Análise Numérica – Lista 3

Patrick Duarte Pimenta

Questão 1.

```
clc;

// Todo: parâmetros de entrada: n, A, b, Toler, IterMax (ordem,
matriz, vetor independente, tolerância e número maximo de iterações)

// Todo: Parâmetros de saída: x, Iter, Erro (vetor solução, número de
iterações e condição de erro)
function [x, Iter, Erro]=gauss_Seidel(n, A, b, Toler, IterMax)
    // Construção das matrizes para as iterações
    for i = 1 : n
        r = 1 / A(i, i)

        for j = 1 : n
            if i ~= j then
                A(i, j) = A(i, j) * r
            end
        end

        b(i) = b(i) * r
        x(i) = b(i)
    end

    // Interações de Gauss-Seidel
    for Iter = 1 : IterMax
        for i = 1 : n
            Soma = 0

            for j = 1 : n
                if i ~= j then
                    Soma = Soma + A(i, j) * x(j)
                end
            end

            v(i) = x(i)
            x(i) = b(i) - Soma
        end

        Norma1 = 0
        Norma2 = 0

        for i = 1 : n
```

```
            if abs(x(i) - v(i)) > Norma1 then
                Norma1 = abs(x(i) - v(i))
            end

            if abs(x(i)) > Norma2 then
                Norma2 = abs(x(i))
            end
        end

        DifMax = Norma1 / Norma2

        disp("Interação: " + string(Iter) + " - x: " + string(x) + "
- DifMax = " + string(DifMax))
        disp(string(DifMax) + " < " + string(Toler));

        if DifMax < Toler || Iter >= IterMax then
            disp("Convergência alcançada após " + string(Iter) + "
iterações.")
            break
        end
    end

    Erro = DifMax >= Toler
    // Variável lógica: Se verdadeiro há erro e se falso não há erro
endfunction

////////////////////////////////////////////////////////////////////
///////

function [x]=gaussPivotamentoParcial(A, b);
    n = length(b);
    // cria a matriz aumentada
    Ab = [A, b];

    // percorre as colunas da matriz aumentada
    for k = 1:n-1
        // encontra o pivô máximo
        [max_val, max_row] = max(abs(Ab(k:n, k)));
        max_row = max_row + k - 1;

        // realiza o pivotamento parcial, se necessário
        if max_row ~= k
            temp = Ab(k, :);
            Ab(k, :) = Ab(max_row, :);
            Ab(max_row, :) = temp;
        end

        // percorre as linhas abaixo do pivô
        for i = k+1:n
            m = Ab(i, k) / Ab(k, k);
```

```
            Ab(i, :) = Ab(i, :) - m * Ab(k, :);
        end

        //printf("\nEtapa: %d", k);
        //disp(Ab);
    end

    // resolve o sistema triangular superior
    x = zeros(n, 1);
    x(n) = Ab(n, n+1) / Ab(n, n);
    for i = n-1:-1:1
        x(i) = (Ab(i, n+1) - Ab(i, i+1:n) * x(i+1:n)) / Ab(i, i);
    end
endfunction

matHilbert = zeros(12, 12);
[linhas, colunas] = size(matHilbert);

////////////////////////////////////////////////////////////////////////
///////

//fórmula h(i, j) = 1 / (i + j - 1), i, j = 1, 2, 3, ..., 12.
// TODO: O primeiro índice de um vetor ou matriz é 1 e não 0
for i = 1 : linhas
    for j = 1 : colunas
        matHilbert(i, j) = 1 / (i + j - 1);
    end;
end;

b = [
    3.1048747;
    2.1815712;
    1.7528267;
    1.4860237;
    1.2984136;
    1.1571464;
    1.0459593;
    0.9556691;
    0.8806136;
    0.8170732;
    0.7624855;
    0.7150172;
]

Toler = 10e-5;
IterMax = 1000;

disp("MatrizHilbert: ");
disp(matHilbert);
disp("Vetor de termos independentes: ")
```

```
disp(b);

[x, Iter, Erro] =  gauss_Seidel(12, matHilbert, b, Toler, IterMax)

disp("====> gaussSeidel. x = ");
disp(x);

[x2] = gaussPivotamentoParcial(matHilbert, b);

disp("====> gaussPivotamentoParcial. x = ");
disp(x2);
```

Saída:

"====> gaussSeidel. x = "

1.0313883

0.8739265

0.5573862

1.8907112

1.6250500

0.9382502

0.4211850

0.2339658

0.3691949

0.7567763

1.3348445

2.0318409

"====> gaussPivotamentoParcial. x = "

31.666434

-3945.1917

125581.12

-1727272.3

12760352.

-56425488.

1.581D+08

-2.874D+08

3.383D+08

-2.486D+08

1.037D+08

-18727925.

Comparação: O método Gauss-Seidel por ser de caráter iterativo, significa que pode rodar um número de soluções aproximadas que são melhoradas conforme são executadas, por outro lado Gauss por Pivotamento Parcial só gera em número estabelecido de execuções. Portanto, Gaus-Seidel é mais preciso e eficaz no tratamento de soluções, enquanto Gauss por Pivotamento Parcial é limitado a apenas um número de operações e não traz respostas bem mais precisas

---

Questão 2.

a)

```
function [x, Iter, Erro]=gauss_Seidel(n, A, b, Toler, IterMax)
    // Construção das matrizes para as iterações
    for i = 1 : n
        r = 1 / A(i, i)

        for j = 1 : n
            if i ~= j then
                A(i, j) = A(i, j) * r
            end
        end

        b(i) = b(i) * r
        x(i) = b(i)
    end

    // Aproximação inicial
    x = [
    1.01;
    2.01;
    3.01;
    ]

    // Interações de Gauss-Seidel
    Iter = 1
    while (Iter < IterMax)
        for i = 1 : n
            Soma = 0

            for j = 1 : n
                if i ~= j then
                    Soma = Soma + A(i, j) * x(j)
                end
            end

            v(i) = x(i)
            x(i) = b(i) - Soma
        end

        Norma1 = 0
        Norma2 = 0

        for i = 1 : n
            if abs(x(i) - v(i)) > Norma1 then
                Norma1 = abs(x(i) - v(i))
            end
```

```
            if abs(x(i)) > Norma2 then
                Norma2 = abs(x(i))
            end
        end

        DifMax = Norma1 / Norma2

        //disp("Interação: " + string(Iter) + " - x: " + string(x) +
" - DifMax = " + string(DifMax))
        //disp(string(DifMax) + " < " + string(Toler));

        if DifMax < Toler || Iter >= IterMax then
            disp("Convergência alcançada após " + string(Iter) + "
iterações.")
            break
        end

        Iter = Iter + 1
    end

    Erro = DifMax >= Toler
    // Variável lógica: Se verdadeiro há erro e se falso não há erro
endfunction

////////////////////////////////////////////////////////////////

// Matriz e termos independentes 1
M1 = [
    3 -3 7;
    1 6 -1;
    10 -2 7;
];

b1 = [
    18;
    10;
    27;
]

M2 = [
    1 2 5;
    1 3 1;
    4 1 2;
];

b2 = [
    20;
    10;
    12;
]
```

```
disp("Matriz 1 aumentada:");
disp([M1, b1]);

disp("Matriz 2 aumentada:");
disp([M2, b2]);

disp("//////////////////////////////////////////");

disp("jacobi M1 - x = ");
[x1m1, Iter1, Erro1] = jacobi(3, M1, b1, 10e-5, 10)
disp(x1m1);

disp("gaussSeidel M1 - x = ");
[x2m1, Iter11, Erro11] = gauss_Seidel(3, M1, b1, 10e-3, 10)
disp(x2m1);

disp("//////////////////////////////////////////");

disp("jacobi M2 - x = ");
[x1m2, Iter2, Erro2] = jacobi(3, M2, b2, 10e-5, 10)
disp(x1m2)

disp("gaussSeidel M2 - x = ");
[x2m2, Iter22, Erro22] = gauss_Seidel(3, M2, b2, 10e-2, 5)
disp(x2m2);

disp("//////////////////////////////////////////");
```

Saída:

"Matriz 1 aumentada:"

3. -3. 7. 18.

1. 6. -1. 10.

10. -2. 7. 27.

"Matriz 2 aumentada:"

1. 2. 5. 20.

1. 3. 1. 10.

4. 1. 2. 12.

"//"

"jacobi M1 - x = "

"Convergência alcançada após 10 iterações."

4.0805481

2.0389338

5.2339529

"gaussSeidel M1 - x = "

"Convergência alcançada após 1 iterações."

0.9866667

2.0038889

3.0201587

"//"

"jacobi M2 - x = "

"Convergência alcançada após 10 iterações."

2716.0213

533.05186

2291.3439

"gaussSeidel M2 - x = "

"Convergência alcançada após 1 iterações."

0.9300000

2.02

3.1300000

Questão 3.

```
function [x]=iterativo(A, b, k)
    // Número de linhas
    n = size(A,1);

    // Identidade da matriz A
    I = eye(A);

    // Inicializando x
    x = zeros(n,1);

    // i percorre ao número de passos (k)
    for i=1:k
        x = ((I + A) * x) - b;
    end
```

```
endfunction

clc

A = [
    -1.1   0.1;
     0.3  -0.3
];

b = [
    1;
    0
];

disp("Matriz Ab = ");
disp([A, b]);

[x] = iterativo(A, b, 100)

disp("x = ")
disp(x)
```

Saída:

"Matriz Ab = "

-1.1 0.1 1.

 0.3 -0.3 0.

"x = "

-1.0000000

-1.0000000

Questão 4.

```
clc;

/*
O algoritmo recebe três argumentos de entrada: a matriz A, o número
máximo de iterações max_iter e a tolerância tol. A função retorna o
valor próprio máximo lambda, o vetor próprio correspondente x, e o
número de iterConv interações antigidas após a convergência (iterConv
).*/
function [lambda, x, iterConv]=metodo_das_potencias(A, max_iter, tol)
    n = size(A, 1);                         // Obtém o tamanho da
matriz A
    x = [1; 1; 1];                          // Vetor inicial não nulo
    x = x / norm(x);                        // Normaliza o vetor para
ter norma igual a 1

    for i = 1:max_iter                      // Loop para executar o
método das potências
        y = A * x;                          // Multiplica a matriz A
pelo vetor x para obter um novo vetor y
        lambda = x' * y;                    // Calcula o produto
interno entre x e y para obter o valor próprio
        x = y / norm(y);                    // Normaliza o vetor y
para ter norma igual a 1 e atribui ao vetor x

        if norm(A * x - lambda * x) < tol   // Verifica se a condição
de parada foi atingida
            break;
        end
    end

    // Armazena i
    iterConv = i

    // Retorna o valor próprio máximo e o vetor próprio
correspondente
    lambda = x' * (A * x);
    x = x / norm(x);
end

A = [
    2  3  1;
    0  3 -1;
    0  0  1
];

// Letra c:
```

```
/* A matriz triangular superior inversa é calculada utilizando a
função inv(triu(A)). Além disso, a multiplicação da matriz A pelo
vetor x é substituída pela multiplicação da matriz triangular
superior inversa L pelo vetor x. O restante do algoritmo é semelhante
ao método das potências.*/
function [lambda, x, iterConv]=metodo_iteracao_inversa(A, max_iter,
tol)
    n = size(A, 1);                 // Obtém o tamanho da matriz A
    x = [1; 1; 1];                  // Vetor inicial não nulo
    x = x / norm(x);                // Normaliza o vetor para ter
norma igual a 1
    L = inv(triu(A));               // Obtém a matriz triangular
superior inversa

    for i = 1:max_iter              // Loop para executar o método da
iteração inversa
        y = L * x;                  // Multiplica a matriz triangular
superior inversa pelo vetor x
        lambda = x' * y;            // Calcula o produto interno
entre x e y para obter o valor próprio
        x = y / norm(y);            // Normaliza o vetor y para ter
norma igual a 1 e atribui ao vetor x

        if norm(A * x - lambda * x) < tol   // Verifica se a condição
de parada foi atingida
            break;
        end
    end

    // Armazena i
    iterConv = i

    // Retorna o valor próprio máximo e o vetor próprio
correspondente
    lambda = x' * (A * x);
    x = x / norm(x);
end

// Letra a:
[lambda, x, iterConv] = metodo_das_potencias(A, 100, 10e-3)

disp("### Letra A ###");

disp("Auto valor lambda = ");
disp(lambda);

disp("x = ");
disp(x);
```

```
// Letra b:

disp("### Letra B ###");
disp("O método convergiu após " + string(iterConv) + " interações.");

// Letra c:
[lambda, x] = metodo_iteracao_inversa(A, 100, 10e-3);

disp("### Letra C ###");

disp("Auto valor lambda = ");
disp(lambda);

disp("x = ");
disp(x);

disp("O método convergiu após " + string(iterConv) + " interações.");
```

a)

"Auto valor lambda = "

2.9863116

"x = "

0.9501144

0.3119017

0.0000035

b)

"O método convergiu após 11 interações."

c)

"Auto valor lambda = "

0.9972477

"x = "

-0.9123749
0.1830917
0.3661276

"O método convergiu após 11 interações."