que temos dados de um experimento $\{(x_i, y_i)\}_{i=0}^{n}$.

Estes dados poderiam ser como mostrados na figura.

Neste caso, queremos encontrar uma aproximação polinomial (p.e., linear, como mostrado na figura ao lado). É claro que neste caso não é possível encontrar um polinômio que passe exatamente por todos os pontos. A

Table 5.1: Exemplo de dados vindos de um experimento.

x_i	y_i
x_0	y_0
x_1	y_1
x_2	y_2
.	.
.	.
.	.
.	.
x_n	y_n

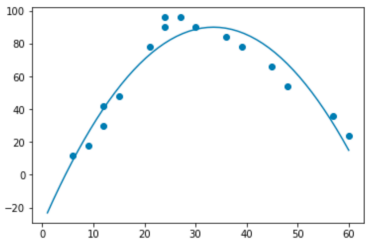


Figure 5.6: Gráfico com o exemplo dos dados vindos de um experimento.

ideia é encontrar um polinômio que aproxime da "melhor forma possível", esses dados.

Pergunta

Para o exemplo anterior, que acontece, se no conjunto de dados, tivermos mais de 4 pontos ou medições, sendo que queremos apenas ajustar um polinômio de grau 2, o qual é definido apenas por 3 coeficientes $c_0,\ c_1,\ c_2$?

$\Rightarrow$ para este exemplo, quando escrever:

$$p(x_i) = y_i, \quad i = 0, 1, \ldots, n$$

$$\begin{array}{ccccccc} c_0 & + & c_1 x_0 & + & c_2 x_0^2 & = & y_0 \\ c_0 & + & c_1 x_1 & + & c_2 x_1^2 & = & y_1 \\ c_0 & + & c_1 x_2 & + & c_2 x_2^2 & = & y_2 \\ c_0 & + & c_1 x_3 & + & c_2 x_3^2 & = & y_3 \\ . & & . & & & . & . \\ . & & . & & & . & . \\ . & & . & & & . & . \\ c_0 & + & c_1 x_n & + & c_2 x_n^2 & = & y_n \end{array}$$

com $n > 3$ (em geral, n pode ser muito grande), acabaremos com um sistema **sobredeterminado**, ou seja, teremos mais equações do que incógnitas. Em forma matricial podemos escrever:

$$\begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ . & . & . \\ . & . & . \\ . & . & . \\ 1 & x_n & x_n^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{pmatrix}$$

- $A \in \mathbb{R}^{(n+1)\times 3}$
- $\mathbf{c} \in \mathbb{R}^3$
- $\mathbf{y} \in \mathbb{R}^{n+1}$

Observação importante

No caso do sistema sobredeterminado $A\mathbf{c} = \mathbf{y}$, $A \in \mathbb{R}^{n\times m}$, $\mathbf{c} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, existirá uma solução (no sentido usual ou clássico) se e só se $b \in \text{col}(A) = \text{Im}(A)$.

Em caso contrário, se

$$\mathbf{y} \notin \text{col}(A) \implies r(\mathbf{c}) = A\mathbf{c} - \mathbf{y} \neq 0\ ,\ \forall\, \mathbf{c} \in \mathbb{R}^n$$

Então, como resolvemos?

Em, geral, para um polinômio de grau máximo m (tipicamente, $m \ll n$), procuramos

$$p_m(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_m x^m$$

tal que

$$\sum_{i=0}^{n} [y_i - p_m(x_i)]^2 \leq \sum_{i=0}^{n} [y_i - q_m(x_i)]^2$$

para qualquer polinômio $q_m(x)$ de grau máximo m. Então, vamos definir a função

$$\Phi(c_0, c_1, c_2, \ldots, c_m) = \sum_{i=0}^{n} [y_i - p_m(x_i)]^2$$

e vamos minimizar ela com respeito a $\mathbf{c} = (c_0, c_1, c_2, \ldots, c_m)$.

$\Rightarrow$ O problema é: Achar $\mathbf{c}$ tal que:

$$\frac{\partial \Phi}{\partial c_i}(c_0, c_1, c_2, \ldots, c_m) = 0, \quad i = 0, 1, \ldots, m$$

Exemplo

Como exemplo, vamos pegar os polinômios de grau 1, i.e.

$$p_m(x) = c_0 + c_1 x$$

$$\Phi(c_0, c_1) = \sum_{i=0}^{n} [y_i - (c_0 + c_1 x_i)]^2$$

Então, vamos fazer:

$$\frac{\partial \Phi}{\partial c_0}(c_0, c_1) = 0, \quad \frac{\partial \Phi}{\partial c_1}(c_0, c_1) = 0$$

que são 2 equações com 2 incógnitas, c_0 e c_1!.

$$\Phi(c_0, c_1) = \sum_{i=0}^{n} [y_i - (c_0 + c_1 x_i)]^2 =$$
$$= \sum_{i=0}^{n} \left[y_i^2 - 2\, y_i\, (c_0 + c_1 x_i) + (c_0 + c_1 x_i)^2 \right]$$

Fazendo as contas:

$$\frac{\partial \Phi}{\partial c_0}(c_0, c_1) = \sum_{i=0}^{n} [-2\, y_i + 2\, (c_0 + c_1 x_i)] = 0,$$

$$\frac{\partial \Phi}{\partial c_1}(c_0, c_1) = \sum_{i=0}^{n} \left[-2\, y_i x_i + 2\,(c_0 + c_1 x_i)\, x_i\right] = 0,$$

$$\begin{aligned} c_0\,(n+1) + c_1 \sum_{i=0}^{n} x_i &= \sum_{i=0}^{n} y_i \\ c_0 \sum_{i=0}^{n} x_i + c_1 \sum_{i=0}^{n} x_i^2 &= \sum_{i=0}^{n} x_i y_i \end{aligned}$$

$$\begin{pmatrix} (n+1) & \sum_{i=0}^{n} x_i \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i\, y_i \end{pmatrix}$$

De maneira geral teremos:

$$\Phi(c_0, c_1, c_2, \ldots, c_m) = \sum_{i=0}^{n} [y_i - p_m(x_i)]^2 = \sum_{i=0}^{n} r_i^2 = \| \mathbf{r} \|_2^2$$

em que o vetor residual é: $\mathbf{r} = \mathbf{y} - A\,\mathbf{c}$. O problema consiste em achar $\mathbf{c}$ que minimiza

$$\Phi(\mathbf{c}) = \| \mathbf{r} \|_2^2$$

As condições necessárias para ter um mínimo em $\mathbf{c}$ são:

$$\nabla \Phi(\mathbf{c}) = 0 \Longleftrightarrow \frac{\partial \Phi}{\partial c_i} = 0, \quad i = 0, 1, \ldots, m$$

Pode-se provar que:

$$\nabla \Phi(\mathbf{c}) = 0 = 2\, A^\intercal\, (A\,\mathbf{c} - \mathbf{y}) \tag{5.1}$$

Equações normais

Para resolver o problema de mínimos quadrados, basta resolver as chamadas equações normais, que são dadas

$$A^\intercal\, A\, \mathbf{c} = A^\intercal\, \mathbf{y}$$

- Lembremos que A era retangular, i.e. $A \in \mathbb{R}^{(n+1)\times(m+1)}$
- $B = (A^\intercal\, A) \in \mathbb{R}^{(m+1)\times(m+1)}$, é quadrada!
- a ordem do polinômio escolhido m é tipicamente pequeno!
- B é não singular se A tem posto completo;
- B é simétrica e definida positiva.

5.4 Cálculo Numérico de Integrais Definidas

A ideia é apresentar algumas fórmulas para o cálculo aproximado da integral de uma função. Lembrando, a ideia intuitiva consiste em calcular

a integral embaixo da função no intervalo [a, x]:

$$F(x) = \int_a^x f(s)\,ds$$

em que f é uma função continua real e de variável real, definida no intervalo finito $a \leq s \leq x$. Para isto, iremos introduzir as chamadas regras de quadratura, que não são outra coisa que fórmulas para avaliar de maneira aproximada essas integrais definidas.

Em que casos precisamos utilizar uma quadratura para calcular a integral definida?

- A função $f(x)$ é conhecida em um conjunto discreto de pontos apenas, p.e., se eles são obtidos através de experimentos (sejam este reais ou virtuais).
- A primitiva não pode ser achada explicitamente, ou não é simples de obter, p.e., consideremos o a integral elíptica de 2a ordem que aparece no cálculo do cumprimento de arco de curvas:

$$I(f) = \int_a^b \sqrt{1 + (\cos(x))^2}\,dx$$

A base da maioria das regras de quadratura é a propriedade aditiva da integral definida, i.e.

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx$$

Então, qual é a ideia:

Ideia

A ideia é aproximar a função f fazendo uma interpolação polinomial **por partes**, e então, calcular a integral do polinômio, o que é facil de se fazer.

Isto leva às chamadas regras de quadratura:

5.4.1 Regras de Newton-Cotes

Na sequência introduzimos três regras de quadratura muito conhecidas:

Fórmulas de quadratura compostas

- **Regra do ponto médio:** Vamos aproximar a função f por um polinômio constante por partes.

$$\mathcal{I}(f) = \int_a^b f(x)\,dx = \sum_{k=1}^{N} \int_{I_k} f(x)\,dx$$

em que $x_k = a + k\,h$, $k = 0, 1, \ldots, N$.

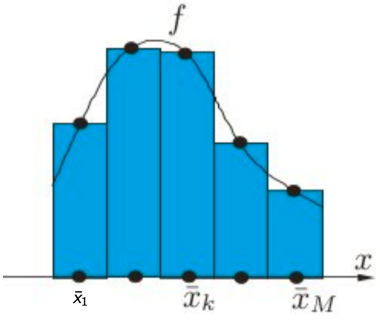


Figure 5.7: Aproximação constante por partes para calcular a integral aproximada de uma função.

Definindo

$$\bar{x}_k = \frac{x_{k-1} + x_k}{2}$$

em cada subintervalo $I_k = [x_{k-1}, x_k]$, $k = 1, 2, \ldots, N$. a constante que aproxima a função é simplesmente $f(\bar{x}_k)$. Isto leva a uma quadratura composta que chamaremos $\mathcal{I}^c_{pm}$:

$$\mathcal{I}(f) \approx \mathcal{I}^c_{pm}(f) = h \sum_{k=1}^{N} f(\bar{x}_k)$$

- **Regra do trapézio:** Se utilizarmos uma interpolação linear por partes

$$\mathcal{I}(f) \approx \mathcal{I}^c_t(f) = \sum_{k=1}^{N} \int_{I_k} p_k(x)$$

em que $p_k(x) = f(x_{k-1}) + f[x_{k-1}, x_k](x - x_{k-1})$, resulta:

$$\mathcal{I}^c_t(f) = \frac{h}{2} \sum_{k=1}^{N} [f(x_{k-1}) + f(x_k)]$$

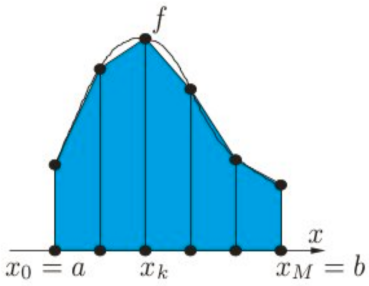


Figure 5.8: Aproximação linear por partes para calcular a integral aproximada de uma função.

- **Regra de Simpson:** Finalmente, se utilizarmos uma aproximação quadrática por partes, teremos em cada subintervalo

$$p_k(x) = 2\frac{(x - \bar{x}_k)(x - x_k)}{h^2} f(x_{k-1}) + 4\frac{(x_{k-1} - x)(x - x_k)}{h^2} f(\bar{x}_k) + 2\frac{(x - \bar{x}_k)(x - x_{k-1})}{h^2} f(x_k)$$

que se corresponde com um polinmio quadrático que interpola a função f nos pontos x_{k-1}, $\bar{x}_k = \frac{x_{k-1} + x_k}{2}$, x_k, resultando na regra de quadratura

$$\mathcal{I}^c_s(f) = \frac{h}{6} \sum_{k=1}^{N} [f(x_{k-1}) + 4f(\bar{x}_k) + f(x_k)]$$

5.5 Cálculo numérico de Derivadas

O último tópico a ser discutido neste capítulo é sobre cálculo aproximado das derivadas de uma função $f(x)$. Assim como no cálculo de integrais numéricas, o cálculo numérico de derivadas se faz necessário quando não conhecemos a expressão da função f e apenas podemos avaliar ela em valores de x arbitrários.

A ideia é calcular o valor da derivada de uma função

$$\mathcal{D}(f) = \frac{df}{dx} = f'(x)$$

em que f é uma função continuamente diferenciável real e de variável real, definida no intervalo finito $a \leq x \leq b$.

Começamos lembrando a definição de derivada de uma função f num

ponto $\bar{x}$:

$$f'(\bar{x}) = \lim_{h \to 0} \frac{f(\bar{x}+h) - f(\bar{x})}{h}$$

Baseado nisto, a primeira fórmula que podemos propor para calcular a derivada de f em $\bar{x}$ é, para h pequeno o suficente

$$f'(\bar{x}) \approx (\delta_+ f)(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x})}{h}$$

que será uma aproximação de $f'(\bar{x})$. Esta fórmula se conhece como diferença finita adiantada (*Forward finite difference*). Vamos supor que f possui derivada segunda continua (i.e., $f \in C^2((a,b))$). Então, podemos fazer a expansão de Taylor:

$$f(\bar{x}+h) = f(\bar{x}) + h\, f'(\bar{x}) + \frac{h^2}{2} f''(\xi)$$

para algum $\xi \in [\bar{x}, \bar{x}+h]$. Da fórmula resulta que:

$$f'(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x})}{h} - \frac{h}{2} f''(\xi) = (\delta_+ f)(\bar{x}) - \frac{h}{2} f''(\xi)$$

o que significa que o erro é $\mathcal{O}(h)$ (é da ordem de h, ou seja, se tomarmos h 10 vezes menor, o erro diminui num fator 10. Similarmente

$$f(\bar{x}-h) = f(\bar{x}) - h\, f'(\bar{x}) + \frac{h^2}{2} f''(\xi)$$

para algum $\xi \in [\bar{x}-h, \bar{x}]$. Da fórmula resulta que:

$$f'(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x}-h)}{h} + \frac{h}{2} f''(\xi) = (\delta_- f)(\bar{x}) + \frac{h}{2} f''(\xi)$$

o que significa que, novamente, o erro é $\mathcal{O}(h)$, sendo:

$$(\delta_- f)(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x}-h)}{h}$$

que é a fórmula de diferença finita atrassada. Finalmente, se considerarmos um desenvolvimento de Taylor até terceiro ordem

$$f(\bar{x}+h) = f(\bar{x}) + h\, f'(\bar{x}) + \frac{h^2}{2} f''(\bar{x}) + \frac{h^3}{6} f'''(\xi_+)$$

para algum $\xi_+ \in [\bar{x}, \bar{x}+h]$ e

$$f(\bar{x}-h) = f(\bar{x}) - h\, f'(\bar{x}) + \frac{h^2}{2} f''(\bar{x}) - \frac{h^3}{6} f'''(\xi_-)$$

para algum $\xi_- \in [\bar{x}-h, \bar{x}]$. Substraindo, obtemos a fórmula centrada

$$f'(\bar{x}) = \underbrace{\frac{f(\bar{x}+h) - f(\bar{x}-h)}{2\,h}}_{(\delta f)(\bar{x})} - \underbrace{\frac{h^2}{12}\left(f'''(\xi_+) + f'''(\xi_-)\right)}_{\mathcal{O}(h^2)}$$

Resumindo, temos as seguintes:

Fórmulas de diferenciação numérica

- **Forward finite difference**

$$(\delta_+ f)(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x})}{h}$$

- **Backward finite difference**

$$(\delta_- f)(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x}-h)}{h}$$

- **Centered finite difference**

$$(\delta f)(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2\,h}$$

5.6 Lista 5 de exercícios

Exercícios

1. Considerar o problema de interpolação com polinômios de grau $n = 2$, que passam por $n+1 = 3$ pontos $\{(x_0, y_0), (x_1, y_1), (x_2, y_2)\}$, usando as funções $\phi_i(x),\ i = 0, 1, 2$:

$$\phi_0(x) = \frac{(x - x_1)\,(x - x_2)}{(x_0 - x_1)\,(x_0 - x_2)}$$

$$\phi_1(x) = \frac{(x - x_0)\,(x - x_2)}{(x_1 - x_0)\,(x_1 - x_2)}$$

$$\phi_2(x) = \frac{(x - x_0)\,(x - x_1)}{(x_2 - x_0)\,(x_2 - x_1)}$$

   Formular o problema algébrico associado para encontrar os coeficientes do polinômio:

$$p(x) = \sum_{i=0}^{2} c_i\,\phi_i(x)$$

   e dar a solução do mesmo. Qual seria a vantagem de usar este tipo de base de funções para o espaço dos polinômios [2].

2. Modificar o exemplo fornecido e calcular o polinomio cúbico por partes para a função de Runge $R(x)$. Considerar outros conjuntos de pontos, inclusive conjuntos que não estejam uniformemente distribuídos no intervalo de interesse.

3. Mostrar que de fato vale a relação que aparece na Eq. 5.1, i.e.,

$$\nabla\Phi(\mathbf{c}) = 2\,A^\intercal\,(A\,\mathbf{c} - \mathbf{y})$$

4. Considerar o arquivo `data.txt` disponível no Tidia, o qual disponibiliza dados experimentais. Queremos estudar possíveis relações entre os dados que aparecem nas diversas colunas desse arquivo. Fazer um código de `python` que carrega o arquivo e

2: Este tipo de funções chamam-se polinômios de Lagrange.

aplica o método das equações normais para estabelecer qual é a melhor relação entre os dados (no sentido dos quadradros mínimos), considerando as seguintes possibilidades:

- $\mathbf{c}_{(3)} \approx k_1 + k_2\mathbf{c}_{(1)}$
- $\mathbf{c}_{(3)} \approx k_1 + k_2\mathbf{c}_{(1)} + k_3\mathbf{c}_{(1)}^2$
- $\mathbf{c}_{(4)} \approx k_1 + k_2\mathbf{c}_{(1)} + k_3\mathbf{c}_{(2)}$

Fazer gráficos mostrando os resultados. Notar que para mostrar o gráfico do polinômio que aproxima os dados em cada caso será necessário avaliar ele em vários pontos no intervalo de interesse, para isto, pode usar por exemplo a função `polyval`.

5. Considerar a integral

$$\int_0^1 \frac{4}{1+x^2}\,dx = 4\,\text{atan}(x)|_0^1 = \pi.$$

Usando as diferentes regras compostas de Newton-Cotes vistas, calcular o valor numérico da integral e calcular o erro do resultado (i.e., $e = |\text{valor exato} - \text{valor numérico}|$). Plotar esse erro como função do número de subintervalos usando a função `loglog` para plotar em papel logarítmico. Que conclusões pode tirar?

6. Considerar as redes hidráulicas estudadas anteriormente, para o qual foi desenvolvida a função `SolveNetwork()` e a função que calcula a potência consumida pela bomba. Considerar uma rede com os seguintes parâmetros:
   - `n = 8`
   - `m = 9`
   - `QB = 3`
   - `natm = n*m - 1`
   - `nB = 0`
   - As condutâncias dependem de um parâmetro x:

$$\begin{aligned} CH &= 2.3 + 10\,e^{-(x-5)^2}, \\ CV &= 1.8 + 10\,e^{-(x-5)^2} \end{aligned}$$

   Calcular a integral da potencia no intervalo $1 \le x \le 10$ usando a regra do ponto médio, do trapezio e de Simpson compostas considerando $2, 4, 6, 8, 10$ intervalos. Organizar os resultados numa tabela.

7. No exercício anterior, calcular a derivada de função da potência consumida pela bomba como função de x usando a regras de diferenciação numérica centrada. Plotar o resultado.

8. Considerar a função de uma variável

$$f(x) = x\,e^{-x}\,\cos(2x)$$

   (a) Calcular a derivada primeira f' exata e plotar no intervalo $[0, \pi]$.

   (b) Usando as fórmulas de diferenciação numérica, calcular a derivada f'_a (aproximada) e plotar como função de x no

intervalo $[0, \pi]$. Usar valores de $\delta = 0.2, 0.1, 0.05$ e 0.025.

(c) Considerar o ponto $\bar{x} = \pi/2$. Calcular o erro das fórmulas nesse ponto para $(\delta_+ f)(\bar{x})$ e $(\delta f)(\bar{x})$, i.e.,

$$e(\bar{x}) = |f'(\bar{x}) - f'_a(\bar{x})|$$

e plotar o resultado como função de δ. Para isto tomar valores de $\delta = 0.25/10^k, \quad k = 0, 1, \ldots, 10$. Usar a função `loglog` para plotar o erro em papel logarítmico. Que acontece quando δ é muito pequeno?.

(d) (**BONUS**) Considerar a seguinte fórmula para o cálculo da derivada segunda de uma função:

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

e o erro desta aproximação é $-\frac{h^2}{24}(f^{(iv)}(\xi_+) + f^{(iv)}(\xi_-))$, ou seja, é um errro $\mathcal{O}(h^2)$). Aplicar a fórmula na função anterior e verificar a ordem da aproximação calculando o erro como função de h no ponto $\bar{x} = \pi/2$.

PROBLEMAS NÃO LINEARES 6

6.1 O método de Newton

Vamos considerar o problema:

Achar $x^* \in \mathbb{R}$ tal que

$$r(x^*) = 0$$

em que a função $r : \mathbb{R} \to \mathbb{R}$, pode ser não linear em x.

Que é um método iterativo para resolver esse problema?

Um método iterativo para achar a solução x^* do problema, é um método que, a partir de um valor inicial $x^{(0)}$, gera uma sequência de valores $x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots$, que aproxima-se a x^*

Que significa se aproximar a x^*?

Matematicamente, o limite da sequência quando k tende para ∞ tem que ser x^*, i.e.

$$\lim_{k\to\infty} x^{(k)} = x^*$$

Então, o método iterativo diz-se convergente. De maneira equivalente, se o método for convergente

$$|x^{(k)} - x^*| \xrightarrow[k\to\infty]{} 0$$

e

$$|r(x^{(k)})| \xrightarrow[k\to\infty]{} 0$$

Existem vários métodos para resolver o caso em que $r(x)$ é não linear. Um dos métodos mais usados na engenharia é o método de Newton, no qual a receita prática para construir a sequência de aproximações é:

$$x^{(k+1)} = x^{(k)} - \frac{r(x^{(k)})}{r'(x^{(k)})} \tag{6.1}$$

O valor inicial x^0 (o chute) é importante. Se ele não estiver razoavelmente perto da solução, o método poderia não convergir ou convergir para uma solução que não é de interesse.

O algoritmos é:

Método de Newton

Dados $x^{(0)}, TOL, MAX\_IT, k = 0$

Enquanto $\varepsilon > TOL$ e $k < MAX\_IT$

1. Calcular avanço: $\delta = -\dfrac{r(x^{(k)})}{r'(x^{(k)})}$
2. Avançar: $x^{(k+1)} = x^{(k)} + \delta$
3. Calcular o erro $\varepsilon = |x^{(k+1)} - x^{(k)}|/|x^{(k)}|$
4. Incrementar k

Fim

Este método será estudado na última atividade a ser desenvolvida em grupo e com relatório que será entregue em data a ser definida, segue na sequência:

6.2 Atividade 3

Exercícios

1. Pesquisar de onde vem a fórmula dada em 6.1. Pode usar o livro de Quarteroni e Salieri (recomendado no inicio do semestre) ou qualquer outro livro de cálculo numérico que você preferir.
2. Programar e aplicar o método para encontrar as raizes da função:

   $$r(x) = \cos(x) - x$$

   Conferir o resultado usando a função:

   ```
   x = scipy.optimize.fsolve(fun, x0, fprime=derfun)
   ```

3. Interpretar geometricamente o que está acontecendo no processo iterativo a cada passo.
4. Nos casos em que não sabemos calcular a derivada da função, precisamos aplicar uma fórmula de diferenciação numérica. Por exemplo, para o caso das redes hidráulicas, se quisermos saber em que valor do parâmatro x a função $W(x)$ da potência consumida pela bomba é igual a 12, deveriamos aplicar o método de Newton na função

   $$r(x) = W(x) - 12 = 0$$

   Neste caso, quando precisarmos calcular a derivada de W podemos fazer:

   $$W'(x) \approx \frac{W(x+h) - W(x-h)}{2\,h}$$

   (tomando algum h pequeno, por exemplo $h \sim 10^{-3}$). Agora, considerar uma rede com os seguintes parâmetros:

   - `n = 8`
   - `m = 9`
   - `QB = 3`
   - `natm = n*m - 1`
   - `nB = 0`

- As condutâncias dependem de um parâmetro x:

$$\begin{aligned} CH &= 2.3 + 10\, e^{-(x-5)^2}, \\ CV &= 1.8 + 10\, e^{-(x-5)^2} \end{aligned}$$

Determinar em que valores de x, $W(x) = 12$. Notar que dependendo do chute inicial $x^{(0)}$ o resultado poderá ser diferente. Fazer um gráfico da função e conferir se o resultado é correto.

RESOLUÇÃO NUMÉRICA DE EDOs NA ENGENHARIA

7

7.1 Preludio

A maioria dos sistemas físicos que aparecem na engenharia são sistemas que possuem uma certa dinâmica e que evoluem no tempo. Nesse caso, se faz necessário resolver equações ou sistemas de equações que envolvem derivadas com respeito ao tempo, ou seja, equações diferenciais ordinárias, as quais precisam ser resolvidas numéricamente, pois raramente é possível encontrar uma solução analítica por meios simbólicos. Para isto, neste capítulo iremos discutir os tópicos na sequência:

Objetivos

- Estudar alguns exemplos de equações e sistemas de equações diferenciais ordinárias;
- Os métodos de Euler e as suas variantes;
- Os métodos de Runge-Kutta.

7.2 Exemplos de EDOs

Vamos estudar métodos numéricos para resolver problemas de valor inicial do tipo

$$y'(t) = \frac{dy(t)}{dt} = f\,(t, y(t)) \quad \forall\, t \in I = [t_0, T] \subset \mathbb{R}$$

junto com a condição inicial

$$y(t_0) = y_0$$

Exemplo 1: Evolução populacional

Consideremos o equação de primeira ordem

$$\begin{cases} \dfrac{dy(t)}{dt} &= \lambda\, y \\ y(0) &= y_0 \end{cases}$$

$\lambda \in \mathbb{R}$. Neste caso, a função f é linear:

$$f(t, y(t)) = \lambda\, y$$

mas, não tem dependência explícita na variável independente t.

Exemplo 2: Problema não linear

Consideremos o problema de primeira ordem

$$\begin{cases} \dfrac{dy(t)}{dt} &= t^2 + y^2(t) \\ y(0) &= \dfrac{1}{2} \end{cases}$$

Neste caso, a função f é não linear e tem dependência explicíta na variável independente t:

$$f(t, y(t)) = t^2 + y^2(t)$$

Exemplo 3: Sistema não linear de Lotka-Volterra

Agora, consideremos um sistema de duas equações de primeira ordem, que é um modelo de interação entre presas e predadores:

$$\begin{cases} \dfrac{du(t)}{dt} &= f_1(u, v) = (\alpha - \beta\, v)\, u \quad \text{Presa} \\ \dfrac{dv(t)}{dt} &= f_2(u, v) = (\delta\, u - \gamma)\, v \quad \text{Predador} \end{cases}$$

Com condições iniciais $u(0) = u_0$ e $v(0) = v_0$.

$$\mathbf{f}(\mathbf{y}) = \begin{pmatrix} f_1(\mathbf{y}) \\ \\ f_2(\mathbf{y}) \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} u \\ \\ v \end{pmatrix}$$

Nos primeiros dois exemplos, até que poderiamos aplicar alguma metodologia para encontrar uma solução fechada das equações, porém, neste último exemplo, pela complexidade, apenas podemos recorrer a métodos numéricos para predizer o comportamento do sistema. Na figura 7.1 mostra-se uma tal solução, cuja evolução não é trivial.

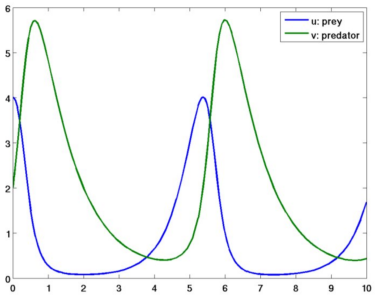


Figure 7.1: Evolução do sistema de Lotka-Volterra que modeliza o comportamento de populações que interagem.

Exemplo 4: O pêndulo

Aplicamos a lei de Newton para uma massa pendurada:

$$m\,\ell^2 \frac{d^2\theta(t)}{dt^2} = m\,\ell^2\theta''(t) = -m\,g\,\ell\,\sin(\theta(t))$$

em que

- m é a massa;
- ℓ é o comprimento
- θ é o ângulo e θ'' é a aceleração angular
- $g = 9.81\ \mathrm{m/s^2}$

Esta é uma equação de segunda ordem, que podemos transformar en

duas equações de primeira ordem. Para isto, definimos:

$$y_1(t) = \theta(t), \quad y_2(t) = \theta'(t)$$

Então, podemos escrever:

$$\begin{aligned} y_1'(t) &= \theta'(t) = \dots = y_2(t) \\ y_2'(t) &= \theta''(t) = -\underbrace{\frac{g}{\ell}}_{\omega^2} \sin(\theta(t)) = -\omega^2 \sin(y_1(t)) \end{aligned}$$

Então, podemos escrever:

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= -\omega^2 \sin(y_1(t)) \end{aligned}$$

ou usando notação vetorial

$$\mathbf{y}'(\mathbf{t}) = \begin{pmatrix} y_1'(t) \\ \\ y_2'(t) \end{pmatrix} = \begin{pmatrix} y_2(t) \\ \\ -\omega^2 \sin(y_1(t)) \end{pmatrix} = \mathbf{f}(t, \mathbf{y}(t))$$

Que é um sistema de duas equações ordinarias de primeira ordem. Com notação vetorial escreveremos em geral

$$\mathbf{y}'(\mathbf{t}) = \frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(t, \mathbf{y}(t))$$

em que precisamos uma condição inicial $\mathbf{y}(0) = \mathbf{y}_0$. Por exemplo, para o caso do pêndulo precisamos:

$$\mathbf{y}(\mathbf{0}) = \begin{pmatrix} y_1(0) \\ \\ y_2(0) \end{pmatrix} = \begin{pmatrix} \theta(0) \\ \\ \theta'(0) \end{pmatrix}$$

Isto é o ângulo e a velocidade inicial da massa.

7.3 Métodos numéricos para EDOs

A idéia é gerar uma sequência de valores $t_0, t_1, \dots,$ e uma sequência correspondente de valores $y_0, y_1, \dots,$, tais que y_n aproxima a solução exata do problema em t_n, i.e.,

$$y_n \approx y(t_n), \quad n = 0, 1, \dots \tag{7.1}$$

Definimos o tamanho do passo como

$$h = t_{n+1} - t_n$$

As vezes, o passo h é chamado Δt ou δt. Na maiora dos métodos que estudaremos, h será considerado constante, embora, em métodos mais sofisticados, o passo h é variável, de tal forma de controlar a precisão dos resultados de maneira mais eficiente.

Agora precisamos fazer algumas aproximações, o que leva a distintos métodos de resolução:

7.3.1 Os métodos de Euler e as suas variantes

Método explícito

É o método mais simples para resolver numericamente uma EDO

- A ideia é definir pontos no intervalo de interesse: $a = t_0 < t_1 < t_2 < \cdots < t_N = b$, considerando um passo fixo $h = t_{n+1} - t_n$
- Vamos avançar desde um ponto t_n ao próximo t_{n+1}, passo por passo.
- Escrevemos:

$$y'(t_n) = \frac{y(t_{n+1}) - y(t_n)}{h} + \mathcal{O}(h)$$

Mas, lembremos que $y'(t) = f(t, y(t))$, então:

$$f(t_n, y(t_n)) = \frac{y(t_{n+1}) - y(t_n)}{h} + \mathcal{O}(h)$$

$$y(t_{n+1}) = y(t_n) + h\, f(t_n, y(t_n)) + \mathcal{O}(h^2)$$

- O que motiva o método numérico:

$$\begin{aligned} y_{n+1} &= y_n + h\, f(t_n, y_n) \\ t_{n+1} &= t_n + h \end{aligned}$$

Com condição inicial $y(t_0) = y_0$. Se estivessemos tratando com um sistema, o método é totalmente analogo:

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\, \mathbf{f}(t_n, \mathbf{y}_n) \end{aligned} \tag{7.2}$$

Aviso importante

O método anterior, diz-se **explícito** pois para avançar no tempo só usamos informação dos passos de tempo anteriores. Os métodos explícitos possuem de maneira geral uma restrição de passo de tempo h, i.e., o h precisa ser escolhido suficientemente pequeno para que o cálculo não diverja. Em contrapartida, o método é muito simples de ser aplicado, pois o único que precisamos fazer para avançar no tempo é avaliar a função $\mathbf{f}(t_n, \mathbf{y}_n)$.

Método implícito

É uma versão numéricamente mais estável do método anterior:

- A ideia é definir pontos no intervalo de interesse: $a = t_0 < t_1 < t_2 < \cdots < t_N = b$, considerando um passo fixo $h = t_{n+1} - t_n$

- Vamos avançar desde um ponto t_n ao próximo t_{n+1}, passo por passo. Neste caso escrevemos:

$$y'(t_{n+1}) = \frac{y(t_{n+1}) - y(t_n)}{h} + \mathcal{O}(h)$$

Mas, lembremos que $y'(t) = f(t, y(t))$, então:

$$f(t_{n+1}, y(t_{n+1})) = \frac{y(t_{n+1}) - y(t_n)}{h} + \mathcal{O}(h)$$

$$\Rightarrow y(t_{n+1}) = y(t_n) + h\, f(t_{n+1}, y(t_{n+1})) + \mathcal{O}(h^2)$$

- O que motiva o método numérico:

$$\begin{aligned} y_{n+1} &= y_n + h\, f(t_{n+1}, y_{n+1}) \\ t_{n+1} &= t_n + h \end{aligned}$$

Com condição inicial y_0.

Aviso importante

Mas, agora aparece um problema: A incógnita y_{n+1} aparece tanto do lado esquerdo como do lado direito. Se a função $f(t, y)$ for não linear no seu segundo argumento $\Rightarrow$ precisaremos resolver, em cada passo, a equação não linear:

$$y_{n+1} - y_n - h\, f(t_{n+1}, y_{n+1}) = 0$$

para achar y_{n+1}, $n = 0, 1, \ldots \rightarrow$ Isto é mais costoso!

Se estivessemos resolvendo um sistema seria pior aínda, pois precisamos resolver um sistema não linear:

$$\mathbf{y}_{n+1} - \mathbf{y}_n - h\, \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) = 0$$

Generalização

Uma variante que mistura os dois métodos anteriores é chamada de método α, na qual se faz uma média pesada da avaliação da função Definindo um parámetro α, $0 \leq \alpha \leq 1$, temos:

Método α

$$y_{n+1} = y_n + h\, [(1-\alpha) f(t_n, y_n) + \alpha\, f(t_{n+1}, y_{n+1})]$$

- Euler explícito $\rightarrow \alpha = 0$
- Método Trapezoidal implícito $\rightarrow \alpha = \frac{1}{2}$
- Euler implícito $\rightarrow \alpha = 1$

Para o método que se chama de trapezoidal, uma forma interessante de obter ele, que aproveita o que temos aprendido antes é :

$$\frac{dy}{dt} = f(t, y(t)) \implies y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

e usando a regra do trapézio para integrar resulta o método:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$$

já que $h = t_{n+1} - t_n$.

Método trapezoidal explícito

Uma forma de tornar o método trapezoidal mais simples, é combinando ele com o método de Euler explícito da seguinte forma: Partimos do método Trapezoidal implícito:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$$

Agora, fazemos a aproximação: $y_{n+1} = y_n + h\, f(t_n, y_n)$ que corresponde ao método de Euler explícito e leva ao método[1]:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_n + h\, f(t_n, y_n))\right]$$

1: Este método também é conhecido como método preditor-corretor, pois é feita uma predição inicial usando o método de Euler e a posteriori se corrige usando o método trapezoidal.

7.3.2 Os métodos de Runge-Kutta

Os métodos de Runge-Kutta, são métods muito usados em física e engenharia pela sua simplicidade. Um método explícito de Runge-Kutta de r estagios é um método da forma:

$$y_{n+1} = y_n + h\,\Phi_r$$

em que

$$\Phi_r = c_1\, k_1 + c_2\, k_2 + \cdots + c_r\, k_r$$

$$\begin{aligned}
k_1 &= f(t, y) \\
k_2 &= f(t + \alpha_2\, h, y + h\,\beta_{21}\, k_1) \\
k_3 &= f(t + \alpha_3\, h, y + h\,(\beta_{31}\, k_1 + \beta_{32}\, k_2)) \\
k_4 &= f(t + \alpha_4\, h, y + h\,(\beta_{41}\, k_1 + \beta_{42}\, k_2 + \beta_{43}\, k_3)) \\
&\vdots \\
k_r &= f(t + \alpha_r\, h, y + h\,(\beta_{r1}\, k_1 + \cdots + \beta_{r,r-1}\, k_{r-1}))
\end{aligned}$$

Agora, dependendo das escolhas para os parâmetros α, β e c's teremos distintos métodos:

- Runge-Kutta RK1: Este método coincide com o método de Euler explícito, i.e.,

$$y_{n+1} = y_n + h\, c_1\, k_1$$

- $r = 1$
- $k_1 = f(t_n, y_n),\ c_1 = 1$

$$\Rightarrow y_{n+1} \quad = \quad y_n + h\, f(t_n, y_n)$$

- Runge-Kutta RK2 Este método de ordem 2, coincide com o método Trapezoidal explícito:

$$y_{n+1} \quad = \quad y_n + h\,(c_1\, k_1 + c_2\, k_2)$$

- $r = 2$
- $k_1 = f(t_n, y_n),\ c_1 = \dfrac{1}{2}$
- $k_2 = f(t_n + h, y_n + h\, k_1),\ c_2 = \dfrac{1}{2}$

$$y_{n+1} \quad = \quad y_n + \frac{h}{2}\,[f(t_n, y_n) + f(t_n + h, y_n + h\, f(t_n, y_n))]$$

- Runge-Kutta RK4: Este método é de ordem 4 e se escreve:

$$y_{n+1} \quad = \quad y_n + h\,(c_1\, k_1 + c_2\, k_2 + c_3\, k_3 + c_4\, k_4)$$

- $r = 4$
- $k_1 = f(t_n, y_n),\ c_1 = \dfrac{1}{6}$
- $k_2 = f(t_n + \frac{1}{2}\, h, y_n + \frac{1}{2}\, h\, k_1),\ c_2 = \dfrac{1}{3}$
- $k_3 = f(t_n + \frac{1}{2}\, h, y_n + \frac{1}{2}\, h\, k_2),\ c_3 = \dfrac{1}{3}$
- $k_4 = f(t_n + h, y_n + h\, k_3),\ c_4 = \dfrac{1}{6}$

$$y_{n+1} \quad = \quad y_n + \frac{h}{6}\,(k_1 + 2\, k_2 + 2\, k_3 + k_4)$$

7.4 Erro das aproximações

Para quantificar o erro dos diferentes métodos temos duas medidas:

- **Erro global:** é a diferença entre a solução exata num determinado tempo e a solução do método numérico:

$$\epsilon_n = y(t_n) - y_n$$

- **Erro num passo:** é o erro feito avançando um passo do método numérico a partir de uma condição inicial que coincide com a solução exata:

$$\varepsilon_n = y(t_n) - [y(t_{n-1}) + h\,\Phi(t_{n-1}, y(t_{n-1}), h)]$$

em que $\Phi(\cdot, \cdot)$ é a função que aparece no método numérico considerado.

Teorema de convergência

Seja um método numérico de um passo para resolver EDOs no qual a função do método Φ é Lipschitz em y. Vamos supor que o método é estável[2] e é consistente, i.e., $\epsilon_n \to 0$ quando $h \to 0$, então o método é convergente, i.e., existe $p > 0$ e $c > 0$, tais que $|\epsilon_n| \leq c\, h^p$.

2: O conceito de estabilidade que precisaríamos definir chama-se de zero-estabilidade, e tem a ver com o fato de que duas condições iniciais muito próximas levem a soluções numéricas muito próximas quando h é suficentemente pequeno.

Alphabetical Index