Relatório - Aula Prática 1

Gustavo Ramalho*

27 de Março de 2022

1 Questão 1

Implementar uma função Scilab resolvendo um sistema linear Ax = b usando o algoritmo iterativo de Jacobi.

```

function [xk, normadif, nk, normares]=Jacobi(A,b,x0,E,M,tiponorma)

    D = diag(diag(A));
    LU = A-D;
    invD = diag(1./diag(A));

    Mj=invD*(LU);
    Cj=invD*b;

    xk = x0;
    normadif = E;
    nk = 0;
    while normadif >= E && nk<M
        x0=xk;
        xk=-Mj*x0+Cj;
        dif = x0 - xk;
        normadif = norm(dif,tiponorma);
        nk = nk+1;
    end

    normares=norm(A*xk-b,tiponorma);

endfunction



```

Dado esse código, é necessário definirmos o que representa cada variável de entrada e saída:

Variáveis de entrada

- A: matriz A
- b: vetor b
- x0: vetor inicial utilizado
- E: tolerância
- M: Número máximo de iterações
- tiponorma: Tipo de norma a ser utilizado

*gustavoramalho384@gmail.com

Variáveis de saída

- xk: solução encontrada
- normadif: Norma da diferença entre as duas últimas aproximações
- normares: norma do resíduo
- nk: número k de iterações efetuadas

Como vimos em aula, transformei $Ax = b$ em $(L+D+U)x = b$, e, nesse sentido, fui fazendo as operações até poder chamar $D^{-1} \cdot LU$ de M_j e $D^{-1} \cdot b$ de C_j (tudo sem usar a função inv). Depois, usei o while para fazer as operações necessárias do método de Jacobi.

2 Questão 2

Implementar uma função Scilab resolvendo um sistema linear Ax = b usando o algoritmo iterativo de Gauss-Seidel.

2.1 1º algoritmo

```

function [xk, normadif, nk, normares]=Gauss_Seidel_1(A,b,x0,E,M,tiponorma)

    LD = tril(A);
    U = A-LD;
    invLD = inv(LD);

    Mgs=-(invLD*(U));
    Cgs=invLD*b;

    xk = x0;
    normadif = E;
    nk = 0;
    while normadif >= E && nk<M
        x0=xk;
        xk=-Mgs*x0+Cgs;
        dif = x0 - xk;
        normadif = norm(dif,tiponorma);
        nk = nk+1;
    end

    normares=norm(A*xk-b,tiponorma);

endfunction



```

As variáveis de saída e de entrada são as mesmas do item anterior, por isso não foi necessário repetí-las aqui. Como pedido no enunciado, nessa primeira implementação foi usada a função inv para calcularmos a inversa de $L + D$. Com isso, a matriz Mgs e o vetor Cgs foram obtidos, e conseguimos fazer as iterações. Da mesma forma que na questão 1, utilizei o while para conseguir fazer isso.

2.2 2º algoritmo

```
function [xk]=resolve(LD,x)
    n=size(x,1)
    xk = zeros(n,1);
    xk(1)= x(1)/LD(1,1);
    for i =2 : n
        xk(i)=(x(i)-LD(i,:)*xk)/LD(i,i);
    end
endfunction

function [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)

    L = tril(A);
    U = A-L;

    xk = x0;
    normadif = E;
    nk = 0;
    while normadif >= E && nk<M
        x0=xk;
        xmedio = -(U*x0) + b;
        xk=resolve(L,xmedio);
        dif = x0 - xk;
        normadif = norm(dif,tiponorma);
        nk = nk+1;
    end

    normares=norm(b - A*xk,tiponorma);
endfunction


```

Agora, criamos a função que resolve sistemas em que a matriz dos coeficientes é triangular inferior, assim podemos fazer as iterações nessa função.

3 Questão 3

Teste as funções implementadas para resolver o sistema $\begin{cases} x - 4y + 2z = 2 \\ 2y + 4z = 1 \\ 6x - y - 2z = 1 \end{cases}$ **com o vetor** $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Definimos tiponorma = 2, M = 100 e E = 10^{-5} neste item. Depois disso, testei a matriz e vetor propostos nos 3 códigos que fiz. Os resultados estão a seguir:

Figura 1: console 3.1

```
--> [xk, normadif, nk, normares]=Jacobi(A,b,x0,E,M,tiponorma)
 xk  =

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

   3.359D+50
 nk  =

   100.
 normares  =

   1.733D+51
```

Figura 2: console 3.2

```
--> [xk, normadif, nk, normares]=Gauss_Seidel_1(A,b,x0,E,M,tiponorma)
 xk  =

   1.278D+68
   4.714D+68
   1.476D+68
 normadif  =

   3.970D+68
 nk  =

   100.
 normares  =

   2.119D+69
```

Figura 3: console 3.3

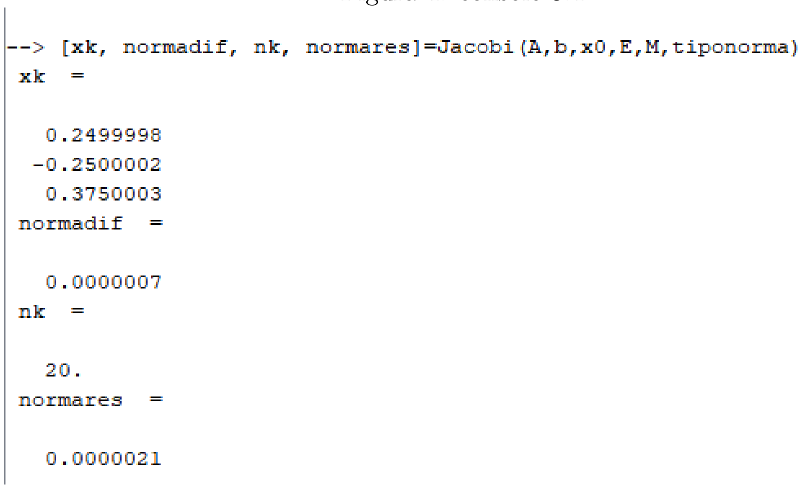
```
--> [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)
 xk  =

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

   3.970D+68
 nk  =

   100.
 normares  =

   1.754D+69
```

A norma das diferenças nos aponta que em nenhum dos métodos converge, pois ela é diferente e possui valores altos. Quando conferimos o raio espectral para cada um desses métodos, notamos que o módulo do resultado é maior que 1, o que nos diz que nao haverá a convergência. Agora, faremos com que essa matriz seja diagonalmente estritamente dominante, para conferirmos se vamos chegar em outra solução.

$$A_{DD} = \begin{bmatrix} 6 & -1 & -2 \\ 1 & -4 & 2 \\ 0 & 2 & 4 \end{bmatrix} \text{ e } b_{DD} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

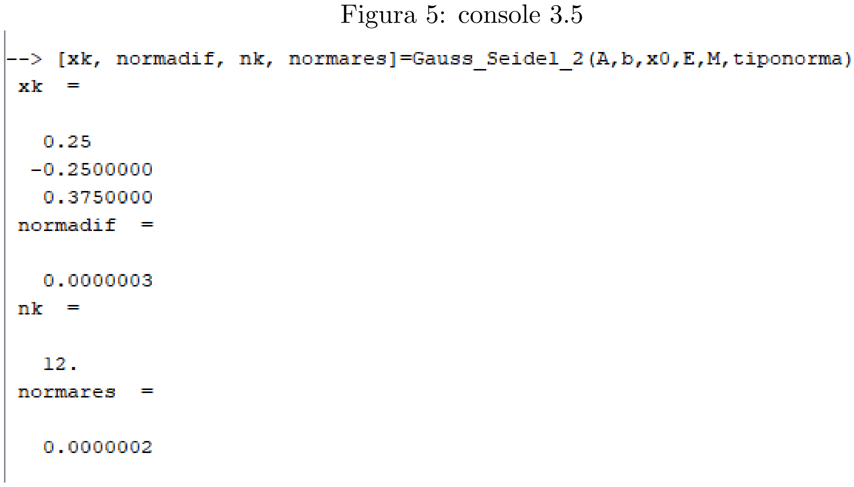
Alterei o E para 10^{-6}, para obter resultados ainda mais próximos. Abaixo, o que eu encontrei com o método de Jacobi e com Gauss-Seidel:

Figura 4: console 3.4



```
--> [xk, normadif, nk, normares]=Jacobi(A,b,x0,E,M,tiponorma)
 xk  =

   0.2499998
  -0.2500002
   0.3750003
 normadif  =

   0.0000007
 nk  =

   20.
 normares  =

   0.0000021
```

Figura 5: console 3.5



```
--> [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)
 xk  =

   0.25
  -0.2500000
   0.3750000
 normadif  =

   0.0000003
 nk  =

   12.
 normares  =

   0.0000002
```

Chegamos em um resultado! Isso aconteceu porque, como dito em um teorema comentado em aula, se A tem diagonal estritamente dominante, então isso implica que o método de Jacobi converge! A mesma ideia acontece com Gauss Seidel, que é um modelo parecido com Jacobi. Como esperado, a norma das diferenças é um número extremamente pequeno, assim como a norma do resíduo. É interessante notar também que o método de GaussSeidel chega em um resultado consideravelmente mais rápido que o método de Jacobi (12 operações contra 20), devido ao funcionamento de Gauss Seidel.

4 Questão 4.1

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.

$$A2 = \begin{bmatrix} 2 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix} \text{ e } b2 = \begin{bmatrix} -1 \\ 4 \\ 5 \end{bmatrix}$$

Coloquei esses valores em A2 e B2 e testei com o método de Jacobi. O resultado está a seguir.

Figura 6: console 4.1

```
--> [xk, normadif, nk, normares]=Jacobi(A2,b2,x0,E,M,tiponorma)
 xk  =

  -20.827873
   2.
  -22.827873
 normadif  =

   34.924597
 nk  =

   25.
 normares  =

   87.311491
```

Conseguimos notar que o método não convergiu. As diagonais não são estritamente dominantes, então nós não tinhamos certeza que iria convergir. Quando calculamos o raio espectral, encontramos resultado 1.11, que é maior que 1. Logo, não converge.

5 Questão 4.2

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 seguir, testei a matriz com E^{-5}. Foi possível notar, pelas 23 operações realizadas, que o método converge. xk está convergindo para (1, -1, 1), e esse resultado é encontrado quando alteramos E para um valor maior, como testei com o valor E^{-8} depois. Diferente da 4.1, que utilizamos o método de Jacobi, agora temos raio espectral igual a 0.5, que nos indica que o método converge.

Figura 7: console 4.2

```
--> [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)
 xk  =

   1.0000023
   1.9999975
  -1.0000001
 normadif  =

   0.0000073
 nk  =

   23.
 normares  =

   0.0000069
```

6 Questão 5.1

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.

Figura 8: console 5.1

```
--> [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)
 xk  =

   0.8975131
  -0.8018652
   0.7015543
 normadif  =

   0.0064659
 nk  =

   13.
 normares  =

   0.0040412

-->
```

É possível notar que o método convergiu, de acordo com os resultados que encontramos. Para verificarmos isso, calculamos o raio espectral, que nos deu valor 0.625, que é um número dentro do que esperavamos.

7 Questão 5.2

O que acontece ao repetir o item a) quando o sistema é alterado para $\begin{cases} x_1 - 2x_3 = 0.2 \\ -0.5x_1 + x_2 - 0.25x_3 = -1.425 \\ x_1 - 0.5x_2 + x_3 = 2 \end{cases}$

Figura 9: console 5.2

```
--> [xk, normadif, nk, normares]=Gauss_Seidel_2(A,b,x0,E,M,tiponorma)
 xk  =

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

   5.085D+41
 nk  =

   300.
 normares  =

   5.162D+41
```

Agora, nessa situação, notamos que o método não convergiu. Ao calcularmos novamente o raio espectral, encontramos nesse item o resultado 1.375, que nos diz que o método não converge, pois é maior que 1.

8 Questão 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.

```
function [tempo]=tempoexecucao(tam)
    b = ceil(10*rand(tam,1));
    A = ceil(10*rand(tam,tam));
    for j= 1 : tam
        A(j,j)=1+sum(abs(A(j,:)));
    end
    x=zeros(tam,1);
    tic();

    [xk, ndif, k, nres] = Gauss_Seidel_1(A,b,x,0.000001,300,2);
    t1 = toc();
    tic();
    [xk, ndif, k1, nres] = Gauss_Seidel_2(A,b,x,0.000001,300,2);
    t2 = toc();
    tempo =[t1,t2,k,k1];
endfunction

```

Gerei essa função para podermos calcular o tempo gasto para executar com os dois métodos, Gauss Seidel 1 e Gauss Seidel 2. A seguir, prints dos resultados, onde o 1º mostra o tempo com a função inv (Gauss Seidel 1) e o 2º mostra sem ela (Gauss Seidel 2), e, na frente, o número de iterações realizadas. O número de entrada mostra o tamanho da matriz quadrada.

Figura 10: console 6.1

```
--> A = tempoexecucao(100)
 A  =

   0.0020847   0.0179577   9.   9.

--> A = tempoexecucao(1000)
 A  =

   0.115101   0.325413   8.   8.

--> A = tempoexecucao(3000)
 A  =

   3.0613025   2.5951646   8.   8.

--> A = tempoexecucao(6000)
 A  =

   22.740992   7.6942623   8.   8.
```

É possível notar que, para matrizes pequenas, a função inv é viável e não gasta muito tempo/-memória. Entretanto, quando colocamos uma matriz 3000x3000 e posteriormente 6000x6000, o tempo de execução das duas funções é muito diferente, o que torna muito mais viável a utilização de Gauss Seidel 2. Depois disso, ainda testamos com matrizes maiores: 10.000 x 10.000 e 20.000 x 20.000:

Figura 11: console 6.2

```
--> A = tempoexecucao(10000)
 A  =

   91.585255   23.880435   7.   7.
```

Figura 12: console 6.3

```
--> A = tempoexecucao(20000)
 A  =

   744.11523   136.80405   7.   7.
```

Aqui, o tempo foi ainda maior. Na primeira tentativa, , com tempoexecucao(10000), Gauss Seidel 1 levou 1 minuto e meio, enquanto Gauss Seidel 2, apenas 23 segundos. Já em tempoexecucao(20000), A função com inv chegou a gastar mais de 12 minutos, enquanto a outra, pouco mais de 2 minutos! Além disso, também fiz o printscreen do meu gerenciador de tarefas no momento da execução de cada item, e o resultado foi um consumo muito grande da memória RAM (de 16 GB) nas duas operações, como esperado.

Figura 13: console 6.4