Matemática Numérica Avançada

Pedro H A Konzen

9 de fevereiro de 2022

Licença

Prefácio

Nestas notas de aula são abordados métodos numéricos aplicados a problemas de grande porte. Como ferramenta computacional de apoio, exemplos de aplicação de códigos Python, são apresentados, mais especificamente, códigos com suporte das bibliotecas NumPy e SciPy.

Agradeço a todos e todas que de modo assíduo ou esporádico contribuem com correções, sugestões e críticas. :)

Pedro H A Konzen

Sumário

Capa i

Licença ii

Prefácio iii

Sumário iv

1 Sistemas Lineares 1

1.1 Matrizes Esparsas 1

1.1.1 Sistemas Tridiagonais 3

1.1.2 Matrizes Banda 6

Referências Bibliográficas 10

Capítulo 1

Sistemas Lineares

[Vídeo] | [Áudio] | [Contatar]

Neste capítulo, apresentam-se métodos numéricos para a resolução de sistemas lineares de grande porte. Salvo explicitado ao contrário, assume-se que os sistemas são quadrados e têm solução única.

1.1 Matrizes Esparsas

[Vídeo] | [Áudio] | [Contatar]

Uma matriz é dita ser **esparsa** quando ela tem apenas poucos elementos não nulos. A ideia é que os elementos não nulos não precisam ser guardados na memória do computador, gerando um grande benefício na redução da demanda de armazenamento de dados. O desafio está no desenvolvimento de estruturas de dados para a alocação eficiente de tais matrizes, i.e. que sejam suficientemente adequadas para os métodos numéricos conhecidos.

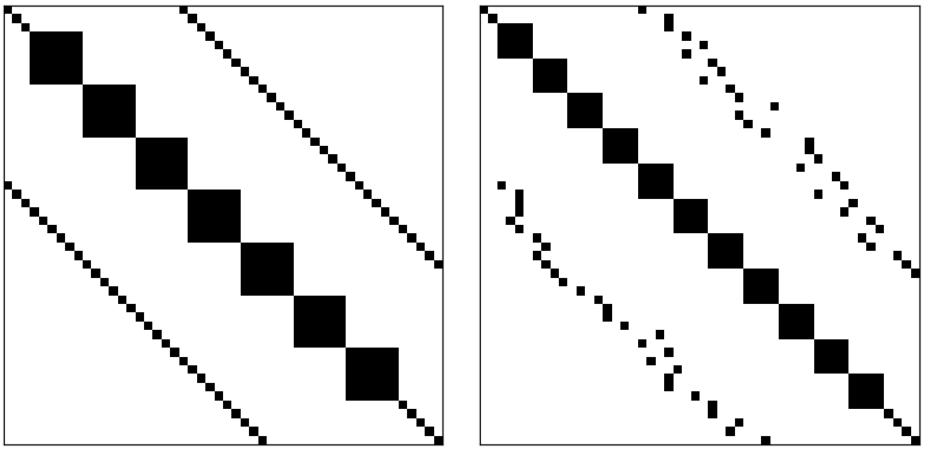

Figura 1.1: Esquerda: exemplo de uma matriz esparsa estruturada. Direita: exemplo de uma matriz esparsa não-estruturada.

Matrizes esparsas podem ser classificadas como **estruturadas** ou **não-estruturadas**. Uma matriz estruturada é aquela em que as entradas não-nulas formam um padrão regular. Por exemplo, estão dispostas em poucas diagonais ou formam blocos (submatrizes densas) ao longo de sua diagonal principal. No caso de não haver um padrão regular das entradas não-nulas, a matriz esparsa é dita ser não-estruturada. Consulte a Figura 1.1 para exemplos.

A **esparsidade** de uma matriz é a porcentagem de elementos nulos que ela tem, i.e. para uma matriz quadrada $n \times n$ tem-se que a esparsidade é

$$\frac{n_{\text{nulos}}}{n^2} \times 100\% \tag{1.1}$$

Por exemplo, a matriz identidade de tamanho $n = 100$ tem esparsidade

$$\frac{100^2 - 100}{100^2} \times 100\% = 99\% \tag{1.2}$$

1.1.1 Sistemas Tridiagonais

Um sistema tridiagonal tem a seguinte forma matricial

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix} \tag{1.3}$$

com, $a_1 = 0$ e $c_n = 0$. Ou seja, é um sistema cuja a matriz dos coeficientes é tridiagonal.

Uma **matriz tridiagonal** é uma matriz esparsa estruturada. Mais especificamente, é chamada de matriz banda, em que os elementos não nulos estão apenas em algumas de suas diagonais. Para armazenarmos tal matriz precisamos alocar apenas os seguintes três vetores

$$a = (0,a_2,\ldots,a_n) \tag{1.4}$$
$$b = (b_1,b_2,\ldots,b_n) \tag{1.5}$$
$$c = (c_1,c_2,\ldots,c_{n-1},0) \tag{1.6}$$

Ou seja, precisamos armazenar $3n$ pontos flutuantes em vez de n^2, como seria o caso se a matriz dos coeficientes fosse densa.

Algoritmo de Thomas

O Algoritmo de Thomas[1] é uma forma otimizada do Método de Eliminação Gaussiana aplicada à sistemas tridiagonais. Enquanto este requer $O(n^3)$ operações, esse demanda apenas $O(n)$.

Eliminando os termos abaixo da diagonal em (1.3), obtemos o sistema equivalente

$$\begin{bmatrix} \tilde{b}_1 & c_1 & & & 0 \\ & \tilde{b}_2 & c_2 & & \\ & & \tilde{b}_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & & \tilde{b}_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \tilde{d}_1 \\ \tilde{d}_2 \\ \tilde{d}_3 \\ \vdots \\ \tilde{d}_n \end{bmatrix} \tag{1.7}$$

[1] Llewellyn Hilleth Thomas, 1903 - 1992, físico e matemático aplicado britânico. Fonte: Wikipedia.

Este é obtido pela seguinte iteração

$$w := \frac{a_i}{b_{i-1}} \tag{1.8}$$

$$b_i := b_i - wc_{i-1} \tag{1.9}$$

$$d_i := d_i - wd_{i-1} \tag{1.10}$$

onde, o ˜ foi esquecido de propósito, indicando a reutilização dos vetores b e d. A solução do sistema é, então, obtida de baixo para cima, i.e.

$$x_n = \frac{d_n}{b_n} \tag{1.11}$$

$$x_i = \frac{d_i - c_i x_{i+1}}{b_i}, \tag{1.12}$$

com $i = n-1$,$n-2,\ldots,1$.

Listing 1.1: Algoritmo de Thomas

```
1  import numpy as np
2
3  def TDMA(a,b,c,d):
4      n = b.size
5      for i in np.arange(1,n):
6        w = a[i]/b[i-1]
7        b[i] = b[i] - w*c[i-1]
8        d[i] = d[i] - w*d[i-1]
9      x = np.empty(n)
10     x[n-1] = d[n-1]/b[n-1]
11     for i in np.arange(n-2,-1,-1):
12       x[i] = (d[i] - c[i]*x[i+1])/b[i]
13     return x
```

E 1.1.1. Considere o seguinte sistema linear

$$2x_1 - x_2 = 0 \tag{1.13}$$

$$x_{i-1} - 3x_i + 4x_{i+1} = \text{sen}\left(i\frac{\pi}{2(n-1)}\right) \tag{1.14}$$

$$x_{n-1} + x_n = 1 \tag{1.15}$$

a) Compute sua solução usando o Algoritmo de Thomas para $n = 3$.

b) Compare a solução obtida no item anterior com a gerada pela função `scipy.linalg.solve`.

c) Compare a solução com a obtida no item anterior com a gerada pela função `scipy.linalg.solve_banded`.

d) Use o módulo Python timeit para comprar a demanda de tempo computacional de cada um dos métodos acima. Compute para $n = 10{,}100{,}1000{,}10000$.

E 1.1.2. Considere que o problema de valor de contorno (PVC)

$$-u'' = \operatorname{sen} \pi x, \quad 0 < x < 1, \tag{1.16}$$
$$u(0) = 0, \tag{1.17}$$
$$u(1) = 0 \tag{1.18}$$

seja simulado com o Método das Diferenças Finitas[2]. Vamos assumir uma discretização espacial uniforme com n nodos e tamanho de malha

$$h = \frac{1}{n-1}. \tag{1.19}$$

Com isso, temos os nodos $x_i = (i-1)h$, $i = 1,2,\ldots,n$. Nos nodos internos, aplicamos a fórmula de diferenças central

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}, \tag{1.20}$$

onde, $u_i \approx u(x_i)$. Com isso, a discretização da EDO fornece

$$-\frac{1}{h^2}u_{i-1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i+1} = \operatorname{sen} \pi x_i \tag{1.21}$$

para $i = 2,3,\ldots,n-1$. Das condições de contorno temos $u_1 = u_n = 0$. Logo, o problema discreto lê-se: encontrar $u = (u_1,u_2,\ldots,u_n) \in \mathbb{R}^n$ tal que

$$u_1 = 0 \tag{1.22}$$
$$-\frac{1}{h^2}u_{i-1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i+1} = \operatorname{sen} \pi x_i \tag{1.23}$$
$$u_n = 0 \tag{1.24}$$

[2]Consulte mais em Notas de Aula: Matemática Numérica.

a) Calcule a solução analítica do PVC.

b) Use a função `scipy.linalg.solve_banded` para computar a solução do problema discreto associado para diferentes tamanhos de malha $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$. Compute o erro da solução discreta em relação à solução analítica.

c) Compare a demanda de tempo computacional se a função `scipy.linalg.solve` for empregada na computação da solução discreta.

1.1.2 Matrizes Banda

Uma matriz banda é aquela em que os elementos não nulos estão dispostos em apenas algumas de suas diagonais. Consulte a Figura 1.2.

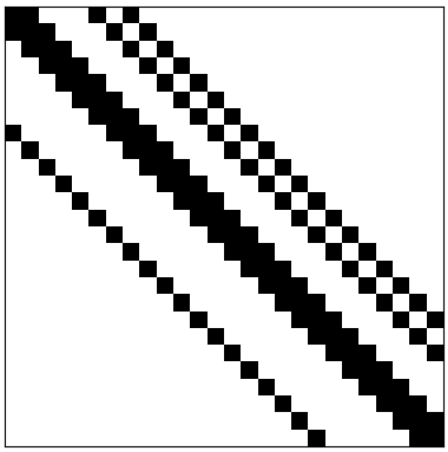

Figura 1.2: Exemplo de uma matriz banda.

E 1.1.3. Considere o seguinte problema de Poisson[3]

$$u_{xx} + u_{yy} = -\operatorname{sen}(x)\operatorname{sen}(y),\ (x,y) \in (0,\pi) \times (0,\pi), \tag{1.25}$$
$$u(0,y) = 0,\ y \in [0,\pi], \tag{1.26}$$
$$u(\pi,y) = 0,\ y \in [0,\pi], \tag{1.27}$$
$$u(x,0) = 0,\ x \in [0,\pi], \tag{1.28}$$
$$u(x,\pi) = 0,\ x \in [0,\pi]. \tag{1.29}$$

Vamos empregar o Método de Diferenças Finitas para computar uma aproximação para a sua solução. Começamos assumindo uma malha uniforme de n^2 nodos

$$x_i = (i-1)h \tag{1.30}$$
$$y_j = (j-1)h \tag{1.31}$$

com tamanho de malha $h = \pi/(n-1)$, $i = 1,2,\ldots,n$ e $j = 1,2,\ldots,n$. Empregando a Fórmula de Diferenças Central[4] encontramos o seguinte problema discreto associado

$$u_{i,1} = u_{1,j} = 0, \tag{1.32}$$

$$\frac{1}{h^2}u_{i,j-1} + \frac{1}{h^2}u_{i-1,j} - \frac{4}{h^2}u_{i,j} + \frac{1}{h^2}u_{i,j+1} + \frac{1}{h^2}u_{i+1,j} = -\operatorname{sen}(x_i)\operatorname{sen}(y_i), \tag{1.33}$$

$$u_{i,n} = u_{n,j} = 0 \tag{1.34}$$

a) Use a função `scipy.linalg.solve_banded` para computar a solução do problema discreto associado para diferentes tamanhos de malha $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$. Compute o erro da solução discreta em relação à solução analítica. Compare as aproximações com a solução analítica

$$u(x,y) = \frac{1}{2}\operatorname{sen}(x)\operatorname{sen}(y). \tag{1.35}$$

b) Compare a demanda de tempo e memória computacional se a função `scipy.linalg.solve` for empregada na computação da solução discreta.

[3] Baron Siméon Denis Poisson, 1781 - 1840, matemático, engenheiro e físico francês. Fonte: Wikipedia.

[4] Consulte mais em Notas de Aula: Matemática Numérica.

Matrizes bloco

E 1.1.4. Seja o seguinte PVC

$$-u'' + v(x) = f(x), \quad 0 < x < 1 \tag{1.36}$$
$$-v'' - u(x) = g(x), \quad 0 < x < 1 \tag{1.37}$$

$$u(0) = u(1) = 0 \tag{1.38}$$
$$v(0) = v(1) = 0 \tag{1.39}$$

Vamos considerar sua formulação discreta pelo Método de Diferenças Finitas em uma malha uniforme

$$x_i = (i-1)h, \tag{1.40}$$

com $i = 1,2,\ldots,n$, com tamanho de malha $h = 1/(n-1)$. Usando a fórmula de diferenças central de ordem 2 nas EDOs, obtemos

$$-\left(\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}\right) + v_i = f_i \tag{1.41}$$
$$-\left(\frac{v_{i-1} - 2v_i + v_{i+1}}{h^2}\right) - u_i = g_i, \tag{1.42}$$

sendo, $u_i \approx u(x_i)$, $v_i \approx v(x_i)$, $f_i = f(x_i)$, $g_i = g(x_i)$ com $i = 2,\ldots,n-1$. Considerando as condições iniciais, tomamos $u_1 = u_n = v_1 = v_n = 0$ e multiplicando por h^2, temos

$$2u_2 - u_3 + h^2 v_2 = h^2 f_2 \tag{1.43}$$
$$-u_{i-1} + 2u_i - u_{i+1} + h^2 v_i = h^2 f_i \tag{1.44}$$
$$-u_{n-2} + 2u_{n-1} + h^2 v_{n-1} = h^2 f_{n-1} \tag{1.45}$$

$$2v_2 - v_3 - u_2 = h^2 g_2 \tag{1.46}$$
$$-v_{i-1} + 2v_i - v_{i-1} - h^2 u_i = h^2 g_i \tag{1.47}$$
$$-v_{n-2} + 2v_{n-1} - h^2 u_{n-1} = h^2 g_{n-1} \tag{1.48}$$
$$\tag{1.49}$$

onde, $i = 3,\ldots,n-2$.

As equações acima formam um sistema linear $2(n-2)\times 2(n-2)$. Podemos escrever sua forma matricial

$$Aw = h \tag{1.50}$$

de várias formas a depender da escolha do vetor w em função de u e v. Vamos aqui considerar duas delas:

F1. $w = (u_2, v_2, \ldots, u_{n-1}, v_{n-1})$

$$u_i = w_{2(i-1)-1} \tag{1.51}$$
$$f_i = h_{2(i-1)-1} \tag{1.52}$$

$$v_i = w_{2(i-1)} \tag{1.53}$$
$$g_i = h_{2(i-1)} \tag{1.54}$$

F2. $w = (u_2, \ldots, u_{n-1}, v_2, \ldots, v_{n-1})$

$$u_i = w_{i-1} \tag{1.55}$$
$$f_i = h_{i-1} \tag{1.56}$$

$$v_i = w_{i+n-3} \tag{1.57}$$
$$g_i = h_{i+n-3} \tag{1.58}$$

Referências Bibliográficas

[1] J.P. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Academic Press, Inc., San Diego, 2. edition, 1984.

[2] C.T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, 1999.

[3] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[4] J. Stoer. *Approximate Calculation of Multiple Integrals*. Prentice-Hall, Inc., 1971.

[5] D. Watkins. *Fundamentals of Matrix Computations*. John Wiley, New York, 2002.