

---

FACULTAD DE CIENCIAS

# Tarea 2

## ANÁLISIS NÚMÉRICO

Oscar Andrés Rosas Hernandez

Alarcón Alvarez Aylin

Laurrabaquio Rodríguez Miguel Salvador

Pahua Castro Jesús Miguel Ángel

Octubre 2018

# Índice

<b>1. Problemas de Computadora</b>	<b>2</b>
1.1. 22 . . . . .	2
1.2. 23 . . . . .	4
1.3. 24 . . . . .	5
1.4. 25 . . . . .	7
1.5. 26 . . . . .	8
1.6. 27 . . . . .	9
1.7. 28 . . . . .	10
 <b>2. Anexo</b>	 <b>12</b>
2.1. BackwardSubstitution . . . . .	12
2.2. CholeskyBanachiewicz . . . . .	13
2.3. CholeskyGaussian . . . . .	14
2.4. CompleteLUdecomposition . . . . .	15
2.5. Condition . . . . .	16
2.6. ForwardSubstitution . . . . .	16
2.7. GaussianElimination . . . . .	17
2.8. LUdecomposition . . . . .	17
2.9. Norm1 . . . . .	18
2.10. NormInf . . . . .	18
2.11. PartialGaussianElimination . . . . .	19
2.12. PartialLUdecomposition . . . . .	20

## 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:

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

Para ejecutar cada uno basta con hacer algo como:

```
1 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:

```
1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd( '/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms ' )
7 clc;
8
9 A22 = [2, 4, -2; 4, 9, -3; -2, -1, 7];
10 b22 = [2; 8; 10];
11
12 disp("Ax = b")
13
14 disp("A:")
15 disp(A22)
16
17 disp("b:")
18 disp(b22)
19
20 disp("Solving...")
21
22 [x22] = GaussianElimination(A22, b22);
23
24 disp("x:")
25 disp(x22)
26
27 disp("Getting the solution: Ax")
28 disp(A22 * x22)
29
30 disp("Expected solution (b)")
31 disp(b22)
32
33 disp("Cheking the error (Ax - b)")
34 disp(A22 * x22 - b22)
```

```
1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
7 clc;
8
9 A22 = [2, 4, -2; 4, 9, -3; -2, -1, 7];
10 c22 = [4; 8; -6];
11
12 disp("Ax = c")
13
14 disp("A:")
15 disp(A22)
16
17 disp("c:")
18 disp(c22)
19
20 disp("Solving...")
21
22 [L22, U22] = LUDecomposition(A22);
23
24 disp("L:");
25 disp(L22)
26
27 disp("U:");
28 disp(U22)
29
30 disp("Solving Ly = c")
31 y22 = FowardSubstitution(L22, c22);
32
33 disp("y:")
34 disp(y22)
35
36 disp("Solving Ux = y")
37 x22 = BackwardSubstitution(U22, y22);
38
39 disp("x:")
40 disp(x22)
41
42 disp("Getting the solution: Ax")
43 disp(A22 * x22)
44
45 disp("Expected solution (c)")
46 disp(c22)
47
48 disp("Cheking the error (Ax - c)")
49 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:

```
1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd( '/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms' )
7 clc;
8 for i = (1 : 4)
9     A23 = grand(8, 8, "unf", data23(i, 1), data23(i, 2));
10
11     disp("Random Matrix");
12     disp(A23);
13
14     disp("Real condition");
15     disp(cond(A23));
16
17     disp("Estimated condition");
18     disp(Condition(A23, 50));
19 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).

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(' /Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms ')
7 clc;
8
9 A24 = [
10     1 0 0 0 1;
11    -1 1 0 0 1;
12    -1 -1 1 0 1;
13    -1 -1 -1 1 1;
14    -1 -1 -1 -1 1;
15 ]
16
17
18 for i = (1 : 5)
19
20     b24 = grand(5, 1, "uin", -20, 20);
21
22     disp("Ax = b")
23
24     disp("A:")
25     disp(A24)
26
27     disp("b:")
28     disp(b24)
29
30     disp("Solving ...")
31
32     [x24] = PartialGaussianElimination(A24, b24);
33
34     disp("x:")
35     disp(x24)
36
37     disp("Getting the solution: Ax")
38     disp(A24 * x24)
39
40     disp("Expected solution (b)")
41     disp(b24)
42
43     disp("Cheking the error (Ax - b)")

```

```

44     disp(A24 * x24 - b24)
45
46 end
47
48 disp("Real condition of A")
49 disp(cond(A24))
50
51 disp("Estimated condition of A")
52 disp(Condition(A24, 50))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd(' /Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms ')
7 clc;
8
9 A24 = [
10     1 0 0 0 1;
11     -1 1 0 0 1;
12     -1 -1 1 0 1;
13     -1 -1 -1 1 1;
14     -1 -1 -1 -1 1;
15 ];
16
17 [L, U, P, Q] = CompleteLUdecomposition(A24);
18
19 disp("L")
20 disp(L)
21
22 disp("U")
23 disp(U)
24
25 disp("P")
26 disp(P)
27
28 disp("Q")
29 disp(Q)
30
31 disp("L * U")
32 disp(L * U)
33
34 disp("P * A * Q")
35 disp(P * A24 * Q)
36
37 disp("Error: L * U - (P * A * Q)")
38 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

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd( '/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms' )
7 clc;
8
9
10 for k = (1 : 10)
11     epsilon = 10 ** (-2 * k)
12
13     A25 = [
14         epsilon 1;
15         1 1;
16     ];
17
18     b25 = [
19         1 + epsilon;
20         2;
21     ];
22
23     realX25 = [
24         1;
25         1;
26     ]
27
28     disp("A")
29     disp(A25)
30
31     disp("b")
32     disp(b25)
33
34     [x] = GaussianElimination(A25, b25);
35
36     disp("Estimated Solution: A \tilde{x}")
37     disp(A25 * x)
38
39     disp("Real Solution: A x")
40     disp(A25 * realX25)
41
42     disp("Difference of Error: \tilde{x} - x")
43     disp(x - realX25)
44
45     disp("Estimated Condition")
46     disp( Condition(A25, 10) )
47
48     disp("Real Condition")
49     disp( cond(A25) )
50
51 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.

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd( '/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms' )
7 clc;
8 disp("Ax = b")
9
10 A26 =
11     fscanfMat( "/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/26.matrix" );
12 b26 = zeros(100, 1);
13 b26(:,1) = 1;
14
15 disp("A:")
16 disp(A26)
17
18 disp("b:")
19 disp(b26)
20
21 disp("Solving...")
22 [L26, U26] = LUDecomposition(A26);
23
24 disp("L:")
25 disp(L26)
26
27 disp("U:")
28 disp(U26)
29
30 disp("Solving Ly = b")
31 y26 = ForwardSubstitution(L26, b26);
32
33 disp("y:")
34 disp(y26)
35
36 disp("Solving Ux = y")
37 x26 = BackwardSubstitution(U26, y26);
38
39 disp("x:")
40 disp(x26)
41
42 disp("Getting the solution: Ax")
43 disp(A26 * x26)
44
45 disp("Expected solution (b)")
46 disp(b26)
47
48 disp("Cheking the error (Ax - b)")
49 disp(A26 * x26 - b26)
50
51 disp("||A - L*U||:")
52 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 clásica 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

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd( '/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms' )
7 clc;
8
9
10 // ===== 1 =====
11 A28 = [
12     2   -1   0;
13    -1   2  -1;
14     0  -1   2;
15 ];
16
17 disp("A")
18 disp(A28)
19
20 [L28] = CholeskyBanachiewicz(A28, 1)
21
22 disp("L")
23 disp(L28)
24
25 disp("L * L^t")
26 disp(L28 * L28')
27
28 [L28, D28] = CholeskyBanachiewicz(A28, 0)
29 disp("L")
30 disp(L28)
31
32 disp("D")
33 disp(D28)
34
35 disp("L * D * L^t")
36 disp(L28 * D28 * L28')
37
38
39 // ===== 2 =====
40 A28 = [
41     4   1   1   1;
42     1   3  -1   1;
43     1  -1   2   0;
44     1   1   0   2;
45 ];
46
47 disp("A")
48 disp(A28)
49
50 [L28] = CholeskyBanachiewicz(A28, 1)
51
52 disp("L")
53 disp(L28)
54
55 disp("L * L^t")
56 disp(L28 * L28')
57
58 [L28, D28] = CholeskyBanachiewicz(A28, 0)
59 disp("L")
60 disp(L28)
61
62 disp("D")
63 disp(D28)
64
65 disp("L * D * L^t")
66 disp(L28 * D28 * L28')
67
68
69 // ===== 3 =====
70 A28 = [
71     4   1  -1   0;
72     1   3  -1   0;
73    -1  -1   5   2;
74     0   0   2   4;
75 ];
76

```

```

77 disp("A")
78 disp(A28)
79
80 [L28] = CholeskyBanachiewicz(A28, 1)
81
82 disp("L")
83 disp(L28)
84
85 disp("L * L^t")
86 disp(L28 * L28')
87
88 [L28, D28] = CholeskyBanachiewicz(A28, 0)
89 disp("L")
90 disp(L28)
91
92 disp("D")
93 disp(D28)
94
95 disp("L * D * L^t")
96 disp(L28 * D28 * L28')
97
98
99 // ===== 4 =====
100 A28 = [
101     6   2   1  -1;
102     2   4   1   0;
103     1   1   4  -1;
104    -1   0  -1   3;
105 ];
106 disp("A")
107 disp(A28)
108
109 [L28] = CholeskyBanachiewicz(A28, 1)
110
111 disp("L")
112 disp(L28)
113
114 disp("L * L^t")
115 disp(L28 * L28')
116
117 [L28, D28] = CholeskyBanachiewicz(A28, 0)
118 disp("L")
119 disp(L28)
120
121 disp("D")
122 disp(D28)
123
124 disp("L * D * L^t")
125 disp(L28 * D28 * L28')

```

## 2. Anexo

### 2.1. BackwardSubstitution

```
1 // Solve a system  $Ux = y$  where  $U$  is an upper triangular
2 // using the famous algorithm backward substitution
3 // @param: U triangular superior matrix
4 // @param: b the b in  $Ux = b$ 
5 // @return: x the solution vector
6
7 // @Author: Rosas Hernandez Oscar Andres
8 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
9 // @Author: Laurrabaquio Rodríguez Miguel Salvador
10 // @Author: Pádua Castro Jesús Miguel Ángel
11
12 function [x] = BackwardSubstitution(U, b)
13     [m, n] = size(U);
14     x = zeros(n, 1);
15
16     for i = (n : -1 : 1)
17         if (U(i, i) == 0)
18             error('Error: Singular matrix');
19             return;
20         end
21
22         x(i) = b(i) / U(i, i);
23
24         for j = (1 : i - 1)
25             b(j) = b(j) - U(j, i) * x(i);
26         end
27     end
28 endfunction
```

## 2.2. CholeskyBanachiewicz

```

1 // Factor A as A = L * L^T
2 // using the famous algorithm called Cholesky using this really awesome property
3 // First A = L U then we make U a unit upper triangular matrix so we have L D L' and then
4 // we do L D2 D2 L' were D2(i, j) = sqrt(D(i, j)) finally we associate and we have A = L2 * L2'
5 // where L2 = L * D2
6 // @param: A a positive defined matrix (so A is symmetric)
7 // @param: option if 1 then A = L * L else A = L * D * L
8 // @return: L lower triangle matrix
9
10 // @Author: Rosas Hernandez Oscar Andres
11 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
12 // @Author: Laurrabaquio Rodríguez Miguel Salvador
13 // @Author: Pahua Castro Jesús Miguel Angel
14
15 function [L, D] = CholeskyBanachiewicz(A, option)
16     [m, n] = size(A);
17     D = eye(n, n);
18     L = eye(m, n);
19     U = A;
20
21     for step = (1 : n - 1)
22         if (A(step, step) == 0)
23             error('Error: Singular matrix');
24             return;
25         end
26
27         for row = (step + 1 : n)
28             L(row, step) = U(row, step) / U(step, step);
29             for column = (1 : n)
30                 U(row, column) = U(row, column) - L(row, step) * U(step, column);
31             end
32         end
33     end
34
35     if option == 1
36         for step = (1 : n)
37             for row = (step : n)
38                 L(row, step) = L(row, step) * sqrt(U(step, step));
39             end
40         end
41     else
42         for step = (1 : n)
43             D(step, step) = U(step, step);
44         end
45     end
46
47
48
49 endfunction

```

## 2.3. CholeskyGaussian

```

1 // Factor A as A = L * L^T
2 // using the famous algorithm called Cholesky using a modification of Gaussian Elimination
3 // @param: A a positive defined matrix (so A is symmetric)
4 // @return: L lower triangle matrix
5
6 // @Author: Rosas Hernandez Oscar Andres
7 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
8 // @Author: Laurrabaquio Rodríguez Miguel Salvador
9 // @Author: Pádua Castro Jesús Miguel Angel
10
11 function [L] = CholeskyGaussian(A)
12
13     [m, n] = size(A);
14     L = zeros(m, n);
15
16     for step = (1 : n)
17         A(step, step) = sqrt( A(step, step) );
18
19         for column = (step + 1 : n)
20             A (column, step) = A(column, step) / A(step, step);
21         end
22
23         for i = (step + 1 : n)
24             for j = (step + 1 : n)
25                 A(i, j) = A(i, j) - A(i, step) * A(j, step);
26             end
27         end
28     end
29
30     for row = (1 : n)
31         for column = (1 : n)
32             if (row >= column)
33                 L(row, column) = A(row, column);
34             end
35         end
36     end
37
38 endfunction

```

## 2.4. CompleteLUdecomposition

```

1 // Factor A as PAQ = LU
2 // @param: A a not singular matrix
3 // @return: L (not sure) lower triangle matrix
4 // @return: U upper triangle matrix
5 // @return: P permutation matrix
6 // @return: Q permutation matrix
7
8 // JUST BECAUSE I WRITE IT, IT DOES NOT MEAN IT IS FAST, IT IS NOT FAST!
9
10 // @Author: Rosas Hernandez Oscar Andres
11 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
12 // @Author: Laurrabaquio Rodríguez Miguel Salvador
13 // @Author: Pádua Castro Jesús Miguel Ángel
14
15 function [L, U, P, Q] = CompleteLUdecomposition(A)
16     [m, n] = size(A);
17     if (m ~= n) == 0 then
18         error('Error: Not square matrix');
19     end
20     P = eye(n, n);
21     Q = eye(n, n);
22     L = eye(n, n);
23     U = A;
24
25     for step = (1 : n - 1)
26         Qi = eye(n, n);
27         Pi = eye(n, n);
28
29         [maxIndex, index] = max(abs(A(step : n, step : n)));
30         index(1) = index(1) + step - 1;
31         index(2) = index(2) + step - 1;
32
33         if (maxIndex == 0)
34             error('Error: Singular matrix');
35         end
36
37         temporal = Pi(step, :);
38         Pi(step, :) = Pi(index(1), :);
39         Pi(index(1), :) = temporal;
40
41         temporal = Qi(:, step);
42         Qi(:, step) = Qi(:, index(2));
43         Qi(:, index(2)) = temporal;
44
45         temporal = U(step, :);
46         U(step, :) = U(index(1), :);
47         U(index(1), :) = temporal;
48
49         temporal = U(:, step);
50         U(:, step) = U(:, index(2));
51         U(:, index(2)) = temporal;
52
53         for row = (step + 1 : n)
54             L(row, step) = U(row, step) / U(step, step);
55             for column = (1 : n)
56                 U(row, column) = U(row, column) - L(row, step) * U(step, column);
57             end
58         end
59
60         Q = Q * Qi;
61         P = P * Pi;
62     end
63
64 endfunction

```



## 2.5. Condition

```

1 // Estimates the condition of a Matrix: ( $|A||A^{-1}|$ )
2 // @param: A a not singular matrix
3 // @return: result the estimation
4
5 // @Author: Rosas Hernandez Oscar Andres
6 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
7 // @Author: Laurrabaquio Rodríguez Miguel Salvador
8 // @Author: Pahua Castro Jesús Miguel Ángel
9
10 function [result] = Condition(A, numberOfTimes)
11     [m, n] = size(A);
12     if m ~= n
13         error('Error: Not square matrix');
14     end
15
16     [normA] = Norm1(A);
17     normAInverse = 0;
18
19     [L, U, P] = PartialLUdecomposition(A);
20
21     for i = (1 : numberOfTimes)
22
23         randomC = zeros(n, 1);
24         for i = (1 : n)
25             if rand() < 0.5 then
26                 randomC(i) = 1;
27             else
28                 randomC(i) = -1;
29             end
30         end
31
32         v = FowardSubstitution(U', P * randomC);
33         y = BackwardSubstitution(L', v);
34
35         t = FowardSubstitution(L, P * y);
36         z = BackwardSubstitution(U, t);
37
38         temporal = Norm1(z) / Norm1(y);
39         if temporal > normAInverse then
40             normAInverse = temporal;
41         end
42     end
43
44     result = normA * normAInverse;
45
46 endfunction

```

## 2.6. FowardSubstitution

```

1 // Solve a system  $Ly = b$  where L is triangular inferior
2 // using the famous algorithm foward substitution
3 // @param: L triangular inferior matrix
4 // @param: b the b in  $Ly = b$ 
5 // @return: x the solution vector
6
7 // @Author: Rosas Hernandez Oscar Andres
8 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
9 // @Author: Laurrabaquio Rodríguez Miguel Salvador
10 // @Author: Pahua Castro Jesús Miguel Ángel
11
12 function [y] = FowardSubstitution(L, b)
13     [m, n] = size(L);
14     y = zeros(n, 1);
15
16     for i = (1 : n)
17         if (L(i, i) == 0)
18             error('Error: Singular matrix');
19             return;
20         end
21
22         y(i) = b(i) / L(i, i);
23
24         for j = (i + 1 : n)
25             b(j) = b(j) - L(j, i) * y(i);
26         end
27     end
28 endfunction

```

## 2.7. GaussianElimination

```

1 // Solve Ax = b using Gaussian Elimination
2 // @param: A a not singular matrix
3 // @return: x such Ax = b
4
5 // @Author: Rosas Hernandez Oscar Andres
6 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
7 // @Author: Laurrabaquio Rodríguez Miguel Salvador
8 // @Author: Pahua Castro Jesús Miguel Ángel
9
10 function [x] = GaussianElimination(A, b)
11     [m, n] = size(A);
12     U = A, B = b, x = zeros(n, 1)
13
14     for step = (1 : n - 1)
15         if (A(step, step) == 0) then
16             error('Error: Singular matrix');
17         end
18
19         pivot = U(step, step);
20         B(step) = B(step) / pivot
21         for column = (1 : n)
22             U(step, column) = U(step, column) / pivot;
23         end
24
25         for row = (step + 1 : n)
26             rowPivot = U(row, step)
27             B(row) = B(row) - rowPivot * B(step)
28             for column = (1 : n)
29                 U(row, column) = U(row, column) - rowPivot * U(step, column);
30             end
31         end
32     end
33
34     for i = (n : -1 : 1)
35         x(i) = B(i) / U(i, i);
36
37         for j = (1 : i - 1)
38             B(j) = B(j) - U(j, i) * x(i);
39         end
40     end
41
42 endfunction

```

## 2.8. LUDecomposition

```

1 // Factor A as A = L * U
2 // @param: A a not singular matrix
3 // @return: L lower triangule matrix
4 // @return: U upper triangule matrix
5
6 // @Author: Rosas Hernandez Oscar Andres
7 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
8 // @Author: Laurrabaquio Rodríguez Miguel Salvador
9 // @Author: Pahua Castro Jesús Miguel Ángel
10
11 function [L, U] = LUDecomposition(A)
12     [m, n] = size(A);
13     L = eye(m, n);
14     U = A;
15
16     for step = (1 : n - 1)
17
18         if (A(step, step) == 0)
19             error('Error: Singular matrix');
20             return;
21         end
22
23         for row = (step + 1 : n)
24             L(row, step) = U(row, step) / U(step, step);
25
26             for column = (1 : n)
27                 U(row, column) = U(row, column) - L(row, step) * U(step, column);
28             end
29         end
30     end
31
32 endfunction

```

## 2.9. Norm1

```

1 // Get the norm 1 (max in columns)
2 // @param: A a matriz
3 // @return: r which is  $r = |A|$ 
4
5 // @Author: Rosas Hernandez Oscar Andres
6 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
7 // @Author: Laurrabaquio Rodríguez Miguel Salvador
8 // @Author: Pahlua Castro Jesús Miguel Angel
9
10 function [r] = Norm1(A)
11     [m, n] = size(A);
12     r = 0;
13
14     for column = (1 : n)
15         temporal = 0
16         for row = (1 : m)
17             temporal = temporal + abs(A(row, column))
18         end
19         if temporal > r
20             r = temporal
21         end
22     end
23 endfunction

```

## 2.10. NormInfy

```

1 // Get the norm 1 (max in columns)
2 // @param: A a matriz
3 // @return: r which is  $r = |A|$ 
4
5 // @Author: Rosas Hernandez Oscar Andres
6 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
7 // @Author: Laurrabaquio Rodríguez Miguel Salvador
8 // @Author: Pahlua Castro Jesús Miguel Angel
9
10 function [r] = Norm1(A)
11     [m, n] = size(A);
12     r = 0;
13
14     for column = (1 : n)
15         temporal = 0
16         for row = (1 : m)
17             temporal = temporal + abs(A(row, column))
18         end
19         if temporal > r
20             r = temporal
21         end
22     end
23 endfunction

```

## 2.11. PartialGaussianElimination

```

1 // Solve Ax = b using Gaussian Elimination
2 // @param: A a not singular matrix
3 // @return: x such Ax = b
4
5 // @Author: Rosas Hernandez Oscar Andres
6 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
7 // @Author: Laurrabaquio Rodríguez Miguel Salvador
8 // @Author: Pahuá Castro Jesús Miguel Angel
9
10 function [x] = PartialGaussianElimination(A, b)
11     [m, n] = size(A);
12     U = A, B = b, x = zeros(n, 1)
13
14     for step = (1 : n - 1)
15         maxPivotRow = step;
16         for posibilidad = (step + 1 : n)
17             if abs(U(posibilidad, step)) > U(maxPivotRow, step)
18                 maxPivotRow = posibilidad
19             end
20         end
21
22         if (A(maxPivotRow, step) == 0) then
23             error('Error: Singular matrix');
24         end
25
26         if maxPivotRow ~= step then
27             for element = (1 : n)
28                 temporal = U(step, element)
29                 U(step, element) = U(maxPivotRow, element)
30                 U(maxPivotRow, element) = temporal
31             end
32
33             temporal = B(maxPivotRow)
34             B(maxPivotRow) = B(step)
35             B(step) = temporal
36         end
37
38         pivot = U(step, step);
39         B(step) = B(step) / pivot
40         for column = (1 : n)
41             U(step, column) = U(step, column) / pivot;
42         end
43
44         for row = (step + 1 : n)
45             rowPivot = U(row, step)
46             B(row) = B(row) - rowPivot * B(step)
47             for column = (1 : n)
48                 U(row, column) = U(row, column) - rowPivot * U(step, column);
49             end
50         end
51     end
52
53     for i = (n : -1 : 1)
54         x(i) = B(i) / U(i, i);
55
56         for j = (1 : i - 1)
57             B(j) = B(j) - U(j, i) * x(i);
58         end
59     end
60
61 endfunction

```

## 2.12. PartialLUDecomposition

```

1 // Factor A as PA = LU
2 // @param: A a not singular matrix
3 // @return: L (not sure) lower triangle matrix
4 // @return: U upper triangle matrix
5 // @return: P permutation matrix
6
7 // To solve use PA = LU
8 // y = FS(PL, Pb)
9 // x = BS(U, y)
10
11 // @Author: Rosas Hernandez Oscar Andres
12 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
13 // @Author: Laurrabaquio Rodríguez Miguel Salvador
14 // @Author: Pahuá Castro Jesús Miguel Angel
15
16 function [L, U, P] = PartialLUDecomposition(A)
17     [m, n] = size(A);
18     P = eye(n, n); L = eye(n, n); U = A;
19
20     for step = (1 : n - 1)
21
22         maxPivotRow = step;
23         for posibilidad = (step + 1 : n)
24             if abs(U(posibilidad, step)) > U(maxPivotRow, step)
25                 maxPivotRow = posibilidad;
26             end
27         end
28
29         if (A(maxPivotRow, step) == 0)
30             error('Error: Singular matrix');
31         end
32
33         if maxPivotRow ~= step
34             if step >= 2
35                 for column = (1 : step - 1)
36                     temporal = L(step, column);
37                     L(step, column) = L(maxPivotRow, column);
38                     L(maxPivotRow, column) = temporal;
39                 end
40             end
41
42             for element = (1 : n)
43                 temporal = U(step, element)
44                 U(step, element) = U(maxPivotRow, element)
45                 U(maxPivotRow, element) = temporal
46
47                 temporal = P(step, element)
48                 P(step, element) = P(maxPivotRow, element)
49                 P(maxPivotRow, element) = temporal
50             end
51         end
52
53         for row = (step + 1 : n)
54             L(row, step) = U(row, step) / U(step, step);
55             for column = (1 : n)
56                 U(row, column) = U(row, column) - L(row, step) * U(step, column);
57             end
58         end
59     end
60
61 endfunction

```