
FACULTAD DE CIENCIAS

Tarea 3

ANÁLISIS NÚMÉRICO

Oscar Andrés Rosas Hernandez

Alarcón Alvarez Aylin

Laurrabaquio Rodríguez Miguel Salvador

Pahua Castro Jesús Miguel Ángel

Noviembre 2018

Índice

1. Problemas de Computadora	2
1.1. 23	3
1.2. 24	4
1.3. 25	10
1.4. 26	11
1.5. 27	17
2. Anexo	19
2.1. Givens	19
2.2. GramSchmidt	20
2.3. HouseHolder	21
2.4. LeastSquares	21
2.5. BackwardSubstitution	22
2.6. CholeskyBanachiewicz	23
2.7. CholeskyGaussian	24
2.8. CompleteLUdecomposition	25
2.9. ForwardSubstitution	26
2.10. LUdecomposition	27

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/Homework3/Code/Algorithms ')
```

Para ejecutar cada uno basta con hacer algo como:

```
1 exec(" /Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework3/Code/23.sce", -1)
```

1.1. 23

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

En este código muestra justo lo que se nos pide, por el método de ecuaciones normales veremos la solución aproximada para cada una de las 2 b que nos dan y vemos a aunque son muy parecidas los resultados son completamente diferentes, y si, aunque puede parecer que o el algoritmo esta mal o algo raro acaba de pasar basta con dar un vistazo a la matriz A y verla como lo que la estamos interpretando, un conjunto de vectores.

Recuerda que lo que estamos haciendo es encontrar una combinación lineal de estos vectores para aproximar a nuestros vectores b .

Pues si nuestros vectores no son ortogonales lo que pasa es que un pequeño cambio en el resultado nos da una combinación lineal muy diferente. Es justo lo que esta pasando estamos trabajando con vectores muy parecidos.

La mejor forma de explicar este fenomeno y mostrar porque la ortogonalidad es tan importante es ver este video que justo habla de porque $1, x, x^2, \dots$ es una base horrible para los polinomios. El link es <https://youtu.be/pYOGYQOXqTk>

Esto es un pdf y no me puedo quejar del espacio que utilizo, así que dejo el código:

```

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/Homework3/Code/Algorithms ')
7 clc;
8
9 A23 = [
10     0.16  0.10;
11     0.17  0.11;
12     2.02  1.29;
13 ]
14
15 b23 = [
16     0.26;
17     0.28;
18     3.31;
19 ]
20
21 b2_23 = [
22     0.27;
23     0.25;
24     3.33;
25 ]
26
27 disp("Ax = b")
28
29 disp("A:")
30 disp(A23)
31
32 disp("b:")
33 disp(b23)
34
35 x23 = LeastSquares(A23, b23)
36 disp("x:")
37 disp(x23)
38
39 disp("Ax:")
40 disp(A23 * x23)
41
42 disp("b_2:")
43 disp(b2_23)
44
45 x2_23 = LeastSquares(A23, b2_23)
46 disp("x_2:")
47 disp(x2_23)
48
49 disp("Ax_2:")
50 disp(A23 * x2_23)

```

1.2. 24

Antes que nada, espero estar haciendo el problema correcto, este:

3.5. A planet follows an elliptical orbit, which can be represented in a Cartesian (x, y) coordinate system by the equation

$$ay^2 + bxy + cx + dy + e = x^2.$$

(a) Use a library routine, or one of your own design, for linear least squares to determine the orbital parameters a, b, c, d, e , given the following observations of the planet's position:

x	1.02	0.95	0.87	0.77	0.67
y	0.39	0.32	0.27	0.22	0.18
x	0.56	0.44	0.30	0.16	0.01
y	0.15	0.13	0.12	0.13	0.15

In addition to printing the values for the orbital parameters, plot the resulting orbit and the given data points in the (x, y) plane.

(b) This least squares problem is nearly rank-deficient. To see what effect this has on the solution, perturb the input data slightly by adding

Ahhhh, este problema estuvo demasiado genial, ojala tuviera mas tiempo de intentar más problemas así, pero este si que requiere más explicación.

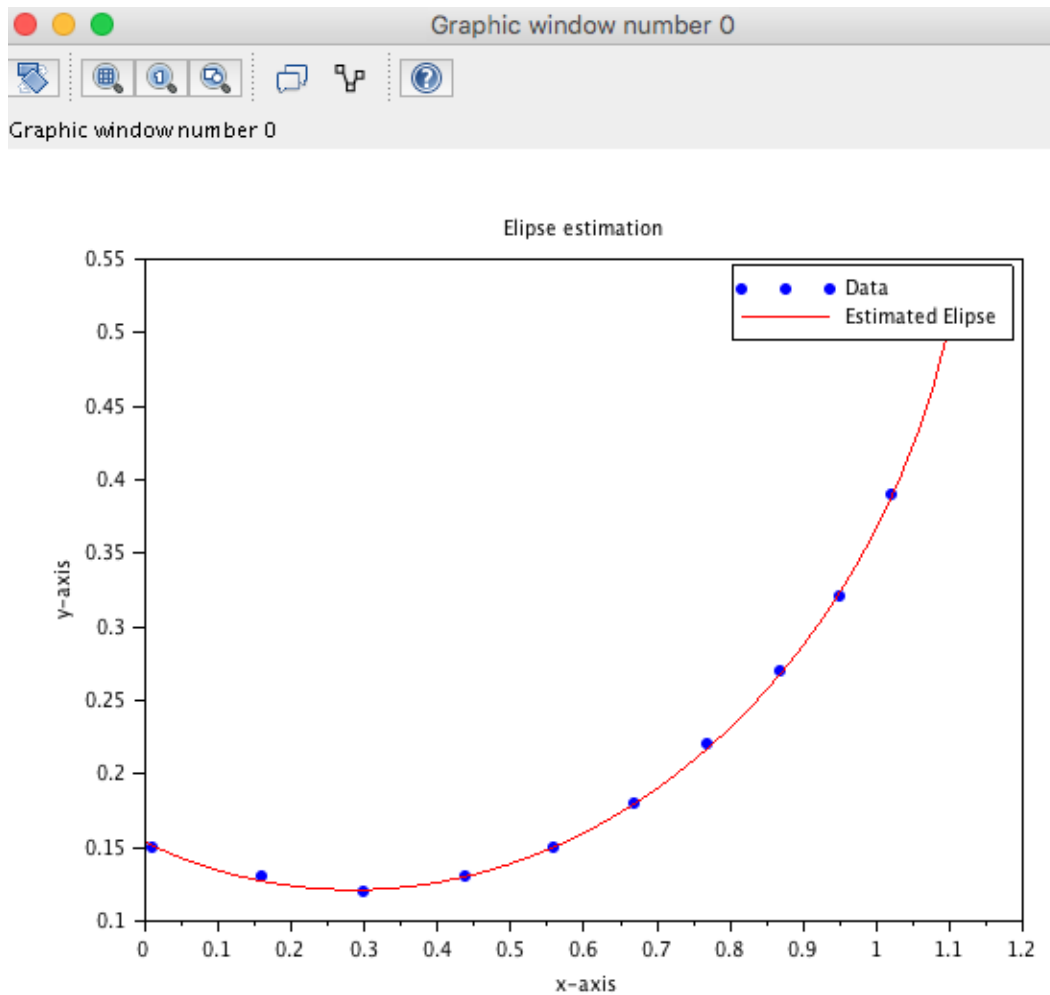
Lo que haremos primero será capturar 2 arreglos con la información que tenemos, después generaremos la matriz, que se basa justo en la ecuación $ay^2 + by + cd + e = x^2$

Ahora podemos resolver nuestro problema usando ecuaciones normales, con ello obtenemos un vector de coheficientes.

Por lo tanto ya nos podemos poner a hacer pronosticos, lo que harremos será generar un arreglo con puntos equidistantes de $x \in [0, 1]$.

Ahora, solo basta con ver que si x es una constante, la función que hemos calculado ahora nos regresa dos y , nos quedaremos con una y y vamos a graficarla, despues de todo con un x constante entonces $ay^2 + by + cd + e = x^2$ es una simple ecuación de segundo grado.

Así que mostramos, nuestra predicción, mira que hermoso, hasta te puse una foto de como se ve.



```

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/Homework3/Code/Algorithms' )
7 clc;
8
9 x24 = [
10     1.02;
11     0.95;
12     0.87;
13     0.77;
14     0.67;
15     0.56;
16     0.44;
17     0.30;
18     0.16;
19     0.01;
20 ]
21
22 y24 = [
23     0.39;
24     0.32;
25     0.27;
26     0.22;
27     0.18;
28     0.15;
29     0.13;
30     0.12;
31     0.13;
32     0.15;

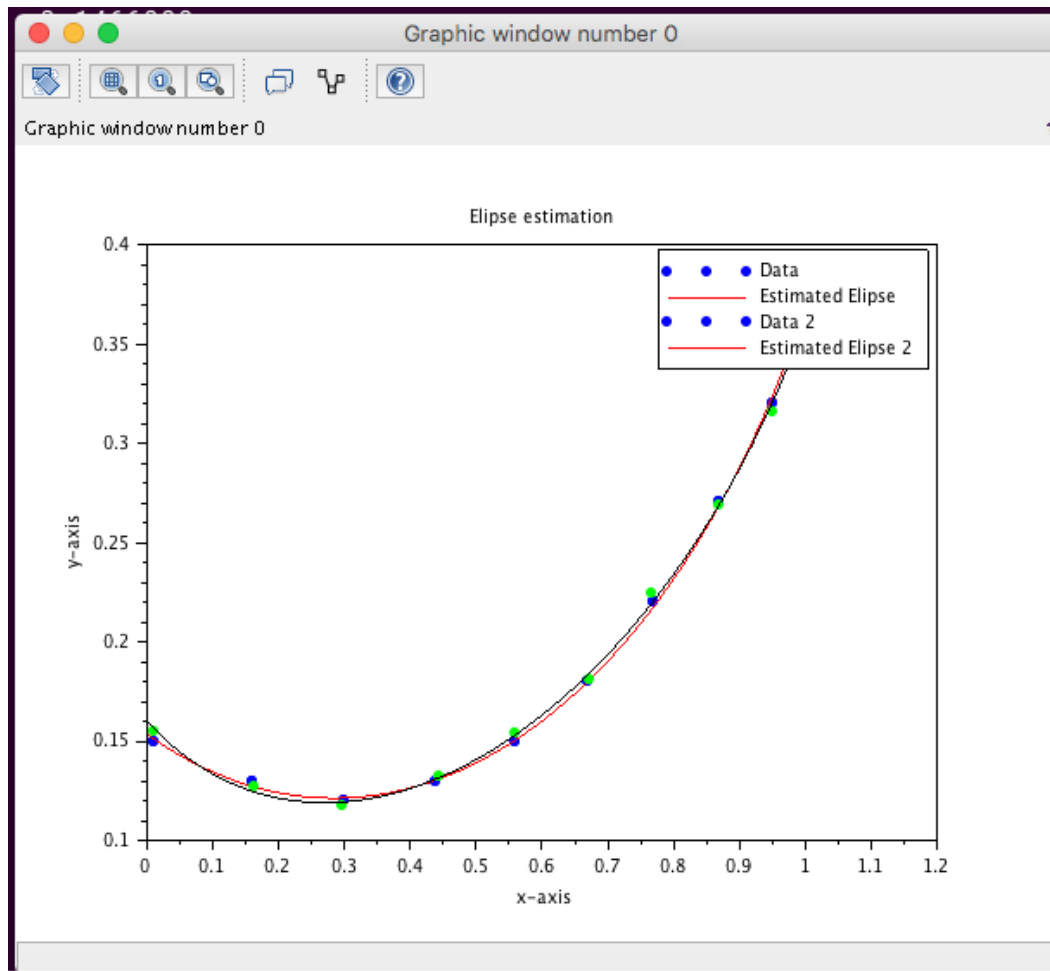
```

```

33 ]
34
35 A24 = eye(10, 5)
36 b24 = x24
37
38 for i = (1 : 10)
39     A24(i, 1) = y24(i) * y24(i)
40     A24(i, 2) = x24(i) * y24(i)
41     A24(i, 3) = x24(i)
42     A24(i, 4) = y24(i)
43     A24(i, 5) = 1
44
45     b24(i) = x24(i) * x24(i)
46 end
47
48 disp("Ax = b")
49
50 disp("xs:")
51 disp(x24)
52
53 disp("ys:")
54 disp(y24)
55
56 disp("A:")
57 disp(A24)
58
59 disp("b:")
60 disp(b24)
61
62 estimatedx24 = LeastSquares(A24, b24)
63 disp("x:")
64 disp(estimatedx24)
65
66 disp("Ax:")
67 disp(A24 * estimatedx24)
68
69 clf()
70
71 function [x] = solveEquation(a, b, c)
72     x = (-1 * b + sqrt(b*b - 4*a*c)) / (2 * a)
73 endfunction
74
75 function [y] = solve(coefficients, x)
76     a = coefficients(1)
77     b = coefficients(2)
78     c = coefficients(3)
79     d = coefficients(4)
80     e = coefficients(5)
81
82     reala = (a)
83     realb = (b * x + d)
84     realc = (c * x + e - x * x)
85
86     y = solveEquation(reala, realb, realc)
87 endfunction
88
89 someX = linspace(0, 1.10, 50)
90 someY = someX
91
92 for i = (1 : 50)
93     someY(i) = solve(estimatedx24, someX(i))
94 end
95
96 plot(x24, y24, 'b')
97 plot(someX, someY, 'r-')
98 hl=legend(['Data'; 'Estimated Ellipse']);
99 xtitle("Ellipse estimation", "x-axis", "y-axis")

```

La sección b trata de intentar mover un poco los datos sobre alguna insertudimbre y con eso ver como afecta la orbita y nuestras estimaciones y la verdad... todo Ok



```

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/Homework3/Code/Algorithms ' )
7 clc;
8
9 x24 = [
10     1.02;
11     0.95;
12     0.87;
13     0.77;
14     0.67;
15     0.56;
16     0.44;
17     0.30;
18     0.16;
19     0.01;
20 ]
21
22 y24 = [
23     0.39;
24     0.32;
25     0.27;
26     0.22;
27     0.18;
28     0.15;

```



```

29     0.13;
30     0.12;
31     0.13;
32     0.15;
33 ]
34
35 A24 = eye(10, 5)
36 b24 = x24
37
38 for i = (1 : 10)
39     A24(i, 1) = y24(i) * y24(i)
40     A24(i, 2) = x24(i) * y24(i)
41     A24(i, 3) = x24(i)
42     A24(i, 4) = y24(i)
43     A24(i, 5) = 1
44
45     b24(i) = x24(i) * x24(i)
46 end
47
48 clf()
49
50 function [x] = solveEquation(a, b, c)
51     x = (-1 * b + sqrt(b*b - 4*a*c)) / (2 * a)
52 endfunction
53
54 function [y] = solve(coefficients, x)
55     a = coefficients(1)
56     b = coefficients(2)
57     c = coefficients(3)
58     d = coefficients(4)
59     e = coefficients(5)
60
61     reala = (a)
62     realb = (b * x + d)
63     realc = (c * x + e - x * x)
64
65     y = solveEquation(reala, realb, realc)
66 endfunction
67
68 someX = linspace(0, 1, 50)'
69 someY = someX
70
71 for i = (1 : 50)
72     someY(i) = solve(estimatedx24, someX(i))
73 end
74
75 x24_2 = eye(10, 1)
76 y24_2 = eye(10, 1)
77
78 A24_2 = eye(10, 5)
79 b24_2 = x24_2
80
81 for i = (1 : 10)
82     x24_2(i) = x24(i) + (rand() * 0.010 - 0.005)
83     y24_2(i) = y24(i) + (rand() * 0.010 - 0.005)
84
85     A24_2(i, 1) = y24_2(i) * y24_2(i)
86     A24_2(i, 2) = x24_2(i) * y24_2(i)
87     A24_2(i, 3) = x24_2(i)
88     A24_2(i, 4) = y24_2(i)
89     A24_2(i, 5) = 1
90
91     b24_2(i) = x24_2(i) * x24_2(i)
92 end
93
94 estimatedx24_2 = LeastSquares(A24_2, b24_2)
95
96 clf()
97
98 someX_2 = linspace(0, 1, 50)'
99 someY_2 = someX_2
100
101 for i = (1 : 50)
102     someY_2(i) = solve(estimatedx24_2, someX_2(i))
103 end
104
105
106 plot(x24, y24, 'b')
107 plot(someX, someY, 'r-')
108
109 disp("=====")
110 disp(someX_2)
111 disp(someY_2)
112
113 plot(x24, y24, 'b')
114 plot(someX, someY, 'r-')
115
116 plot(x24_2, y24_2, 'green')

```

```
117 plot(someX_2, someY_2, 'black-')
118
119 hl=legend(['Data'; 'Estimated Elipse'; 'Data 2'; 'Estimated Elipse 2']);
120 xtitle("Elipse estimation", "x-axis", "y-axis")
```

1.3. 25

Dejemos que el código hable:

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pádua Castro Jesús Miguel Ángel
5
6 // Creates a very bad conditioned least squares problem
7 // @param: m a number, the size of the problem
8 // @param: n a number, the size of the problem
9 // @param: epsilon a number, the pertubation size
10
11 // @return: Q a matriz in  $M_{\{m \times 25\}}$  that is ortogonal
12 // @return: R a matriz in  $M_{\{m \times 25\}}$  that is triangular superior
13
14
15 function [] = 25(m, n, epsilon)
16
17     // Create the polynomial
18     t = zeros(m, 1)
19     for(i = 1 : m)
20         t(i) = (i-1) / (m-1)
21     end
22
23     //Some valuation result
24     y25 = zeros(m, 1)
25     for(i = 1 : m)
26         for(j = 1 : n)
27             y25(i) = y25(i) + t(i)^(j-1)
28         end
29     end
30
31     //Create the least square matrix
32     A25 = zeros(m, n)
33     for(i = 1 : m)
34         for(j = 1 : n)
35             A25(i, j) = t(i)^(j - 1)
36         end
37     end
38
39     //Lets create the perturbations
40     y2_25 = zeros(m, 1)
41     for(i = 1 : m)
42         y2_25(i) = y25(i) + (2*rand() - 1) * epsilon
43     end
44
45     //Lets solve
46     x25 = LeastSquares(A25, y25)
47     z = LeastSquares(A25, y2_25)
48
49     [Q1, R1] = GramSchmidt(A25, 0)
50     x2_25 = QRDecomposition(Q1, R1, y25)
51
52     [Q1, R1] = GramSchmidt(A25, 0)
53     x2_25 = QRDecomposition(Q1, R1, y2_25)
54
55     disp("a)")
56     if(norm(x25-z) < norm(x2_25-z))
57         disp("Least square is the least sensitive to pertubations")
58     else
59         disp("QR Decomposition is the least sensitive to pertubations")
60     end
61
62     disp("c)")
63     if(A25*x25 == y25)
64         disp("The difference in the solution to least squares do not affect the ajust in the polynomial")
65     else
66         disp("The difference in the solution to least squares do affect the ajust in the polynomial")
67     end
68
69     if(A25*x2_25 == y25)
70         disp("The difference in the solution to QR do not affect the ajust in the polynomial")
71     else
72         disp("The difference in the solution to QR do affect the ajust in the polynomial")
73     end
74
75 endfunction

```

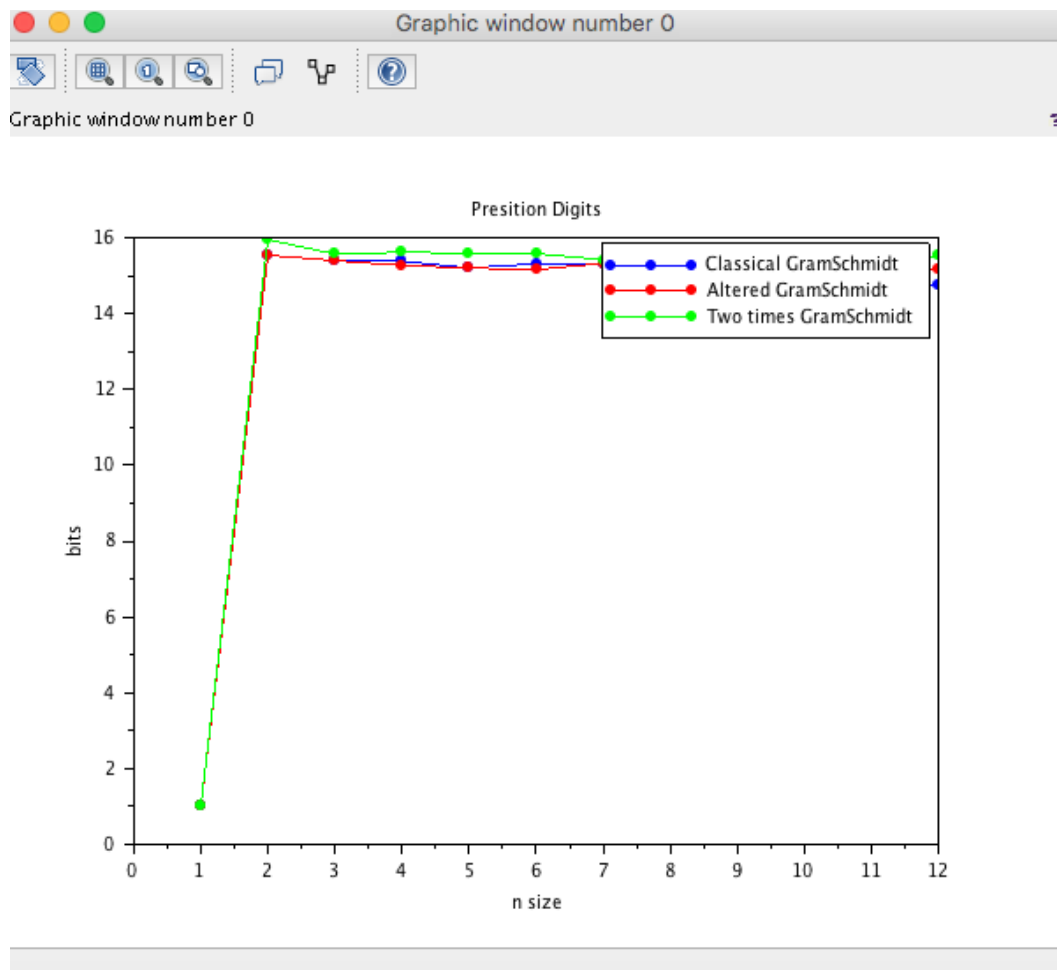
1.4. 26

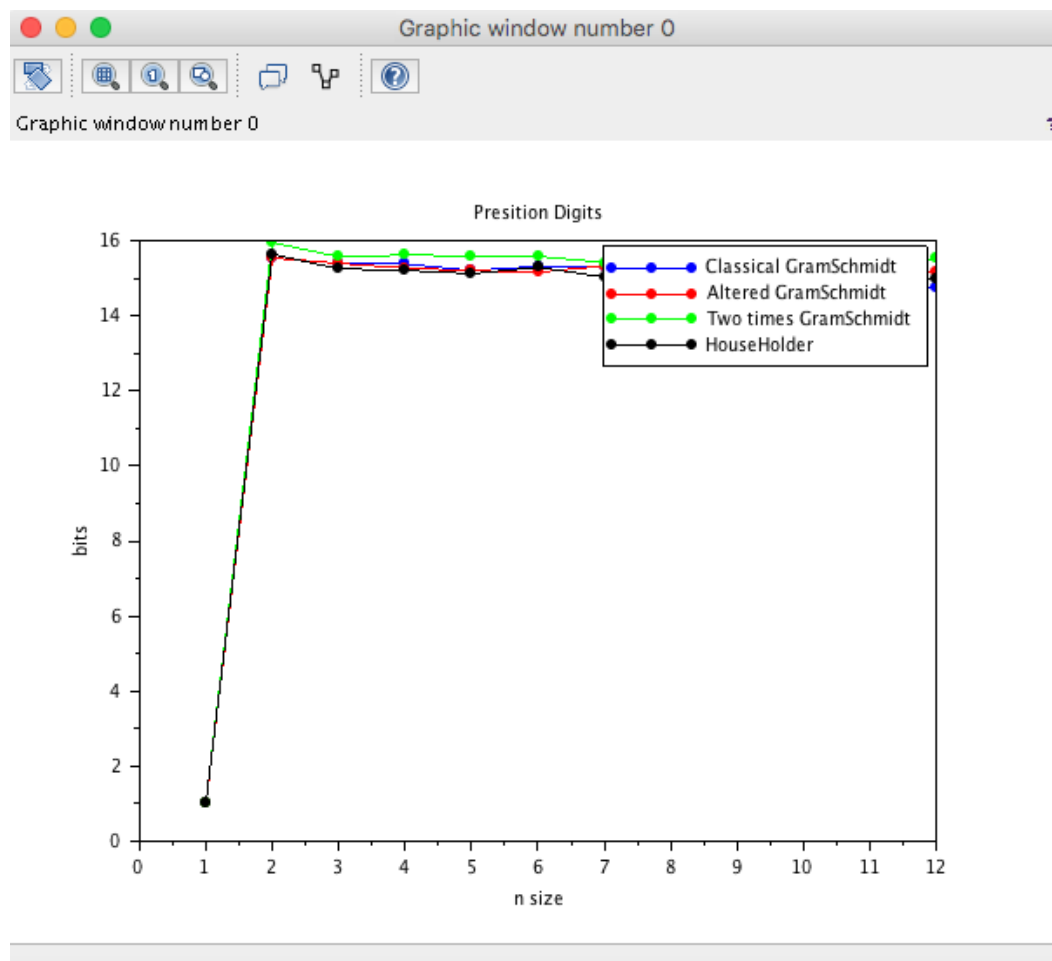
Admiremos antes que nada lo bonita que es la matriz de Hilbert:

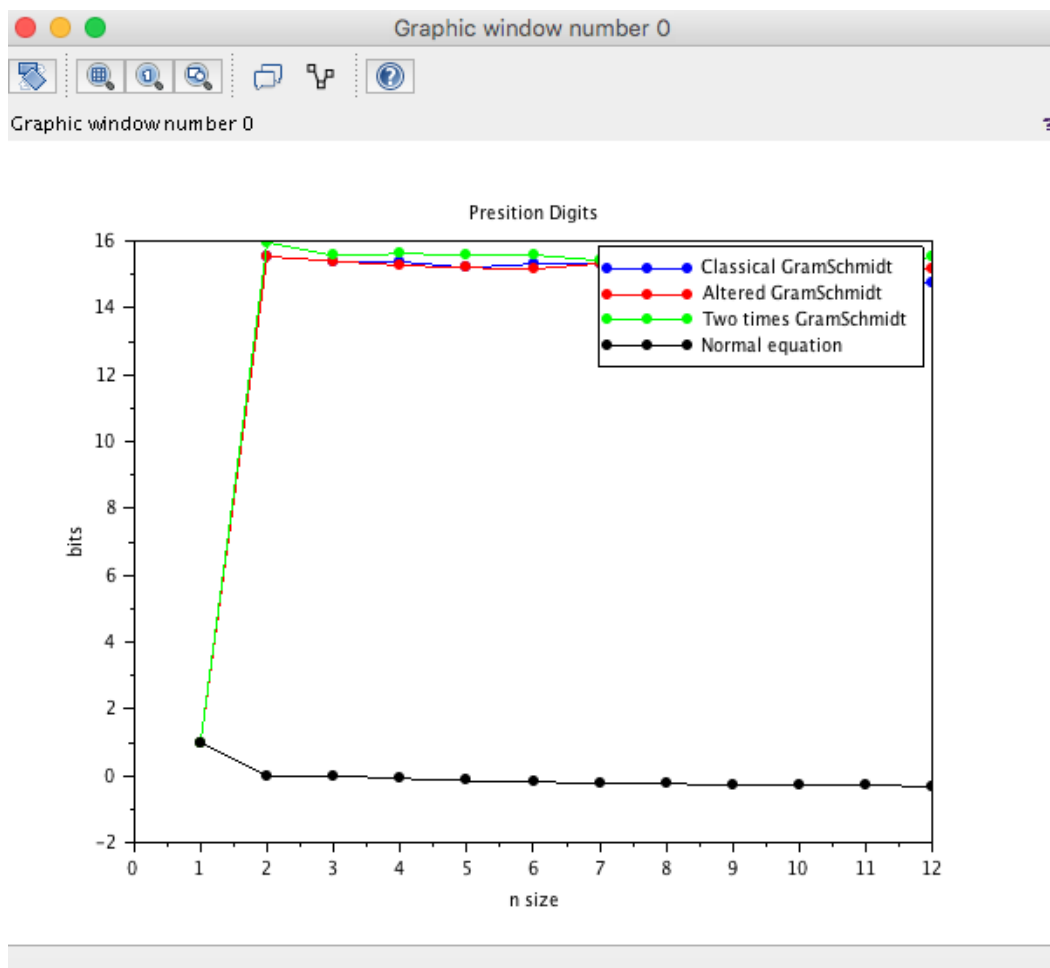
$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$$

Ahora si, basicamente lo que nos piden es graficar diversos métodos y los digitos de precisión conforme se hace crecer esta matriz.

Miremos los resultados y el código necesario para hacerlo funcionar:







```

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
10 digitsAltered = eye(12, 1)
11 digitsClassic = eye(12, 1)
12 digitsTwo = eye(12, 1)
13 data = eye(12, 1)
14
15 for i = (2 : 12)
16     H = eye(i, i)
17     data(i) = i
18
19     for i = (1 : i)
20         for j = (1 : i)
21             H(i, j) = 1 / (i + j - 1)
22         end
23     end
24
25
26 [Q26Classic, R26] = GramSchmidt(H, 0)
27 [Q26Altered, R26] = GramSchmidt(H, 1)
28 [Q26Two, R26] = GramSchmidt(Q26Classic, 0)
29
30 digitsClassic(i) = -log10(norm(eye(i, i) - Q26Classic' * Q26Classic))
31 digitsAltered(i) = -log10(norm(eye(i, i) - Q26Altered' * Q26Altered))
32 digitsTwo(i) = -log10(norm(eye(i, i) - Q26Two' * Q26Two))
33
34 disp("Q of GramSchmidt Classical:")
35 disp(Q26Classic)
36
37 disp("Digits:")
38 disp(digitsClassic)
39
40 disp("Q of GramSchmidt Altered:")
41 disp(Q26Altered)
42
43 disp("Digits:")
44 disp(digitsAltered)
45 end
46
47 plot(data, digitsClassic, '.b-')
48 plot(data, digitsAltered, '.r-')
49 plot(data, digitsTwo, '.g-')
50
51
52 hl=legend(['Classical GramSchmidt'; 'Altered GramSchmidt'; 'Two times GramSchmidt']);
53 xtitle("Presition Digits", "n size", "bits")

```

```

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
10 digitsAltered = eye(12, 1)
11 digitsClassic = eye(12, 1)
12 digitsTwo = eye(12, 1)
13 digitsHouse = eye(12, 1)
14 data = eye(12, 1)
15
16 for i = (2 : 12)
17     H = eye(i, i)
18     data(i) = i
19
20     for i = (1 : i)
21         for j = (1 : i)
22             H(i, j) = 1 / (i + j - 1)
23         end
24     end
25
26
27 [Q26Classic, R26] = GramSchmidt(H, 0)
28 [Q26Altered, R26] = GramSchmidt(H, 1)
29 [Q26Two, R26] = GramSchmidt(Q26Classic, 0)
30 [Q26H, R26] = HouseHolder(H)
31

```



```

32 digitsClassic(i) = -log10(norm(eye(i, i) - Q26Classic' * Q26Classic))
33 digitsAltered(i) = -log10(norm(eye(i, i) - Q26Altered' * Q26Altered))
34 digitsTwo(i) = -log10(norm(eye(i, i) - Q26Two' * Q26Two))
35 digitsHouse(i) = -log10(norm(eye(i, i) - Q26H' * Q26H))
36
37 disp("Q of GramSchmidt Classical:")
38 disp(Q26Classic)
39
40 disp("Digits:")
41 disp(digitsClassic)
42
43 disp("Q of GramSchmidt Altered:")
44 disp(Q26Altered)
45
46 disp("Digits:")
47 disp(digitsAltered)
48 end
49
50 plot(data, digitsClassic, '.b-')
51 plot(data, digitsAltered, '.r-')
52 plot(data, digitsTwo, '.g-')
53 plot(data, digitsHouse, '.black-')
54
55 hl=legend(['Classical GramSchmidt'; 'Altered GramSchmidt'; 'Two times GramSchmidt'; 'HouseHolder']);
56 xtitle("Presition Digits", "n size", "bits")

```

```

1 // @Author: Rosas H26ernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahuá Castro Jesús Miguel Ángel
5
6 getd('/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework2/Code/Algorithms')
7 clc;
8
9
10 digitsAltered = eye(12, 1)
11 digitsClassic = eye(12, 1)
12 digitsTwo = eye(12, 1)
13 digitsNormal = eye(12, 1)
14 data = eye(12, 1)
15
16 for i = (2 : 12)
17     H26 = eye(i, i)
18     data(i) = i
19
20     for i = (1 : i)
21         for j = (1 : i)
22             H26(i, j) = 1 / (i + j - 1)
23         end
24     end
25
26
27 [Q26Classic, R26] = GramSchmidt(H26, 0)
28 [Q26Altered, R26] = GramSchmidt(H26, 1)
29 [Q26Two, R26] = GramSchmidt(Q26Classic, 0)
30 [L26] = CholeskyGaussian(H26' * H26)
31 Q26N = H26 * L26'
32
33 digitsClassic(i) = -log10(norm(eye(i, i) - Q26Classic' * Q26Classic))
34 digitsAltered(i) = -log10(norm(eye(i, i) - Q26Altered' * Q26Altered))
35 digitsTwo(i) = -log10(norm(eye(i, i) - Q26Two' * Q26Two))
36 digitsNormal(i) = -log10(norm(eye(i, i) - Q26N' * Q26N))
37
38 disp("Q of GramSchmidt Classical:")
39 disp(Q26Classic)
40
41 disp("Q of GramSchmidt Altered:")
42 disp(Q26Altered)
43
44 end
45
46 plot(data, digitsClassic, '.b-')
47 plot(data, digitsAltered, '.r-')
48 plot(data, digitsTwo, '.g-')
49 plot(data, digitsNormal, '.black-')
50
51
52 hl=legend(['Classical GramSchmidt'; 'Altered GramSchmidt'; 'Two times GramSchmidt'; 'Normal
53 equation']);
54 xtitle("Presition Digits", "n size", "bits")

```

1.5. 27

Este proceso lo haremos hasta $\epsilon = 2^{-10}$ porque ϵ más pequeñas simplemente ya hacer que se trate a la matriz como singular pues se ve como una matriz con 3 columnas iguales, por lo tanto causa errores al tomarse como una matriz singular.

Lo que podemos ver que mientras epsilon más y más pequeña nos acercamos al claro resultado donde $x_1 = x_2 = x_3 = 0.333$

Algo curioso y diferente de los métodos anteriores, del parcial pasado es que prácticamente todos los métodos dan el mismo resultado, curiosamente GramSchmidt es el único diferente en ciertos ϵ .

```

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/Homework3/Code/Algorithms')
7 clc;
8
9
10 values = [
11     2;
12     1.1;
13     1;
14     0.9;
15     2e-01;
16     2e-02;
17     2e-05;
18     2e-08;
19     2e-10;
20     2e-11;
21 ]
22
23 [numberOfValues, _] = size(values)
24
25 for i = (1 : numberOfValues)
26
27     e = values(i)
28     disp("=====")
29     disp("epsilon:")
30     disp(e)
31
32     A27 = [
33         1 1 1;
34         e 0 0;
35         0 e 0;
36         0 0 e;
37     ]
38
39     b27 = [
40         1;
41         0;
42         0;
43         0;
44     ]
45
46     disp("Ax = b")
47
48     disp("A:")
49     disp(A27)
50
51     try
52         x27LeastSquares = LeastSquares(A27, b27)
53
54         [Q27HH, R27HH] = HouseHolder(A27)
55         x27HouseHolder = QRDecomposition(Q27HH, R27HH, b27)
56
57         [Q27GS, R27GS] = GramSchmidt(A27, 0)
58         x27GramSchmidt = QRDecomposition(Q27GS, R27GS, b27)
59
60         [Q27G, R27G] = Givens(A27)
61         x27Givens = QRDecomposition(Q27G, R27G, b27)
62
63
64     disp("x of Least Squares:")
65     disp(x27LeastSquares)

```

```
66
67     disp("Ax of Least Squares:")
68     disp(A27 * x27LeastSquares)
69
70     disp("x of HouseHolder:")
71     disp(x27HouseHolder)
72
73     disp("Ax of HouseHolder:")
74     disp(A27 * x27HouseHolder)
75
76     disp("x of GramSchmidt:")
77     disp(x27GramSchmidt)
78
79     disp("Ax of GramSchmidt:")
80     disp(A27 * x27GramSchmidt)
81
82     disp("x of Givens:")
83     disp(x27Givens)
84
85     disp("Ax of Givens:")
86     disp(A27 * x27Givens)
87 catch
88     disp("Error: Singular matrix")
89 end
90
91
92
93
94 end
```

2. Anexo

2.1. Givens

```
1 // Get the estimated solution to Ax = b using the HouseHolder transformation
2 // @param: A a matriz in  $M_{\{m \times n\}}$  where  $m > n$ 
3 // @return: Q a matriz in  $M_{\{m \times m\}}$  that is ortogonal
4 // @return: R a matriz in  $M_{\{m \times n\}}$  that is triangular superior
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 [Q, R] = Givens(A)
12     [m, n] = size(A);
13     Q = eye(m, m);
14     R = A;
15
16     for j = (1 : n)
17         for i = (m : -1 : j + 1)
18             GivenMatrix = eye(m, m);
19             [c, s] = GivensRotations(R(i-1, j), R(i, j));
20             GivenMatrix([i-1, i], [i-1, i]) = [c -s; s c];
21
22             R = GivenMatrix' * R;
23             Q = Q * GivenMatrix;
24         end
25     end
26 end
27
28 function [c,s] = GivensRotations(a,b)
29     if (b == 0)
30         c = 1;
31         s = 0;
32     else
33         if abs(b) > abs(a)
34             r = a / b;
35             s = 1 / sqrt(1 + r**2);
36             c = s * r;
37         else
38             r = b / a;
39             c = 1 / sqrt(1 + r**2);
40             s = c * r;
41         end
42     end
43 end
44
45 end
```

2.2. GramSchmidt

```

1 // Get the estimated solution to  $Ax = b$  using the Gram-Schmidt transformation
2 // @param: A a matriz in  $M_{\{m \times n\}}$  where  $m > n$ 
3 // @param: option if 1 then is the alterate else is the classic algorithm
4 // @return: Q a matriz in  $M_{\{m \times m\}}$  that is ortogonal
5 // @return: R a matriz in  $M_{\{m \times n\}}$  that is triangular superior
6
7
8 // @Author: Rosas Hernandez Oscar Andres
9 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
10 // @Author: Laurrabaquio Rodríguez Miguel Salvador
11 // @Author: Pahua Castro Jesús Miguel Angel
12
13 function [Q, R] = GramSchmidt(A, option)
14     [m, n] = size(A);
15
16     Q = zeros(m, n);
17     R = A;
18
19     for j = (1:n)
20
21         Q(:, j) = A(:, j)
22
23         for i = (1 : j - 1)
24             if (option == 1)
25                 R(i, j) = Q(:, i)' * Q(:, j);
26             else
27                 R(i, j) = Q(:, i)' * A(:, j);
28             end
29
30             Q(:, j) = Q(:, j) - R(i, j) * Q(:, i);
31         end
32
33         R(j, j) = norm(Q(:, j));
34         Q(:, j) = Q(:, j) / R(j, j);
35     end
36
37 endfunction

```

2.3. HouseHolder

```

1 // Get the estimated solution to Ax = b using the HouseHolder transformation
2 // @param: A a matriz in M_{m x n} where m > n
3 // @return: Q a matriz in M_{m x m} that is ortogonal
4 // @return: R a matriz in M_{m x n} that is triangular superior
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 [Q, R] = HouseHolder(A, b)
12     [m, n] = size(A);
13     Q = eye(m, m);
14
15     for i = (1 : n)
16         aei = zeros(m - i + 1, 1);
17
18         alpha = -sign(A(i,i)) * norm(A(i : m, i));
19         aei(1) = alpha
20
21         v = A(i : m, i) - aei;
22         HouseHolder = eye(m - (i-1), m-(i-1)) -2 * ((v * v') / (v' * v));
23
24         RealHouseHolder = eye(m, m);
25         RealHouseHolder(i : m, i : m) = HouseHolder;
26
27         A = RealHouseHolder * A;
28         Q = Q * RealHouseHolder;
29
30     end
31     R = A;
32
33 endfunction

```

2.4. LeastSquares

```

1 // Get the estimated solution to Ax = b using least squares
2 // @param: A a matriz in M_{m x n} where m > n
3 // @param: b a vector of m rows
4 // @return: x a estimated solution
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 [x] = LeastSquares(A, b)
12     [L] = CholeskyBanachiewicz(A' * A, 1);
13
14     y = FowardSubstitution(L, A' * b);
15     x = BackwardSubstitution(L', y);
16 endfunction

```

2.5. 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
```

2.6. 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: Pahuá Castro Jesús Miguel Ángel
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

```


2.7. 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 Ángel
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

```

2.8. 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

```

2.9. 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
```

2.10. 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: Pádua Castro Jesús Miguel Angel
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 endfunction
32
```