Relatório da Aula Prática 2

Sillas Rocha da Costa

5 de abril de 2024

Questão 01

Baseado no que foi passado em aula, a função do algoritmo iterativo de Jacobi obtida foi a seguinte, podendo ser visualizada no arquivo "jacobi.sci", na pasta "funcs":

```
function [x_k2, diff, k, r] = jacobi(A, b, x_0, E, M, norma)
    n = size(A, 1);
    // Inicializa a iteração anterior como a estimativa inicial
    x_k1 = x_0;
    // Temos (L + D + U) = A, onde LU = (L + U) e D_inv = D^-1
    LU = A - diag(diag(A));
    D_inv = diag(1 ./ diag(A));

    // Calcula a próxima iteração
    x_k2 = -D_inv * LU * x_k1 + D_inv * b;
    // Calcula a diferença entre a iteração atual e a anterior
    diff = norm(x_k2 - x_k1, norma);
    // Esta foi a primeira iteração
    k=1;

    // Loop que continua até que alguma das duas condições seja cunmprida
    while (k < M & diff > E) do
        // Atualiza a iteração anterior para a iteração atual
        x_k1 = x_k2;

        // Calcula a próxima iteração
        x_k2 = -D_inv * LU * x_k1 + D_inv * b;

        // Incrementa o contador de iterações
        k = k + 1;
        // Calcula a diferença entre as iterações
        diff = norm(x_k2 - x_k1, norma);
    end

    // Calcula a norma do resíduo
    r = norm(b - A*x_k2, norma);
endfunction
```

Questão 02

Alternativa a

A implementação do algoritmo iterativo de Gauss-Seidel utilizando a inversa pode ser encontrada no arquivo "gauss_seidel_inv.sci"na pasta "funcs". Segue o seu resultado:

```
function [x_k2, diff, k, r] = gauss_seidel_inv(A, b, x_0, E, M, norma)
    n = size(A, 1);
    x_k1 = x_0;
    // Calcula a matriz triangular inferior inversa
    LD_inv = inv(tril(A));
    // Obtém a parte triangular superior de A
    U = triu(A, 1);

    // Calcula a próxima iteração
    x_k2 = - LD_inv * U * x_k1 + LD_inv * b;
    // Calcula a diferença entre a iteração atual e a anterior
    diff = norm(x_k2 - x_k1, norma);
    k = 1;

    // Loop que continua até que alguma das duas condições seja cumprida
    while (k < M & diff > E) do
        // Atualiza a iteração anterior para a iteração atual
        x_k1 = x_k2;

        // Calcula a próxima iteração
        x_k2 = - LD_inv * U * x_k1 + LD_inv * b;

        // Atualiza o contador e a diferença
        k = k + 1;
        diff = norm(x_k2 - x_k1, norma);
    end

    // Calcula a norma do resíduo
    r = norm(b - A*x_k2, norma);
endfunction
```

Alternativa b

A implementação do algoritmo iterativo de Gauss-Seidel utilizando a resolução de um sistema linear pode ser encontrada no arquivo "gauss_seidel_lin.sci"na pasta "funcs". Segue o seu resultado:

```
function [x_k2, diff, k, r] = gauss_seidel_lin(A, b, x_0, E, M, norma)
    n = size(A, 1);
    x_k1 = x_0;
    x_k2 = zeros(x_0);
    // Calcula a matriz triangular inferior de A
    LD = tril(A);
    // Obtém a parte triangular superior de A
    U = triu(A, 1);

    // Calcula b - U * x_k1
    b_Ux = b - U * x_k1;

    // Calcula a primeira iteração de x_k2
    x_k2(1) = b_Ux(1)/LD(1,1)
    for i = 2:n
        x_k2(i) = (b_Ux(i) - LD(i, 1:i-1) * x_k2(1:i-1)) / LD(i,i);
    end

    // Calcula a diferença entre a iteração atual e a anterior
    diff = norm(x_k2 - x_k1, norma);
    k = 1;

    // Loop que continua até que alguma das duas condições seja cumprida
    while (k < M & diff > E) do
        // Atualiza a iteração anterior para a iteração atual
        x_k1 = x_k2;
        // Recalcula b - U * x_k1
        b_Ux = b - U * x_k1;

        // Calcula a próxima iteração de x_k2
        x_k2(1) = b_Ux(1)/LD(1,1)
        for i = 2:n
            x_k2(i) = (b_Ux(i) - LD(i, 1:i-1)*x_k2(1:i-1)) / LD(i,i);
        end

        // Incrementa o contador de iterações
        k = k + 1;
        // Calcula a diferença entre as iterações
        diff = norm(x_k2 - x_k1, norma);
    end

    // Calcula a norma do resíduo
    r = norm(b - A*x_k2, norma);
endfunction
```

Questão 03

Alternativa a

Agora, ao executar o arquivo "quest_3_1.sci", que testa as funções no sistema linear com as condições passadas, vemos que os resíduos explodem para o infinito, entretanto, ao reordenar a matrix para que ela seja estritamente diagonal dominante, com a matriz de permutação $P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, chegamos no sistema $PAx = Pb$, com a execução do arquivo "quest_3_2.sci", vemos que neste sistema os 3 métodos convergem para a solução $x = \begin{pmatrix} 0,25 \\ -0,25 \\ 0,375 \end{pmatrix}$, onde o algoritmo de jacobi converge após 20 iterações, e os dois de gauss-seidel, após 12.

```
-> E = 1e-6;

-> M = 100;

-> norma = 2;


-> [x_jacobi, diff_jacobi, k_jacobi, r_jacobi] = jacobi(PA, Pb, x_0, E, M, norma);

-> [x_gs_inv, diff_gs_inv, k_gs_inv, r_gs_inv] = gauss_seidel_inv(PA, Pb, x_0, E, M, n

-> [x_gs_lin, diff_gs_lin, k_gs_lin, r_gs_lin] = gauss_seidel_lin(PA, Pb, x_0, E, M, n


-> disp(k_jacobi, k_gs_inv, k_gs_lin)

  20.

  12.

  12.
```

Questão 04

Alternativa a

No exercício 3 da lista 2, temos o seguinte caso: $A = \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix}$ e $b = \begin{bmatrix} -1 \\ 4 \\ -5 \end{bmatrix}$ Deste modo, ao executar o arquivo "quest_4_a.sci", verificaremos que o resíduo (o quão próxima a solução encontrada se aproxima da real) é de 111 na norma 2, em outras palavras, não é uma boa solução. Temos também, que a matriz do método, $M = D^{-1}(L + U)$ possui autovalores maiores que 1, ou seja, não é convergente:

```
--> disp(r_jacobi)

   87.311491

--> D = diag(diag(A));

--> LU = A - D;

--> abs(spec(inv(D)*LU))
 ans  =

   1.1180340
   1.1180340
   1.267D-17
```

```
-> exec("quest_4_a.sci")

-> clear;


-> exec("./funcs/jacobi.sci");


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

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

-> x_0 = zeros(b);

-> E = 0;

-> M = 25;

-> norma = %inf;


-> [x_jacobi, diff_jacobi, k_jacobi, r_jacobi] = jacobi(A, b, x_0, E, M, norma);


-> disp(r_jacobi)

  87.311491
```

Alternativa b

Agora, ao executar o arquivo "quest_4_b.sci", chegamos ao resultado pela aproximação desejada, e antes das 25 iterações que o método de jacobi falhou em dar um bom resultado.

```
-> exec("./funcs/gauss_seidel_inv.sci");

-> exec("./funcs/gauss_seidel_lin.sci");


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

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

-> x_0 = zeros(b);

-> E = 10e-5;

-> M = 25;

-> norma = %inf;


-> [x_gs_inv, diff_gs_inv, k_gs_inv, r_gs_inv] = gauss_seidel_inv(A, b, x_0, E, M, norma

-> [x_gs_lin, diff_gs_lin, k_gs_lin, r_gs_lin] = gauss_seidel_lin(A, b, x_0, E, M, norma


-> disp(r_gs_inv)

  0.0000877

-> disp(r_gs_lin)

  0.0000877
```

Questão 05

Alternativa a

Neste exercício, temos o seguinte sistema: $A = \begin{bmatrix} 1 & 0 & -1 \\ -1/2 & 1 & -1/4 \\ 1 & -1/2 & 1 \end{bmatrix}$ e $b = \begin{bmatrix} 0.2 \\ -1.425 \\ 2 \end{bmatrix}$

Assim, ao executar o arquivo "quest_5_a.sci", obteremos os resultados que convergem para a solução do sistema, a seguir:

```
-> exec("./funcs/gauss_seidel_inv.sci");

-> exec("./funcs/gauss_seidel_lin.sci");


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

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

-> x_0 = [0; 0; 0];

-> E = 10e-2;

-> M = 300;

-> norma = 2;


-> [x_gs_inv, diff_gs_inv, k_gs_inv, r_gs_inv] = gauss_seidel_inv(A, b, x_0, E, M, 

-> [x_gs_lin, diff_gs_lin, k_gs_lin, r_gs_lin] = gauss_seidel_lin(A, b, x_0, E, M, 


-> disp(r_gs_inv, r_gs_lin)

  0.0436793

  0.0436793
```

Alternativa b

Agora, com a matriz $A = \begin{bmatrix} 1 & 0 & -2 \\ -1/2 & 1 & -1/4 \\ 1 & -1/2 & 1 \end{bmatrix}$, veremos que, ao executar o arquivo "quest_5_b.sci", teremos um resultado não convergente para a solução do sistema, isto se deve pois a matriz do método, a matriz $(L + D)^{-1}U$ possui autovalores maiores que 1, como veremos a seguir:

```
-> spec(inv(tril(A)) * triu(A,1))
ans  =

  0.    + 0.i
  0.    + 0.i
  1.375 + 0.i
```

Agora, a saída da execução do código:

```
-> exec("./funcs/gauss_seidel_inv.sci");

-> exec("./funcs/gauss_seidel_lin.sci");


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

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

-> x_0 = [0; 0; 0];

-> E = 10e-2;

-> M = 300;

-> norma = 2;


-> [x_gs_inv, diff_gs_inv, k_gs_inv, r_gs_inv] = gauss_seidel_inv(A, b, x_0, E, M,

-> [x_gs_lin, diff_gs_lin, k_gs_lin, r_gs_lin] = gauss_seidel_lin(A, b, x_0, E, M,


-> disp(r_gs_inv, r_gs_lin)

  5.162D+41

  5.162D+41
```

Questão 06

Para gerar as matrizes, a função "gera_mat"foi utilizada, e pode ser encontrada na pasta funcs, que gera matrizes de diagonal dominante, a partir delas, resolvemos os sistemas $Ax = b$ pelos dois métodos do item 2, ao executar o arquivo "quest_6.sci", o que resultará nos seguintes resultados:

No caso 10X10, vemos que a inversa é mais veloz que a solução linear:

```
-> disp("10X10 por iversa:", inv_10, "10X10 por linear:", lin_10)

 "10X10 por iversa:"

  0.0001341

 "10X10 por linear:"

  0.0003672
```

Isso segue no caso 100X100, com a inversa ganhando da solução linear:

```
--> disp("100X100 por iversa:", inv_100, "100X100 por linear:", lin_100)

  "100X100 por iversa:"

   0.0005637

  "100X100 por linear:"

   0.0023464
```

Entretanto, a situação muda para o caso 1000X1000, onde a solução linear sai na frente da inversa:

```
--> disp("1000X1000 por iversa:", inv_1000, "1000X1000 por linear:", lin_1000)

  "1000X1000 por iversa:"

   0.1116392

  "1000X1000 por linear:"

   0.0865092
```

E, essa situação continuará acontecendo de maneira cada vez mais evidente para os próximos casos, como o 2000X2000:

```
-> disp("2000X2000 por iversa:", inv_2000, "2000X2000 por linear:", lin_2000)

 "2000X2000 por iversa:"

  0.7881561

 "2000X2000 por linear:"

  0.1576958
```

Conclusões finais: Foi perceptível, ao longo das resoluções, que, tanto o método de Jacobi, quanto o de Gauss-Seidel, não são perfeitos em sempre encontrar a solução para o sistema, mas alguns destes problemas podem ser contornados com, por exemplo, uma permutação das linhas do sistema, além disso, foi perceptível que o método de Gauss-Seidel aparenta ser superior ao de Jacobi em certas ocasiões, tanto em achar a solução em menos iterações, quanto em com um resíduo final menor. Além disso, entre os dois métodos de Gauss-Seidel aplicados (com a inversa e a linear), vimos que, mesmo que, a inversa resolva sistemas menores que 1000X1000 de maneira mais rápida que a linear por exemplo, a linear acaba sendo superior para sistemas desta ordem e maiores.