Aula prática 2
Algoritmos de resolução de sistemas lineares

Will Sena*

Contents

*wnsena@protonmail.com

Algoritmos Iterativos

1)

Implementar uma função Scilab resolvendo um sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ usando o algoritmo iterativo de Jacobi. A função deve ter como variáveis de entrada:

- **a matriz A;**
- **o vetor $\boldsymbol{b}$;**
- **uma aproximação inicial a $\boldsymbol{x}_0$ da solução do sistema;**
- **uma tolerância E;**
- **um número máximo de iterações M;**
- **o tipo de norma a ser utilizada: 1, 2 ou $\%inf$.**

Como variáveis de saída:

- **a solução $\boldsymbol{x}_k$ do sistema encontrada pelo método;**
- **a norma da diferença entre as duas últimas aproximações ($||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - 1||$);**
- **o número k de iterações efetuadas;**
- **a norma do resíduo ($||r_k|| = ||\boldsymbol{b} - A\boldsymbol{x}_k||$).**

Critério de parada do algoritmo: use "$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - 1|| < E$ ou $k > M$".

```
function [xk, e, k, rk] = jacobi(A, b, x0, E, M, Norm)
  xk = x0; e = %inf; k = 0; rk = 0;
  D = diag(diag(A)); LMU = A - D; invD = diag(1 ./ diag(D));
  MJ = - invD * LMU; cJ = invD * b;

  while e > E & k < M
    xk1 = MJ * xk + cJ;
    e = norm(xk1 - xk, Norm);
    xk = xk1;
    k = k + 1;
```

```
  end

  rk = norm(A * xk - b, Norm);
endfunction
```

2)

Implementar uma função Scilab resolvendo um sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ usando o algoritmo iterativo de Gauss-Seidel. A função deve ter como variáveis de entrada:

- a matriz A;
- o vetor $\boldsymbol{b}$;
- uma aproximação inicial $\boldsymbol{x}_0$ da solução do sistema;
- uma tolerância E;
- um número máximo de iterações M;
- o tipo de norma a ser utilizada: 1, 2 ou $\%inf$.

Como variáveis de saída:

- a solução $\boldsymbol{x}_k$ do sistema encontrada pelo método;
- a norma da diferença entre as duas últimas aproximações $(||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - 1||)$;
- o número k de iterações efetuadas;
- a norma do resíduo $(||r_k|| = ||\boldsymbol{b} - A\boldsymbol{x}_k||)$.

Critério de parada do algoritmo: use "$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - 1|| < E$ ou $k > M$".

Faça duas implementações diferentes: uma usando a função "inv" do Scilab para calcular a inversa de $L + D$, obtendo assim a matriz do método $M_G = -(L + D)^{-1}U$ e o vetor $\boldsymbol{c}_G = (L + D)^{-1}\boldsymbol{b}$ para fazer as iterações $\boldsymbol{x}_{k+1} = M_G * \boldsymbol{x}_k + \boldsymbol{c}_G$ e outra resolvendo o sistema linear $(L + D) * \boldsymbol{x}_{k+1} = -U * \boldsymbol{x}_k + \boldsymbol{b}$ para fazer as iterações (a matriz $L+D$ é triangular inferior; escreva uma função para resolver

sistemas em que a matriz dos coeficientes é triangular inferior e use-a a cada iteração).

Primeira implementação:

```
function [xk, e, k, rk] = gauss_seidel_1(A, b, x0, E, M, Norm)
  xk = x0; e = %inf; k = 0; rk = 0;
  invLMD = inv(tril(A));
  MG = - invLMD * triu(A, 1); cG = invLMD * b;

  while e > E & k < M
    xk1 = MG * xk + cG;
    e = norm(xk1 - xk, Norm);
    xk = xk1;
    k = k + 1;
  end

  rk = norm(A * xk - b, Norm);
endfunction
```

Segunda implementação:

```
function [x] = solve_Lb(L, b)
  n = size(b, 1); x = b; x(1) = x(1) / L(1,1);
  for i = 2:n
    x(i) = (x(i) - L(i, 1:i-1) * x(1:i-1)) / L(i, i) ;
  end
endfunction
```

```
function [xk, e, k, rk] = gauss_seidel_2(A, b, x0, E, M, Norm)
  xk = x0; e = %inf; k = 0; rk = 0;
  D = diag(diag(A)); LMD = tril(A); U = triu(A, 1);

  while e > E & k < M
    xk1 = solve_Lb(LMD, -U*xk + b);
    e = norm(xk1 - xk, Norm);
    xk = xk1;
    k = k + 1;
```

```
  end

  rk = norm(A * xk - b, Norm);
endfunction
```

3)

Teste as funções implementadas para resolver o sistema

$$\begin{cases} x & - & 4y & + & 2z & = & 2 \\ & & 2y & + & 4z & = & 1 \\ 6x & - & y & - & 2z & = & 1 \end{cases}$$

Use o vetor $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ como aproximação inicial.

```
--> A = [0  2 4; 1 -4 2; 6 -1 -2];

--> b = [2; 1; 1];

--> x0 = [0; 0; 0];

--> [xk, e, k, rk] = jacobi(A, b, x0, 10e-6, 25, 2)
xk  =

Nan
-0.25
-0.5

e  =

Nan

k  =

1.

rk  =

Nan
```

```
--> [xk, e, k, rk] = gauss_seidel_1(A, b, x0, 10e-6, 25, 2)
at ... .

inv: Problem is singular.

--> [xk, e, k, rk] = gauss_seidel_2(A, b, x0, 10e-6, 25, 2)
xk  =

Inf
Inf
Nan

e  =

Nan

k  =

1.

rk  =

Nan
```

Nenhuma das funções o sistema converge, retornam assim resultados falhos.

Agora reordene as equações do sistema dado, de modo que a matriz dos coeficientes seja estritamente diagonal dominante e teste novamente as funções implementadas. Comente os resultados.

```
--> A = [6 -1 -2; 1 -4 2; 0 2 4];

--> b = [1; 2; 1];

--> x0 = [0; 0; 0];
```

```
--> [xk, e, k, rk] = jacobi(A, b, x0, 10e-6, 25, 2)
xk  =

0.2500019
-0.2499972
0.3750002

e  =

0.0000072

k  =

17.

rk  =

0.0000138

--> [xk, e, k, rk] = gauss_seidel_1(A, b, x0, 10e-6, 25, 2)
xk  =

0.2500000
-0.2499992
0.3749996

e  =

0.0000044

k  =

10.

rk  =

0.0000040

--> [xk, e, k, rk] = gauss_seidel_2(A, b, x0, 10e-6, 25, 2)
xk  =
```

```
0.25
-0.2499992
0.3749996

e  =

0.0000044

k  =

10.

rk  =

0.0000040
```

As funções de Gauss Seidel tiveram melhores resultados ($rk = 0.000004$) e menores iterações (10) em relação a função de Jacobi ($rk = 0.0000138$ e 17 iterações).

4)

- **a) Para o sistema do exercício 3 da Lista de Exercícios 2, mostre que o método de Jacobi com $\boldsymbol{x}_0 = \mathbf{0}$ falha em dar uma boa aproximação após 25 iterações.**

O sistema é: $\begin{cases} 2x_1 - x_2 + x_3 = -1 \\ 2x_1 + 2x_2 + 2x_3 = 4 \\ -x_1 - x_2 + 2x_3 = -5 \end{cases}$

A solução é: $\boldsymbol{x}^* = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$

```
--> A = [2 -1 1; 2 2 2; -1 -1 2];

--> b = [-1; 4; -5];

--> x0 = [0; 0; 0];
```

```
--> [xk, e, k, rk] = jacobi(A, b, x0, 10e-6, 25, 2)
xk  =

-20.827873
2.
-22.827873

e  =

48.219347

k  =

25.

rk  =

111.30075
```

A função não convergiu.

- **b) Use o método de Gauss-Seidel com $\boldsymbol{x}_0 = \mathbf{0}$ para obter uma aproximação da solução do sistema linear com precisão de 10^{-5} na norma-infinito.**

```
--> A = [2 -1 1; 2 2 2; -1 -1 2];

--> b = [-1; 4; -5];

--> x0 = [0; 0; 0];

--> [xk, e, k, rk] = gauss_seidel_1(A, b, x0, 10e-6, 25, %inf)
xk  =

1.0000023
1.9999975
-1.0000001

e  =
```

```
0.0000073

k  =

23.

rk  =

0.0000069
```

Uma boa aproximação.

5)

- **a. Utilize o método iterativo de Gauss-Seidel para obter uma aproximação da solução do sistema linear do exercício** 5 **da Lista de Exercícios 2, com tolerância de** 10^{-2} **e o máximo de** 300 **iterações.**

O sistema é: $\begin{cases} x_1 & & - & x_3 & = & 0.2 \\ -\frac{1}{2}x_1 & + & x_2 & - & \frac{1}{4}x_3 & = & -1.425 \\ x_1 & - & \frac{1}{2}x_2 & + & x_3 & = & 2 \end{cases}$

A solução é: $\boldsymbol{x}^* = \begin{bmatrix} 0.9 \\ -0.8 \\ 0.7 \end{bmatrix}$

```
--> A = [1 0 -1; -1/2 1 -1/4; 1 -1/2 1];

--> b = [0.2; -1.425; 2];

--> x0 = [0; 0; 0];

--> [xk, e, k, rk] = gauss_seidel_1(A, b, x0, 10e-3, 300, 2)
xk  =

0.8975131
-0.8018652
0.7015543
```

```
e  =

0.0090364
k  =

13.
rk  =

0.0041656
```

Uma boa aproximação.

- **b. O que acontece ao repetir o item a) quando o sistema é alterado para:**

$$\begin{cases} x_1 & & - & 2x_3 & = & 0.2 \\ -\frac{1}{2}x_1 & + & x_2 & - & \frac{1}{4}x_3 & = & -1.425 \\ x_1 & - & \frac{1}{2}x_2 & + & x_3 & = & 2 \end{cases}$$

```
--> A = [1 0 -2; -1/2 1 -1/4; 1 -1/2 1];

--> b = [0.2; -1.425; 2];

--> x0 = [0; 0; 0];

--> [xk, e, k, rk] = gauss_seidel_1(A, b, x0, 10e-3, 300, 2)
xk  =

2.157D+41
1.348D+41
-1.483D+41
e  =

5.085D+41
k  =

300.
rk  =

5.162D+41
```

A função não convergiu.

6)

Agora gere matrizes $An \times n$ com diagonal estritamente dominante para $n = 10$, $n = 100$, $n = 1000$, $n = 2000$, ... bem como vetores $\boldsymbol{b}$ com dimensões compatíveis e resolva esses sistemas $A\boldsymbol{x} = \boldsymbol{b}$ pelo Método de Gauss-Seidel, usando as duas versões implementadas no item 2. Use as funções $tic()$ e $toc()$ do Scilab para medir os tempos de execução e compará-los.

Função para gerar matrizes $n \times n$ estritamente dominantes:

```
function [M] = generate_SDM(n)
  M = rand(n, n);
  for i = 1:n
    M(i, i) = sum(M(i, :)) + rand(1);
  end
endfunction
```

Função que retorna as proporções dos tempos de execução da primeira função de Gauss-Seidel em relação à segunda dado os ns mínimos e máximos e o intervalo:

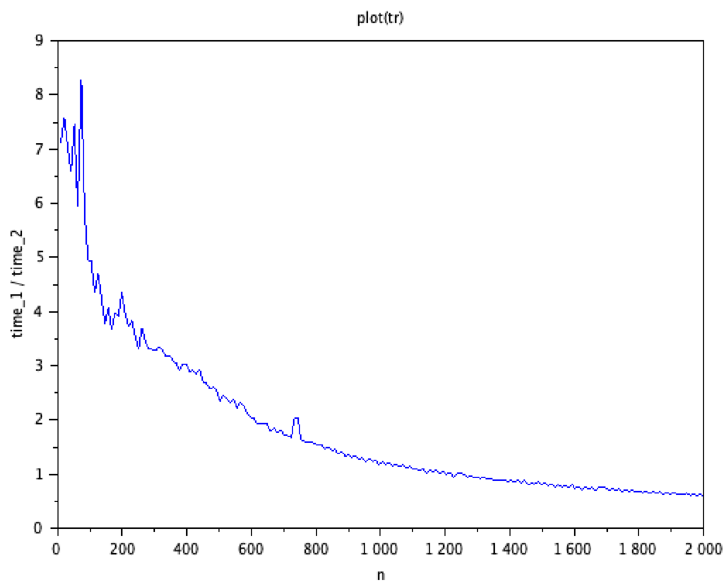
```
function [tr] = generate_time_ratios(n_min, n_max, step)
  tr = zeros(1, floor((n_max - n_min)/step)); i = 1;

  for n = n_min:step:n_max
    A = generate_SDM(n); b = rand(n, 1); x0 = rand(n, 1);
    tic(); gauss_seidel_1(A, b, x0, 10e-6, 100, 2); t1 = toc();
    tic(); gauss_seidel_2(A, b, x0, 10e-6, 100, 2); t2 = toc();
    tr(i) = t1 / t2;
    i = i + 1;
  end
endfunction
```

Obtendo essas proporções para n mínimo igual a 100, n máximo igual a 2000 e intervalo igual a 10:

```
tr = generate_time_ratios(100, 2000, 10);
```

Visualização gráfica deste vetor:



Para ns menores que, mais ou menos, 1200, a primeira função de Gauss Seidel é mais rápida que a segunda, provavelmente pela função *inv* do Scilab ser mais otimizada que a função *solve_Lb* implementada manualmente, mas para ns maiores a segunda função tende a ser cada vez mais rápida que a primeira, isto porque *solve_Lb* tem complexidade $O(n)$, enquanto uma função para calcular a inversa de uma matriz tem complexidade mínima $O(n^{2.373})$ (Optimized Coppersmith–Winograd algorithm).