Álgebra Linear - Aula Prática 2

Iara Cristina Mescua Castro

3 de abril de 2022

"Não use a função inv do Scilab, ao menos que seja pedido. -Branco, 2022"

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 x0 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 ($||xk–xk-1||$);
   · o número k de iterações efetuadas;
   · a norma do resíduo ($||rk|| = ||b–Ax_k||$).
   Critério de parada do algoritmo: use "$||xk–xk-1|| < E$ ou $k > M$".

   Resolução:

   Para a função Jacobi, primeiro vamos separar a matriz A em 3 matrizes: L, D e U.
   L sendo a matriz inferior de A, D a matriz diagonal de A e U a matriz superior de A, sendo que A = L + D + U. Então, a partir de $Ax = b$, teremos:

$$(L + U + D)x = b$$
$$Dx = b - (L + U)x$$

   Multiplicando a equação por D^{-1}:

$$x = -D^{-1}(L + U)x + D^{-1}b$$

   Vamos definir $-D^{-1}(L + U)$ como a matriz de Jacobi Mj e $D^{-1}b$ como a constante de Jacobi Cj.

   Para as iterações: $x_k = -D^{-1}(L+U)x_k + D^{-1}b$. O processo se repetirá em um loop até que o número de iterações chegue ao máximo M, ou a norma da diferença entre as duas últimas aproximações seja menor que a tolerância E. Note que tratando-se de um método iterativo, nosso x* resultado nunca será exato, mas sendo extremamente próximo já será muito mais eficaz que a Eliminação Gaussiana para maiores matrizes.

```
function [x, k, normadif, normaresid] = Jacobi(A, b, x_old, M, E, p)

L = tril(A, -1); //matriz L
D = diag(diag(A)); //matriz D
U = triu(A,1); //matriz U

InvD = diag(1./diag(A)); //matriz inversa de D

Mj = -InvD * (L + U);
```

```
Cj = InvD * b;

//x_old = zeros(1,size(A,1));

normadif = norm((A*x_old - b),p);
N = size(A,1);
//itr = 0;

for k = 1:M //criterio de parada 1: Num max de iteracoes M
        for i = 1:N
                x = Mj*x_old + Cj; //Jacobi Formula
                normadif = norm((x - x_old),p);
                normaresid = norm((b - A*x),p)
        end

        //itr = itr+1;

        if normadif < E //criterio de parada 2:
        //Norma da diff entre as duas ultimas aproximacoes
                break
        end
        x_old = x;
end

        endfunction
```

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 x0 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 ($||xk–xk-1||$);
   · o número k de iterações efetuadas;
   · a norma do resíduo ($||rk|| = ||b–Ax_k||$).
   · 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 $x_(k+1) = M_G*x_k+C_G$ e outra resolvendo o sistema linear $(L+D)*x_k+1 = -U*x_k+b$ 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).

   Resolução:
   Primeira implementação:

   Para a primeira implementação, partiremos da mesma forma que na Jacobi, decompondo A em matrizes, L, D e U. A partir de $Ax = b$, teremos:

$$(L + U + D)x = b$$

$$(L + D)x = b - Ux$$

   Multiplicando a equação por $(L + D)^{-1}$

$$x = -(L+D)^{-1}Ux + (L+D)^{-1}b$$

Vamos definir $-(L+D)^{-1}U$ como a matriz de Gauss Mg e $(L+D)^{-1}b$ como a constante de Gauss Cg. Analogamente, para as iterações: $x_{k+1} = -(L+D)^{-1}Ux_k + (L+D)^{-1}b$. O processo se repetirá em um loop até que o número de iterações chegue ao máximo M, ou a norma da diferença entre as duas últimas aproximações seja menor que a tolerância E.

```
function [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, x_old, M, E, p)

//x_old: uma aproximacao inicial x0 da solucao do sistema
//E: Tolerancia
//n: Numero maximo de iteracoes M

L = tril(A, -1); //matriz L
D = diag(diag(A)); //matriz D
U = triu(A,1); //matriz U

InvLD = inv(L+D); //matriz inversa de L + D

Mg = -InvLD*U;
Cg = InvLD*b;

//x_old = zeros(1,size(A,1));

normadif = %inf; //Norma da diff entre Ax e b.
//N = size(A,1);
//itr = 0;

x(:,1) = x_old;

for k = 1:M //criterio de parada 1: Num maximo de iteracoes M
        x = Mg*x_old + Cg; // Gauss-Seidel formula
        normadif = norm((x - x_old),p); //
        normaresid = norm((b - A*x),p) // finding residue

        if normadif < E //criterio de parada 2:
                //Norma da diff entre as duas ultimas aproximacoes
                break
        end
        x_old = x;
end

endfunction
```

Segunda implementação:

Para a segunda implementação, também vamos decompor A em matrizes, L, D e U. A partir de $Ax = b$, teremos:

$$(L+U+D)x = b$$
$$(L+D)x = b - Ux$$

Mas dessa vez, não vamos calcular a inversa de $(L+D)$, e sim, resolver um novo sistema linear:

$$(L+D)x_k = B$$

Onde a solução x será x_{k+1}. Para isso, criamos outra função "ResolveL"no arquivo para resolver esse sistema linear a cada iteração, sabendo que $(L+D)$ uma matriz inferior.

```
function [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, x_old, M, E, p)

//x_old: uma aprox. inicial x0 da solucao do sistema
//E: Tolerancia
//n: Num. max. de iteracoes M
//p: tipo da norma (1, 2, 'inf')

L = tril(A, -1); //matriz L
D = diag(diag(A)); //matriz D
U = triu(A,1); //matriz U

LD = L + D;

//x_old = zeros(1,size(A,1));

N = size(A,1);
//itr = 0;

//(L+D)x = -Ux_k-1 + b
//Ax = b
x = x_old;
normadif = %inf;
normaresid = %inf;

for k = 1:M //criterio de parada 1: Num. max. de iteracoes M
        B = -U*x_old + b;
        x = resolveL(LD,B)

        normadif = norm((x - x_old),p); //norma da diff. entre as duas ultimas ap
        normaresid = norm((b - A*x),p); //norma residuo

        if normadif < E //criterio de parada 2:
        //Norma da diff entre as duas ultimas aprox.
                break
        end
        x_old = x;
end

endfunction


function x = resolveL(L,b)
n = size(L,1);
m = size(b,1);
x = zeros(m,1);
x(1) = b(1)/L(1,1);
for i=2:n
        x(i) = (b(i) - L(i,1:i-1)*x(1:i-1))/L(i,i);
end
endfunction
```

3. Teste as funções implementadas para resolver o sistema:

$$\begin{cases} x - 4y + 2z = 2 \\ 2y + 4z = 1 \\ 6x - y - 2z = 1 \end{cases} \tag{0.1}$$

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. **Resolução:**

```
--> A
 A  =

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

--> b
 b  =

   2.
   1.
   1.

--> [x, k, normadif, normaresid] = Jacobi(A, b, [0;0;0], 25, 10^(-5), 1)
 x  =

  -2.946D+12
  -1.285D+12
  -6.636D+11
 k  =

   25.
 normadif  =

   4.832D+12
 normaresid  =

   2.116D+13



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

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

--> b
 b  =

   2.
   1.
   1.

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 25, 10^(-5), 1)
 x  =

   5.548D+16
   6.966D+15
   1.630D+17
 k  =

   25.
 normadif  =

   2.262D+17
 normaresid  =

   1.019D+18
```

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 25, 10^(-5), 1)
 x  =

   5.548D+16
   6.966D+15
   1.630D+17
 k  =

   25.
 normadif  =

   2.262D+17
 normaresid  =

   1.019D+18
```

Testando a função nesse sistema, além de obtermos uma solução extremamente grande, podemos observar que tanto a norma da diferença entre as duas últimas aproximações quanto a norma do resíduo estão longe de estarem menores que a tolerância. Vendo o número de iterações utilizado, concluímos que ele foi o critério de parada e a solução está incorreta. Somado ao fato que a divisão direta do Scilab fornece outro resultado.

```
--> 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.

--> A\b
 ans  =

   0.25
  -0.2500000
   0.375
```

O motivo da solução estar incorreta é porque essa matriz não é convergente. Visto que a diagonal não é estritamente dominante.

De modo que a matriz dos coeficientes seja estritamente diagonal dominante, podemos apenas permutar as linhas 1 e 3, e depois 2 e 3, de tal modo que:

$$\begin{cases} 6x - y - 2z = 1 \\ x - 4y + 2z = 2 \\ 0x + 2y + 4z = 1 \end{cases} \tag{0.2}$$

Agora temos uma matriz A convergente, e poderemos utilizar nossos métodos iterativos:

```
--> A
 A  =

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

--> b
 b  =

   1.
   2.
   1.

--> [x, k, normadif, normaresid] = Jacobi(A, b, [0;0;0], 100, 10^(-5), 1)
 x  =

   0.2500005
  -0.2499994
   0.3749986
 k  =

   18.
 normadif  =

   0.0000052
 normaresid  =

   0.0000145



--> A
 A  =

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

--> b
 b  =

   1.
   2.
   1.

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 100, 10^(-5), 1)
 x  =

   0.2500000
  -0.2499992
   0.3749996
 k  =

   10.
 normadif  =

   0.0000060
 normaresid  =

   0.0000040
```

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 100, 10^(-5), 1)
 x  =

   0.25
  -0.2499992
   0.3749996
 k  =

   10.
 normadif  =

   0.0000060
 normaresid  =

   0.0000040
```

Podemos ainda observar que ao mudar E para 10^{-10}, temos um resultado mais próximo. Isso acontece pois o algoritmo terá um maior número de interações até a norma da diferença entre x_k e x_{k-1} ser o novo E. Já que diminuindo ele drasticamente, estará mais próximo da solução, aumentando mais ainda a convergência.

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 100, 10^(-10), 1)
 x  =

   0.2500000
  -0.2500000
   0.3750000
 k  =

   18.
 normadif  =

   9.095D-11
 normaresid  =

   6.063D-11

--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 100, 10^(-10), 1)
 x  =

   0.25
  -0.2500000
   0.3750000
 k  =

   18.
 normadif  =

   9.095D-11
 normaresid  =

   6.063D-11
```

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.

```
-->  [x, k, normadif, normaresid] = Jacobi(A, b, [0;0;0], 25, 10^(-5), 1)
 x  =

  -20.827873
   2.
  -22.827873
 k  =

   25.
 normadif  =

   78.580342
 normaresid  =

   174.62298
```

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.

Resolução:

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 25, 10^(-5), 'inf')
 x  =

   1.0000023
   1.9999975
  -1.0000001
 k  =

   23.
 normadif  =

   0.0000073
 normaresid  =

   0.0000069
```

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 25, 10^(-5), 'inf')
 x  =

   1.0000023
   1.9999975
  -1.0000001
 k  =

   23.
 normadif  =

   0.0000073
 normaresid  =

   0.0000069
```

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.

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 300, 10^(-2), 1)
 x  =

   0.9015543
  -0.7988343
   0.6990286
 k  =

   14.
 normadif  =

   0.0095979
 normaresid  =

   0.0031572
```

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 300, 10^(-2), 1)
 x  =

   0.9015543
  -0.7988343
   0.6990286
 k  =

   14.
 normadif  =

   0.0095979
 normaresid  =

   0.0031572
```

As 300 iterações não são necessárias. Em 14 iterações a norma da diferença entre os dois últimos resultados já é menor que 10^{-2}. Podemos ver no x de saída que o vetor resultado está bem próximo de $[0.9; -0.7; 0.6]$

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

$$\begin{cases} 1x - 2x_3 = 0,2 \\ -\frac{1}{2} + x_2 - \frac{1}{4}x_3 = -1,425 \\ 1x_1 - \frac{1}{2}x_2 + x_3 = 2 \end{cases} \tag{0.3}$$

Resolução:

```
--> A
 A  =

   1.    0.   -2.
  -0.5   1.   -0.25
   1.   -0.5   1.

--> b
 b  =

   0.2
  -1.425
   2.

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, [0;0;0], 300, 10^(-2), 1)
 x  =

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

   300.
 normadif  =

   8.615D+41
 normaresid  =

   5.763D+41
```

```
--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, [0;0;0], 300, 10^(-2), 1)
 x  =

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

   300.
 normadif  =

   8.615D+41
 normaresid  =

   5.763D+41
```

Os resultados estão incorretos. Com essa pequena mudança, a diagonal deixa de ser estritamente dominante, visto que $|1| > |0| + |2|$. Consequentemente a matriz não é mais convergente, sendo impossível aplicar os métodos iterativos.

```
--> A = [1 0 -2; -0.5, 1, -1/4; 1, -0.5, 1]
 A  =

    1.     0.    -2.
   -0.5    1.    -0.25
    1.    -0.5    1.

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

    0.2
   -1.425
    2.

--> A\b
 ans  =

    1.1578947
   -0.7263158
    0.4789474
```

6. Agora gere matrizes A_{nxn} 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.

Resolução:

Para essa questão, criei uma função chamada "*matriz_converg_aleat*()"que recebe a dimensão desejada para a matriz A e gera, uma matriz A aleatória com diagonal estritamente dominante, um vetor b com dimensão (n,1) e um vetor inicial nulo com dimensão (n,1).

```
function [A,b,v] = matriz_converg_aleat(n)

// formula para gerar um num. aleat. entre [a,b]
//   r = a + (b-a)*rand();

//matriz A com num. aleat. entre [1, n]
A = floor(1 + ((n-1)*rand(n,n,'uniform')))

//soma de cada linha de A
soma_linhas = sum(A,2);

for i = 1:n
//soma da linha i de A
```

```
        soma_linha = soma_linhas(i);
        //diagonal de A com num maiores que a soma de cada linha de A.
                A(i,i) = soma_linha + floor(1 + ((n-1)*rand(1,1,'uniform')));
    end

    b = floor(1 + ((n-1)*rand(n,1,'uniform')))
    v = zeros(n,1);
    endfunction
```

Ao fazer alguns testes com as funções em matriz 10 por 10, observamos que a *Gauss_Seidel_1* é geralmente mais rápida.

```
--> [A,b,v] = matriz_converg_aleat(10);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0004581

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0015011

--> [A,b,v] = matriz_converg_aleat(10);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0003604

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0015254

--> [A,b,v] = matriz_converg_aleat(10);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0003469

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.001512
```

Ao fazer alguns testes com as funções em matriz 100 por 100, observamos que a *Gauss_Seidel_1* é geralmente mais rápida.

```
--> [A,b,v] = matriz_converg_aleat(100);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0011033

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0171962

--> [A,b,v] = matriz_converg_aleat(100);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0012464

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0131091
```

Ao fazer alguns testes com as funções em matriz 1000 por 1000, observamos que a *Gauss_Seidel_1* é geralmente mais rápida.

```
--> [A,b,v] = matriz_converg_aleat(1000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0732292

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.2960484

--> [A,b,v] = matriz_converg_aleat(1000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0655208

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.3020648

--> [A,b,v] = matriz_converg_aleat(1000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.0644903

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.2731984
```

Então podemos cogitar que a *Gauss_Seidel_1* sempre seja mais rápida que a *Gauss_Seidel_2*, algo poderá mudar ao continuar com matrizes maiores?

```
--> [A,b,v] = matriz_converg_aleat(2000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.6927349

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   1.0621004

--> [A,b,v] = matriz_converg_aleat(2000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.5762546

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.9982868

--> [A,b,v] = matriz_converg_aleat(2000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.5868795

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   0.9950009
```

Fazendo testes com matrizes 2000 por 2000, note que a diferença de tempo entre elas já diminui. Um tempo que antes da *Gauss_Seidel_2* era cerca de 10 vezes maior, agora é apenas o dobro maior que a *Gauss_Seidel_1*.

Ao continuar os testes 5000 por 5000, concluímos que SIM, a função $Gauss\_Seidel_2$ passa a ser mais eficiente que a *Gauss_Seidel_1*. E agora seu tempo de execução cerca de metade da $Gauss\_Seidel_1$.

```
--> [A,b,v] = matriz_converg_aleat(5000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   10.06812

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   5.7335774

--> [A,b,v] = matriz_converg_aleat(5000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   9.8602955

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   5.6454244

--> [A,b,v] = matriz_converg_aleat(5000);

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   9.7368109

--> tic(); [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1); tl=toc()
 tl  =

   5.4114456
```

Bônus:

Sabemos que apesar de $Gauss\_Seidel_1$ e $Gauss\_Seidel_2$ terem tempos de execução diferentes, ambos levam o mesmo número de iterações para alcançar uma condição de parada. Mas o tipo de norma também influencia?

Testando os 3 tipos de norma para matrizes 100 por 100.

```
--> [A,b,v] = matriz_converg_aleat(100);

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 1);

--> k
 k  =

   9.

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 2);

--> k
 k  =

   7.

--> [x, k, normadif, normaresid] = Gauss_Seidel_1(A, b, v, 1000, 10^(-5), 'inf');

--> k
 k  =

   6.
```

Testando os 3 tipos de norma para matrizes 1000 por 1000.

```
-->  [A,b,v] = matriz_converg_aleat(1000);

--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1);

--> k
 k  =

   9.

-->  [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 2);

--> k
 k  =

   7.

-->  [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 'inf');

--> k
 k  =

   5.
```

Testando os 3 tipos de norma para matrizes 2000 por 2000.

```
-->  [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 'inf');

--> k
 k  =

   5.

-->  [A,b,v] = matriz_converg_aleat(2000);

--> [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 1);

--> k
 k  =

   9.

-->  [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 2);

--> k
 k  =

   6.

-->  [x, k, normadif, normaresid] = Gauss_Seidel_2(A, b, v, 1000, 10^(-5), 'inf');

--> k
 k  =

   4.
```

Pelo visto, a norma infinita requer um menor número de iterações, enquanto a norma 1 requer um maior número de iterações para chegar na mesma aproximação.