Facultad de Ciencias

# Tarea 2

## Análisis Númerico


Oscar Andrés Rosas Hernandez

Alarcón Alvarez Aylin

Laurrabaquio Rodríguez Miguel Salvador

Pahua Castro Jesús Miguel Ángel


Octubre 2018

# Índice

# 1. Problemas de Computadora

**Una nota importante es que al inicio de CADA script se incluyen los algoritmos, porfavor cambia la primera linea de cada script para que el path sea el correcto, porfavor.**

Esta linea:

```
getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
```

Para ejecutar cada uno basta con hacer algo como:

```
exec("/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/22a.sce", -1)
```

## 1.1. 22

Ejecuta los scripts que esta dentro de Code llamado:

- 22a.sce
- 22b.sce

Pero ya que esto es un pdf y no me puedo quejar del espacio que utilizo:

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;

A22 = [2, 4, -2; 4, 9, -3; -2, -1, 7];
b22 = [2; 8; 10];

disp("Ax = b")

disp("A:")
disp(A22)

disp("b:")
disp(b22)

disp("Solving...")

[x22] = GaussianElimination(A22, b22);

disp("x:")
disp(x22)

disp("Getting the solution: Ax")
disp(A22 * x22)

disp("Expected solution (b)")
disp(b22)

disp("Cheking the error (Ax - b)")
disp(A22 * x22 - b22)
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;

A22 = [2, 4, -2; 4, 9, -3; -2, -1, 7];
c22 = [4; 8; -6];

disp("Ax = c")

disp("A:")
disp(A22)

disp("c:")
disp(c22)

disp("Solving...")

[L22, U22] = LUDecomposition(A22);

disp("L:");
disp(L22)

disp("U:");
disp(U22)

disp("Solving Ly = c")
y22 = FowardSubstitution(L22, c22);

disp("y:")
disp(y22)

disp("Solving Ux = y")
x22 = BackwardSubstitution(U22, y22);

disp("x:")
disp(x22)

disp("Getting the solution: Ax")
disp(A22 * x22)

disp("Expected solution (c)")
disp(c22)

disp("Cheking the error (Ax - c)")
disp(A22 * x22 - c22)
```

## 1.2. 23

Ejecuta los scripts que esta dentro de Code llamado: 23.sce

En este código mostramos el resultado de la implementación del algoritmo que estima la condición, dicho algoritmo esta dentro del archivo Condicion.sci de la carpeta de algoritmos.

En script lo que hace es generar varias matrices aleatoriamente y muestra nuestra estimación y el valor calculado de verdad.

Pero ya que esto es un pdf y no me puedo quejar del espacio que utilizo:

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;
for i = (1 : 4)
    A23 = grand(8, 8, "unf", data23(i, 1), data23(i, 2));

    disp("Random Matrix");
    disp(A23);

    disp("Real condition");
    disp(cond(A23));

    disp("Estimated condition");
    disp(Condition(A23, 50));
end
```

## 1.3. 24

Ejecuta los scripts que esta dentro de Code llamado:

- 24ab.sce
- 24c.sce

Ahora, la parte a) nos pregunta que pasa cuando resolvemos la matriz:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

Y la respuesta es sencilla, pues se resuelve, :v

Nunca se hace el pivoteo y siempre es muy sencillo poner ceros debajo del pivote basta con sumar una fila con la otra, por lo tanto no entiendo muy bien este inciso.

Mientras que en el inciso b) lo que hicimos fue solucionar 5 sistemas con $\vec{b}$ elegidos aleatorios y mostrar su condición (estimada y calculada).

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;

A24 = [
    1 0 0 0 1;
    -1 1 0 0 1;
    -1 -1 1 0 1;
    -1 -1 -1 1 1;
    -1 -1 -1 -1 1;
]


for i = (1 : 5)

    b24 = grand(5, 1, "uin", -20, 20);

    disp("Ax = b")

    disp("A:")
    disp(A24)

    disp("b:")
    disp(b24)

    disp("Solving...")

    [x24] = PartialGaussianElimination(A24, b24);

    disp("x:")
    disp(x24)

    disp("Getting the solution: Ax")
    disp(A24 * x24)

    disp("Expected solution (b)")
    disp(b24)

    disp("Cheking the error (Ax - b)")
```

```
    disp(A24 * x24 - b24)

end

disp("Real condition of A")
disp(cond(A24))

disp("Estimated condition of A")
disp(Condition(A24, 50))
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;

A24 = [
    1 0 0 0 1;
    -1 1 0 0 1;
    -1 -1 1 0 1;
    -1 -1 -1 1 1;
    -1 -1 -1 -1 1;
];

[L, U, P, Q] = CompleteLUDecomposition(A24);

disp("L")
disp(L)

disp("U")
disp(U)

disp("P")
disp(P)

disp("Q")
disp(Q)

disp("L * U")
disp(L * U)

disp("P * A * Q")
disp(P * A24 * Q)

disp("Error: L * U - (P * A * Q)")
disp( (L * U) - (P * A24 * Q) )
```

## 1.4. 25

Pasa algo muy interesante en este problema, lo que hacemos es basicamente resolver el sistema

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\epsilon \\ 2 \end{bmatrix}$$

Este sistema el principio lo podemos resolver muy bonito, pero alrededor de $1 \times 10^{-8}$ resulta que la operación $1+\epsilon = 1$, por lo tanto el vector solución esta mal y tenemos pequeños errores, pero $1 \times 10^{-16}$ tambien $\epsilon = 0$, por lo que nuestro error ahora si se va por las nubes, nunca mejor dicho, aritmetica de punto flotante, esa locura.

Puedes verlo por ti mismo en 25.sce

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;


for k = (1 : 10)
    epsilon = 10 ** (-2 * k)

    A25 = [
        epsilon 1;
        1 1;
    ];

    b25 = [
        1 + epsilon;
        2;
    ];

    realX25 = [
        1;
        1;
    ]

    disp("A")
    disp(A25)

    disp("b")
    disp(b25)

    [x] = GaussianElimination(A25, b25);

    disp("Estimated Solution: A \tilde x")
    disp(A25 * x)

    disp("Real Solution: A x")
    disp(A25  * realX25)

    disp("Difference of Error: \tilde x - x")
    disp(x -  realX25)

    disp("Estimated Condition")
    disp( Condition(A25, 10) )

    disp("Real Condition")
    disp( cond(A25) )

end
```

## 1.5. 26

Para hacer este tuve primero que crear la matriz, en este caso la cargo desde un archivo, así que para comprobarlo porfavor, cambia el path del archivo a donde lo estes ejecutando.

Luego resolvemos, nota que al ser un algoritmo $O(n^3)$ toma su ratito, pero cuando lo resolvemos y hacemos la norma de la diferencia de nuestras soluciones.

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;
disp("Ax = b")

A26 =
    fscanfMat("/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/26.matrix"
b26 = zeros(100, 1);
b26(:,1) = 1;

disp("A:")
disp(A26)

disp("b:")
disp(b26)

disp("Solving...")

[L26, U26] = LUDecomposition(A26);

disp("L:");
disp(L26)

disp("U:");
disp(U26)

disp("Solving Ly = b")
y26 = FowardSubstitution(L26, b26);

disp("y:")
disp(y26)

disp("Solving Ux = y")
x26 = BackwardSubstitution(U26, y26);

disp("x:")
disp(x26)

disp("Getting the solution: Ax")
disp(A26 * x26)

disp("Expected solution (b)")
disp(b26)

disp("Cheking the error (Ax - b)")
disp(A26 * x26 - b26)

disp("|A - L*U|:")
disp(Norm1(A26 - (L*U)))
```

## 1.6. 27

Esto estuvo bueno, porque tuvimos que hacer el mismo algoritmo el de Cholesky de dos maneras:

- La clasica que se basa en Gauss Jordan como $A = L * L^T$, esta se puede hacer muy sencilla y se encuentra en CholeskyGaussian.sci
- La otra se basa en primero hacer la clásica factorización L * U, y luego factorizar de U una matriz diagonal y una U que es triangular unitaria superior, es decir creamos $A = LDL^T$ y se encuentra en CholeskyBanachiewicz.sci

## 1.7. 28

Ejecuta los scripts que esta dentro de Code llamado:

- 28.sce

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
clc;


// ================ 1 ===============
A28 = [
    2  -1   0;
    -1  2  -1;
    0  -1   2;
];

disp("A")
disp(A28)

[L28] = CholeskyBanachiewicz(A28, 1)

disp("L")
disp(L28)

disp("L * L^t")
disp(L28 * L28')

[L28, D28] = CholeskyBanachiewicz(A28, 0)
disp("L")
disp(L28)

disp("D")
disp(D28)

disp("L * D * L^t")
disp(L28 * D28 * L28')


// ================ 2 ===============
A28 = [
    4   1   1   1;
    1   3  -1   1;
    1  -1   2   0;
    1   1   0   2;
];

disp("A")
disp(A28)

[L28] = CholeskyBanachiewicz(A28, 1)

disp("L")
disp(L28)

disp("L * L^t")
disp(L28 * L28')

[L28, D28] = CholeskyBanachiewicz(A28, 0)
disp("L")
disp(L28)

disp("D")
disp(D28)

disp("L * D * L^t")
disp(L28 * D28 * L28')


// ================ 3 ===============
A28 = [
    4   1  -1   0;
    1   3  -1   0;
   -1  -1   5   2;
    0   0   2   4;
];

```

```
disp("A")
disp(A28)

[L28] = CholeskyBanachiewicz(A28, 1)

disp("L")
disp(L28)

disp("L * L^t")
disp(L28 * L28')

[L28, D28] = CholeskyBanachiewicz(A28, 0)
disp("L")
disp(L28)

disp("D")
disp(D28)

disp("L * D * L^t")
disp(L28 * D28 * L28')


// ================ 4 ===============
A28 = [
     6   2   1  -1;
     2   4   1   0;
     1   1   4  -1;
    -1   0  -1   3;
];
disp("A")
disp(A28)

[L28] = CholeskyBanachiewicz(A28, 1)

disp("L")
disp(L28)

disp("L * L^t")
disp(L28 * L28')

[L28, D28] = CholeskyBanachiewicz(A28, 0)
disp("L")
disp(L28)

disp("D")
disp(D28)

disp("L * D * L^t")
disp(L28 * D28 * L28')
```

# 2. Anexo

## 2.1. BackwardSubstitution

```
// Solve a system Ux = y where U is an upper triangular
// using the famous algorithm backward substitution
// @param: U triangular superior matrix
// @param: b the b in Ux = b
// @return: x the solution vector

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [x] = BackwardSubstitution(U, b)
    [m, n] = size(U);
    x = zeros(n, 1);

    for i = (n : -1 : 1)
        if (U(i, i) == 0)
            error('Error: Singular matrix');
            return;
        end

        x(i) = b(i) / U(i,i);

        for j = (1 : i - 1)
            b(j) = b(j) - U(j, i) * x(i);
        end
    end
endfunction
```

## 2.2. CholeskyBanachiewicz

```
// Factor A as A = L * L^T
// using the famous algorithm called Cholesky using this really awesome propierty
// First A = L U then we make U a unit upper triangular matrix so we have L D L' and then
// we do L D2 D2 L' were D2(i, j) = sqrt(D(i, j)) finally we associate and we have A = L2 * L2'
// where L2 = L * D2
// @param: A a positive defined matrix (so A is symetric)
// @param: option if 1 then A = L * L else A = L * D * L
// @return: L lower triangule matrix

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [L, D] = CholeskyBanachiewicz(A, option)
    [m, n] = size(A);
    D = eye(n, n);
    L = eye(m, n);
    U = A;

    for step = (1 : n - 1)
        if (A(step, step) == 0)
            error('Error: Singular matrix');
            return;
        end

        for row = (step + 1 : n)
            L(row, step) = U(row, step) / U(step, step);
            for column = (1 : n)
                U(row, column) = U(row, column) - L(row, step) * U(step, column);
            end
        end
    end

    if option == 1
        for step = (1 : n)
            for row = (step : n)
                L(row, step) = L(row, step) * sqrt(U(step, step));
            end
        end
    else
        for step = (1 : n)
            D(step, step) = U(step, step);
        end
    end



endfunction
```

## 2.3. CholeskyGaussian

```
// Factor A as A = L * L^T
// using the famous algorithm called Cholesky using a modification of Gaussian Elimination
// @param: A a positive defined matrix (so A is symetric)
// @return: L lower triangule matrix

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [L] = CholeskyGaussian(A)

    [m, n] = size(A);
    L = zeros(m, n);

    for step = (1 : n)
        A(step, step) = sqrt( A(step, step) );

        for column = (step + 1 : n)
            A (column, step) = A(column, step) / A(step, step);
        end

        for i = (step + 1 : n)
            for j = (step + 1 : n)
                A(i, j) = A(i, j) - A(i, step) * A(j, step);
            end
        end
    end

    for row = (1 : n)
        for column = (1 : n)
            if (row >= column)
                L(row, column) = A(row, column);
            end
        end
    end

endfunction
```

## 2.4. CompleteLUDecomposition

```
// Factor A as PAQ = LU
// @param: A a not singular matrix
// @return: L (not sure) lower triangule matrix
// @return: U upper triangule matrix
// @return: P permutation matrix
// @return: Q permutation matrix

// JUST BECAUSE I WRITE IT, IT DOES NOT MEAN IT IS FAST, IT IS NOT FAST!

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [L, U, P, Q] = CompleteLUDecomposition(A)
    [m, n] = size(A);
    if (m ~= n) == 0 then
        error('Error: Not square matrix');
    end
    P = eye(n, n);
    Q = eye(n, n);
    L = eye(n, n);
    U = A;

    for step = (1 : n - 1)
        Qi = eye(n, n);
        Pi = eye(n, n);

        [maxIndex, index] = max(abs(A(step : n, step : n)));
        index(1) = index(1) + step - 1;
        index(2) = index(2) + step - 1;

        if (maxIndex == 0)
            error('Error: Singular matrix');
        end

        temporal = Pi(step, :);
        Pi(step, :) = Pi(index(1), :);
        Pi(index(1), :) = temporal;

        temporal = Qi(:, step);
        Qi(:, step) = Qi(:, index(2));
        Qi(:, index(2)) = temporal;

        temporal = U(step, :);
        U(step, :) = U(index(1), :);
        U(index(1), :) = temporal;

        temporal = U(:, step);
        U(:, step) = U(:, index(2));
        U(:, index(2)) = temporal;

        for row = (step + 1 : n)
            L(row, step) = U(row, step) / U(step, step);
            for column = (1 : n)
                U(row, column) = U(row, column) - L(row, step) * U(step, column);
            end
        end

        Q = Q * Qi;
        P = Pi * P;
    end

endfunction
```

## 2.5. Condition

```
// Estimates the condition of a Matrix: (|A||A^{-1}|)
// @param: A a not singular matrix
// @return: result the estimation

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [result] = Condition(A, numberOfTimes)
    [m, n] = size(A);
    if m ~= n
        error('Error: Not square matrix');
    end

    [normA] = Norm1(A);
    normAInverse = 0;

    [L, U, P] = PartialLUDecomposition(A);

    for i = (1 : numberOfTimes)

        randomC = zeros(n, 1);
        for i = (1 : n)
            if rand() < 0.5 then
                randomC(i) = 1;
            else
                randomC(i) = -1;
            end
        end

        v = FowardSubstitution(U', P * randomC);
        y = BackwardSubstitution(L', v);

        t = FowardSubstitution(L, P * y);
        z = BackwardSubstitution(U, t);

        temporal = Norm1(z) / Norm1(y);
        if temporal > normAInverse then
            normAInverse = temporal;
        end
    end

    result = normA * normAInverse;

endfunction
```

## 2.6. FowardSubstitution

```
// Solve a system Ly = b where L is triangular inferior
// using the famous algorithm foward substitution
// @param: L triangular inferior matrix
// @param: b the b in Ly = b
// @return: x the solution vector

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [y] = FowardSubstitution(L, b)
    [m, n] = size(L);
    y = zeros(n, 1);

    for i = (1 : n)
        if (L(i, i) == 0)
            error('Error: Singular matrix');
            return;
        end

        y(i) = b(i) / L(i, i);

        for j = (i + 1 : n)
            b(j) = b(j) - L(j, i) * y(i);
        end
    end
endfunction
```

## 2.7. GaussianElimination

```
// Solve Ax = b using Gaussian Elimination
// @param: A a not singular matrix
// @return: x such Ax = b

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [x] = GaussianElimination(A, b)
    [m, n] = size(A);
    U = A, B = b, x = zeros(n, 1)

    for step = (1 : n - 1)
        if (A(step, step) == 0) then
            error('Error: Singular matrix');
        end

        pivot = U(step, step);
        B(step) = B(step) / pivot
        for column = (1 : n)
            U(step, column) = U(step, column) / pivot;
        end

        for row = (step + 1 : n)
            rowPivot = U(row, step)
            B(row) = B(row) - rowPivot * B(step)
            for column = (1 : n)
                U(row, column) = U(row, column) - rowPivot * U(step, column);
            end
        end
    end

    for i = (n : -1 : 1)
        x(i) = B(i) / U(i,i);

        for j = (1 : i - 1)
            B(j) = B(j) - U(j, i) * x(i);
        end
    end

endfunction
```

## 2.8. LUDecomposition

```
// Factor A as A = L * U
// @param: A a not singular matrix
// @return: L lower triangule matrix
// @return: U upper triangule matrix

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [L, U] = LUDecomposition(A)
    [m, n] = size(A);
    L = eye(m, n);
    U = A;

    for step = (1 : n - 1)

        if (A(step, step) == 0)
            error('Error: Singular matrix');
            return;
        end

        for row = (step + 1 : n)
            L(row, step) = U(row, step) / U(step, step);

            for column = (1 : n)
                U(row, column) = U(row, column) - L(row, step) * U(step, column);
            end
        end
    end

endfunction
```

## 2.9. Norm1

```
// Get the norm 1 (max in columns)
// @param: A a matriz
// @return: r which is r = |A|

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [r] = Norm1(A)
    [m, n] = size(A);
    r = 0;

    for column = (1 : n)
        temporal = 0
        for row = (1 : m)
            temporal = temporal + abs(A(row, column))
        end
        if temporal > r
            r = temporal
        end
    end
endfunction
```

## 2.10. NormInfty

```
// Get the norm 1 (max in columns)
// @param: A a matriz
// @return: r which is r = |A|

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [r] = Norm1(A)
    [m, n] = size(A);
    r = 0;

    for column = (1 : n)
        temporal = 0
        for row = (1 : m)
            temporal = temporal + abs(A(row, column))
        end
        if temporal > r
            r = temporal
        end
    end
endfunction
```

## 2.11. PartialGaussianElimination

```
// Solve Ax = b using Gaussian Elimination
// @param: A a not singular matrix
// @return: x such Ax = b

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [x] = PartialGaussianElimination(A, b)
    [m, n] = size(A);
    U = A, B = b, x = zeros(n, 1)

    for step = (1 : n - 1)
        maxPivotRow = step;
        for posibility = (step + 1 : n)
            if abs(U(posibility, step)) > U(maxPivotRow, step)
                maxPivotRow = posibility
            end
        end

        if (A(maxPivotRow, step) == 0) then
            error('Error: Singular matrix');
        end

        if maxPivotRow ~= step then
            for element = (1 : n)
                temporal = U(step, element)
                U(step, element) = U(maxPivotRow, element)
                U(maxPivotRow, element) = temporal
            end

            temporal = B(maxPivotRow)
            B(maxPivotRow) = B(step)
            B(step) = temporal
        end

        pivot = U(step, step);
        B(step) = B(step) / pivot
        for column = (1 : n)
            U(step, column) = U(step, column) / pivot;
        end

        for row = (step + 1 : n)
            rowPivot = U(row, step)
            B(row) = B(row) - rowPivot * B(step)
            for column = (1 : n)
                U(row, column) = U(row, column) - rowPivot * U(step, column);
            end
        end
    end

    for i = (n : -1 : 1)
        x(i) = B(i) / U(i,i);

        for j = (1 : i - 1)
            B(j) = B(j) - U(j, i) * x(i);
        end
    end

endfunction
```

## 2.12. PartialLUDecomposition

```
// Factor A as PA = LU
// @param: A a not singular matrix
// @return: L (not sure) lower triangule matrix
// @return: U upper triangule matrix
// @return: P permutation matrix

// To solve use PA = LU
// y = FS(PL, Pb)
// x = BS(U, y)

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

function [L, U, P] = PartialLUDecomposition(A)
    [m, n] = size(A);
    P = eye(n, n); L = eye(n, n); U = A;

    for step = (1 : n - 1)

        maxPivotRow = step;
        for posibility = (step + 1 : n)
            if abs(U(posibility, step)) > U(maxPivotRow, step)
                maxPivotRow = posibility
            end
        end

        if (A(maxPivotRow, step) == 0)
            error('Error: Singular matrix');
        end

        if maxPivotRow ~= step
            if step >= 2
                for column = (1 : step - 1)
                    temporal = L(step, column);
                    L(step, column) = L(maxPivotRow, column);
                    L(maxPivotRow, column) = temporal;
                end
            end

            for element = (1 : n)
                temporal = U(step, element)
                U(step, element) = U(maxPivotRow, element)
                U(maxPivotRow, element) = temporal

                temporal = P(step, element)
                P(step, element) = P(maxPivotRow, element)
                P(maxPivotRow, element) = temporal
            end
        end

        for row = (step + 1 : n)
            L(row, step) = U(row, step) / U(step, step);
            for column = (1 : n)
                U(row, column) = U(row, column) - L(row, step) * U(step, column);
            end
        end
    end

endfunction
```