

---

# Fundação Getúlio Vargas

Escola de Matemática Aplicada

Álgebra Linear Numérica

---

## **Algoritmos Iterativos de Resolução de Sistemas Lineares**

Aluno:

- Jeann da Rocha Silva

Professor:

- Antonio Carlos Saraiva Branco

Março  
2024

1) Implementar uma função Scilab resolvendo um sistema linear  $Ax = b$  usando o algoritmo iterativo de Jacobi. A função deve ter como variáveis de entrada:

- a matriz  $A$ ;
- o vetor  $b$ ;
- uma aproximação inicial  $x_0$  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  $x_k$  do sistema encontrada pelo método;
- a norma da diferença entre as duas últimas aproximações ( $\|x_k - x_{k-1}\|$ );
- o número  $k$  de iterações efetuadas;
- a norma do resíduo ( $\|r_k\| = \|b - Ax_k\|$ ).

**Critério de parada do algoritmo:** use " $\|x_k - x_{k-1}\| < E$  ou  $k > M$ ".

---

A função criada extrai as matrizes triangulares inferior ( $L$ ) e superior ( $U$ ) de  $A$  desconsiderando a diagonal e extrai também o vetor diagonal ( $D$ ) de  $A$ . Em seguida, cria a variável de iteração ( $k$ ) e o vetor da primeira iteração  $x_1 = D^{-1}(-(L + U)x_0 + b)$ , segundo o método de Jacobi. Após isso, é feita a iteração para  $k > 1$ , até que  $\|x_k - x_{k-1}\| < E$  (a tolerância deve ser admitida) ou  $k > M$  (o número de iterações deve ser, no máximo, o permitido). Vale ressaltar que no decorrer das iterações, alocamos o valor de  $x_{k-1}$  em  $x_0$  (para poupar a criação desnecessária de mais variáveis), a fim de salvar a penúltima iteração, e atualizamos o valor de  $x_{k-1}$  para  $x_k$ . Outro detalhe é que a fim de evitar tomar a matriz inversa da matriz diagonal obtida de  $D$  no Scilab, consideramos a operação de divisão coordenada a coordenada do vetor diagonal  $D$ , dada por  $./D$  no Scilab. Para mais detalhes, consulte o arquivo *Jacobi.sci* anexado.

2) Implementar uma função Scilab resolvendo um sistema linear  $Ax = b$  usando o algoritmo iterativo de Gauss-Seidel. A função deve ter como variáveis de entrada:

- a matriz  $A$ ;
- o vetor  $b$ ;
- uma aproximação inicial  $x_0$  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  $x_k$  do sistema encontrada pelo método;
- a norma da diferença entre as duas últimas aproximações ( $\|x_k - x_{k-1}\|$ );
- o número  $k$  de iterações efetuadas;
- a norma do resíduo ( $\|r_k\| = \|b - Ax_k\|$ ).

**Critério de parada do algoritmo:** use " $\|x_k - x_{k-1}\| < E$  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  $c_G = (L + D)^{-1}b$  para fazer as iterações  $x_{k+1} = M_G x_k + c_G$ ;
- outra resolvendo o sistema linear  $(L + D)x_{k+1} = -Ux_k + b$  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).

---

A construção dessas funções é inteiramente análoga a do item 1), bastando trocar a iteração  $x_k = D^{-1}(-(L + U)x_{k-1} + b)$  do método de Jacobi por  $x_k = L^{-1}(-Ux_{k-1} + b)$ , onde  $L$  é a matriz triangular de  $A$  com a diagonal e  $U$  é a mesma do item 1) (e não consideramos a matriz  $D$ ), no item (a) e, no caso do item (b), implementamos a função *Resolve\_Tri\_Inf* para resolver o sistema linear triangular inferior  $Lx_k = -Ux_{k-1} + b$  a fim de não necessitar calcular a inversa de  $L$ . Para mais detalhes, consultar os respectivos arquivos *GaussSeidel1.sci* e *GaussSeidel2.sci* anexados.

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  $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$  como aproximação inicial. 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.

---

Colocando a matriz  $A$  e os vetores  $b$  e  $x_0$  no Terminal, temos:

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

    1.   -4.    2.
    0.    2.    4.
    6.   -1.   -2.

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

    2.
    1.
    1.

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

    0.
    0.
    0.
```

Vamos considerar a tolerância razoavelmente pequena, dada por  $E = 10^{-5}$ . Vamos considerar também um número de iterações para testes de  $M = 10, 50, 100$ . Por fim, vamos utilizar a norma infinito e, não há muita diferença em qual (das 3) normas utilizar, já que elas são equivalentes e vale para vetores  $x \in \mathbb{R}^n$  que  $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq n \cdot \|x\|_\infty$ . Assim, obtemos os seguintes testes:

**- Teste 1 - M = 10 (Jacobi)**

```
--> [xk, dif_norm, k, rk_norm]=Jacobi(A, b, x0, 10^(-5), 10, %inf)
xk =

-52063.
-4594.5
-53676.75
dif_norm =

55974.25
k =

10.
rk_norm =

331956.13
```

**- Teste 2 - M = 50 (Jacobi)**

```
--> [xk, dif_norm, k, rk_norm]=Jacobi(A, b, x0, 10^(-5), 50, %inf)
xk =

-2.619D+24
-5.935D+24
1.117D+25
dif_norm =

8.201D+24
k =

50.
rk_norm =

6.322D+25
```

### - Teste 3 - M = 100 (Jacobi)

```
--> [xk, dif_norm, k, rk_norm]=Jacobi(A, b, x0, 10^(-5), 100, %inf)
xk =

    2.103D+50
    1.852D+50
   -1.818D+50
dif_norm =

    2.699D+50
k =

    100.
rk_norm =

    1.733D+51
```

Como se observa os valores para o resíduo aumentam cada vez mais rapidamente

$$(331956.13 \rightarrow 6.322 \cdot 10^{25} \rightarrow 1.733 \cdot 10^{51})$$

o que indica que o método é ineficaz para esta matriz. Analogamente, pelo Método de Gauss-Seidel, temos um resultado similar, como segue abaixo:

### - Teste 4 - M = 10 (Gauss-Seidel)

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^(-5), 10, %inf)
xk =

    2729633.5
    885950.
    7745925.
dif_norm =

    8188899.8
k =

    10.
rk_norm =

    35893782.
```

**- Teste 5 - M = 50 (Gauss-Seidel)**

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^(-5), 50, %inf)
xk =

-1.188D+34
-9.829D+33
-3.072D+34
dif_norm =

3.563D+34
k =

50.
rk_norm =

1.465D+35
```

**- Teste 6 - M = 100 (Gauss-Seidel)**

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^(-5), 100, %inf)
xk =

2.936D+67
-3.025D+68
2.393D+68
dif_norm =

3.855D+68
k =

100.
rk_norm =

1.754D+69
```

Isto decorre do fato de que a matriz em questão não tem diagonal estritamente dominante e, portanto, não temos a garantia da convergência dos métodos (isto será discutido melhor no item 4)). Entretanto, reordenando as equações, isto é, permutando as linhas de modo que a matriz se torne de diagonal estritamente dominante, temos a eficácia dos métodos, já que neste caso a matriz converge. Com efeito, a permutação em questão será a troca  $L_1 \leftrightarrow L_2$  e, em seguida,  $L_1 \leftrightarrow L_3$  e, portanto, teremos nesse caso  $\begin{bmatrix} 6 & -1 & -2 \\ 1 & -4 & 2 \\ 0 & 2 & 4 \end{bmatrix}$  e  $b = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$ . Os testes gerarão, agora, resultado satisfatório, para  $M = 18$  (em Jacobi) e  $M = 11$  (em Gauss-Seidel), e estes são os melhores valores, como se observa no número de iterações que o algoritmo realiza, como segue abaixo:

- Teste 7 -  $M = 18$  (Jacobi)

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

    6.   -1.   -2.
    1.   -4.    2.
    0.    2.    4.

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

    1.
    2.
    1.

--> [xk, dif_norm, k, rk_norm]=Jacobi(A, b, x0, 10^(-5), 18, %inf)
xk =

    0.2500019
   -0.2499972
    0.3750002
dif_norm =

    0.0000058
k =

    17.
rk_norm =

    0.0000138
```

Observe que o número de iterações, sendo igual a 18, efetua 17 iterações (valor de  $k$ ), o que indica que são necessárias somente 17, dado que a tolerância é  $10^{-5}$ . Analogamente, para o teste de Gauss-Seidel serão necessárias apenas 10 iterações e a justificativa disso se dá tomando  $M = 11$ , como consta abaixo:



**- Teste 8 - M = 11 (Gauss-Seidel)**

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^(-5), 11, %inf)
xk  =

    0.25
   -0.2499992
    0.3749996
dif_norm  =

    0.0000040
k  =

    10.
rk_norm  =

    0.0000040
```

4)

- a) Para o sistema do exercício 3 da Lista de Exercícios 2, mostre que o método de Jacobi com  $x^{(0)} = 0$  falha em dar uma boa aproximação após 25 iterações.
- b) Use o método de Gauss-Seidel com  $x^{(0)} = 0$  para obter uma aproximação da solução do sistema linear com precisão de  $10^{-5}$  na norma-infinito.
- 

- a) O método de Jacobi falha em 25 iterações como se vê abaixo (note que o resultado é diferente do que consta na Lista de Exercícios 2):

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

    2.    1.    1.
    2.    2.    2.
   -1.   -1.    2.

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

   -1.
    4.
   -5.

--> [xk, dif_norm, k, rk_norm]=Jacobi(A, b, x0, 10^(-5), 25, %inf)
xk =

   -3.0070616
    5.9954786
   -0.9690474
dif_norm =

    0.0525829
k =

    25.
rk_norm =

    0.0839805
```

Isso se deve porque a matriz do método possui raio espectral  $> 1$ , como consta abaixo, que como se sabe é suficiente para garantir que a matriz não é convergente e, portanto, que o método não é eficiente para esta matriz (a função utilizada "*Raio\_Jacobi.sci*" está anexada).

```
--> Raio_Jacobi(A)
ans =

1.1180340
```

- b) O método de Gauss-Seidel é eficaz e a aproximação tem precisão de  $10^{-5}$  em 23 iterações (note que o resultado é aproximadamente o que consta na Lista de Exercícios 2), como segue abaixo:

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^(-5), 25, %inf)
xk =

1.0000023
1.9999975
-1.0000001
dif_norm =

0.0000073
k =

23.
rk_norm =

0.0000070
```

Para fins de verificação, o raio espectral para a matriz do método é  $< 1$ , como consta abaixo (a função utilizada "*Raio\_GaussSeidel.sci*" está anexada).

```
--> Raio_GaussSeidel(A)
ans =

0.5
```

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.

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) Utilizando o Método de Gauss-Seidel, obtemos uma aproximação já nas 13 primeiras iterações com a norma infinito (note que o resultado é aproximadamente o que consta na Lista de Exercícios 2), como segue a seguir:

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

    1.    0.   -1.
   -0.5    1.  -0.25
    1.   -0.5    1.

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

    0.2
   -1.425
    2.

--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^-2, 300, %inf)
xk =

    0.8975131
   -0.8018652
    0.7015543
dif_norm =

    0.0064659
k =

    13.
rk_norm =

    0.0041656
```

Claramente, o raio espectral deve ser  $< 1$  e, isto é visto abaixo:

```
--> Raio_GaussSeidel(A)
ans =

    0.625
```

b) Repetindo o processo para esta nova matriz, encontramos um problema:

```
--> [xk, dif_norm, k, rk_norm]=GaussSeidel2(A, b, x0, 10^-2, 300, %inf)
xk =

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

    3.726D+41
k =

    300.
rk_norm =

    5.162D+41
```

cujo raio espectral é o seguinte:

```
--> Raio_GaussSeidel(A)
ans =

    1.375
```

O problema é que uma pequena perturbação em um dos coeficientes da matriz original pode ocasionar um problema na convergência caso a matriz esteja mal condicionada, que não é o caso, pois seu condicionamento é pequeno (próximo de 1), como feito abaixo:

```
--> cond(A)
ans =

    2.7337385
```

Entretanto, a perturbação não é necessariamente pequena, mas sim de  $-1$  unidade, o que de fato pode implicar na não convergência da matriz, independente do seu condicionamento, e, consequentemente, na falha do método de Gauss-Seidel para achar uma solução aproximada para o sistema.

6) Agora gere matrizes  $A_{n \times n}$  com diagonal estritamente dominante para  $n = 10$ ,  $n = 100$ ,  $n = 1000$ ,  $n = 2000$ , ... bem como vetores  $b$  com dimensões compatíveis e resolva esses sistemas  $Ax = b$  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.

---

Para  $n \in \mathbb{N}$ , vamos gerar uma matriz aleatória (com valores entre 0 e 1)  $n \times n$  e somar  $n$  a todos os elementos da sua diagonal, a fim de obter uma matriz de diagonal estritamente dominante. Efetuando os testes para as funções *Gauss-Seidel1.sci* e *Gauss-Seidel2.sci*, com a função *Compare\_GaussSeidel.sci* implementada para isso (cujo arquivo está anexado), temos

- **Teste 1** -  $n = 10$

```
--> [time1, time2]=Compare_GaussSeidel(10, 10^-5, 100, %inf)
time1 =

    0.0001881
time2 =

    0.0007913
```

- **Teste 2** -  $n = 100$

```
--> [time1, time2]=Compare_GaussSeidel(100, 10^-5, 100, %inf)
time1 =

    0.0092207
time2 =

    0.005933
```

- **Teste 3** -  $n = 1000$

```
--> [time1, time2]=Compare_GaussSeidel(1000, 10^-5, 100, %inf)
time1 =

    0.1734295
time2 =

    0.1442969
```

- **Teste 4** -  $n = 2000$

```
--> [time1, time2]=Compare_GaussSeidel(2000, 10^-5, 100, %inf)
time1 =

    2.4804098
time2 =

    0.5153299
```

- **Teste 5** -  $n = 4000$

```
--> [time1, time2]=Compare_GaussSeidel(4000, 10^-5, 100, %inf)
time1 =

   29.849049
time2 =

    1.8231598
```

- **Teste 6** -  $n = 8000$

```
--> [time1, time2]=Compare_GaussSeidel(8000, 10^-5, 100, %inf)
time1 =

  270.16021
time2 =

    9.2404664
```

- **Teste 7** -  $n = 16000$

```
--> [time1, time2]=Compare_GaussSeidel(16000, 10^-5, 100, %inf)
time1 =

  2284.1396
time2 =

   43.182047
```

Observando os tempos time1 (que representa o método com a função inv do Scilab implementada) e time2 (que representa o método com a solução do sistema triangular implementado), vemos que para o primeiro teste ( $n = 10$ ), a função inv tem melhor eficiência que não calcular a matriz inversa, mas a diferença de valores é muito pequena e a dimensão da matriz também. Ao aumentarmos a dimensão da matriz para 100, 1000, 2000, 4000, ..., a função inv passa a ser demasiadamente lenta em relação a outra implementação, o que justifica a não utilização disso. Eu poderia ter testado para  $n > 16000$ , mas não quero correr o risco de dar tela azul no PC devido ao excesso de memória

Processos		Executar nova tarefa	Finalizar tarefa	Modo de eficiência		
Nome	Status	39% CPU	78% Memória	14% Disco	0% Rede	
Aplicativos (4)						
> Gerenciador de Tarefas		0,5%	65,0 MB	0,1 MB/s	0 Mbps	
> Google Chrome (14)		0,1%	314,8 MB	3,6 MB/s	0 Mbps	
> Scilab 2024.0.0 (Desktop) (2)		37,3%	2.205,4 MB	0,1 MB/s	0 Mbps	
> Windows Explorer		0,1%	29,1 MB	0,5 MB/s	0 Mbps	

Figura 1: **DON'T INVERT THIS MATRIX**



Figura 2: **Breaking the code!**