EAFIT UNIVERSITY
IT AND SYSTEMS DEPARTMENT
FINAL PROJECT
Third report

Objective: Implementation of the second batch of numerical methods seen until this point in the semester, including the additional ones.

Course: Numerical analysis.
Teacher: Edwar Samir Posada Murillo.
Semester: 2020-2.
Due date for the Third report:Wednesday November 11.
Project name: Numerical views.
Project Repository: Link Repo

Members:

Mariana Ramírez Duque (marami21@eafit.edu.co)

Nicolás Roldán Ramírez (nroldanr@eafit.edu.co)

Mateo Sánchez Toro (msanchezt@eafit.edu.co)

Maria Cristina Castrillon (Mcastri6@eafit.edu.co)

Report description:We will write the pseudocode of each method followed by an example execution. The code can be found on the repository link cited above

1 Methods

Algorithm 1: Incremental search

input : f: function to find root of,
X0: first root approximation,
Delta: delta(x),
N: maximum number of iterations,

if *(y=0)* **then**
    x0 is root;
else
    $x1 \leftarrow [x0\ delta]$
    $y1 \leftarrow [f(x1)]$
    $counter \leftarrow [1]$
while $y1 * y0 > 0$ *and counter* $< n$ *do* **do**
    $x0 \leftarrow [x1]$
    $y0 \leftarrow [y1]$
    $x1 \leftarrow [x0 + delta]$
    $y1 \leftarrow [f(x1)]$
    $counter \leftarrow counter + 1$
if *(y1==0)* **then**
    x1 is root;
else
    if $(y1 * y0 < 0)$ **then**
        [x0,x1] define an interval;
    else
        print("didn't find interval in N iterations")

Algorithm 2: Extra method Muller

input : function f, root approximation 1 x0,root approximation 2 x1 , root approximation 3 x2, tolerance tol, Maximum number of iterations N
output: root approximation

$h1 = x1 - x0$
$h2 = x2 - x1$
$t1 = (f(x1) - f(x0))/h1$
$t2 = (f(x2) - f(x1))/h2$
$d = (t2 - t1)/(h2 + h1)$
for $i \leftarrow 3$ **to** N **do**
  $b = t2 + h2 * d$
  $D = (b^2 - 4 * f(x2) * d)^{1/2}$
  if $abs(b - D) < abs(b + D)$ **then**
    $E = b + D$;
  else
    $E = b - D$
  $h = (-2 * f(x2))/E$
  $e = x$
  $x = x2 + h$
  $e = abs(e - x)$
  if $abs(e) \leq tol)$ **then**
    break;
  $x0 = x1$
  $x1 = x2$
  $x2 = x$
  $h1 = x1 - x0$
  $h2 = x2 - x1$
  $t1 = (f(x1) - f(x0))/h1$
  $t2 = (f(x2) - f(x1))/h2$
  $d = (t2 - t1)/(h2 + h1)$

Algorithm 3: Vandermonde

input : x, y
output: Polynomial coefficients, Vandermonde polynom

Repeat
Repeat
A(j,i)=X(j)**(n-i)
until($j < n$)
until($i < n$)
print: (Reverse A)*y

Algorithm 4: Newton Divided difference

input : vector x0, x1, ...xn, vector values f(x0); a point to evaluate
output: Divided differences F0,0 ...Fn,n

Step 1
for *i=0,...,n* **do**
  Set Fi,0= f(Xi)
Step 2
for *i=1,...,n* **do**
  for *j=1,2,....i* **do**
    Set Fi,j= ((Fi,j-1) -(Fi-1,j-1))/(Xi-Xi-j)
  End ;
Output (F0,0...,Fi,i,...Fn,n)
STOP

Algorithm 5: Lagrange

input : vector x0, x1, ...xn, vector values f(x0); a point to evaluate
output: P Lagrange polynomial P(x) evaluated at z

Step 1
Initialize variables. Set P z equal zero. Set n to the number of pairs of points (x, y). Set L to be the all ones vector of length n
Step 2
for *i=1,2,....n* **do**
  Step 3
  for *j=1,2,....i* **do**
    Step 4
    if *i = j then Li=(z-Xj)/(Xi-Xj)*Li* **then**
      break ;
    Step 5
Pz= $(Li * yi + Pz)$)
Step 6 Output Pz. STOP

Algorithm 6: Neville

input : Numbers x0, x1, ...xn, values f(x0),f(x1)...f(xn)
output: the table Q with p(x)=Qn,n

Step 1
for *i=1,2,....n* **do**
  for *j=1,2,....i* **do**
Set Q(ij)=((x-xi-j)Qi,j-1 - (x-xi)Qi-1,j-1)/(xi-xi-j)
Step 2
Output(Q);
STOP

Algorithm 7: Simple LU

input : nxn matrix A, column vector b
output: solution vector x

if *(A is not square) or (size of A and size of b are not computable)* **then**
  break ;
if *det(A) = 0* **then**
  break ;
$A \leftarrow LU$
$z \leftarrow \ progresiveSustitution([L\ b])$
$x \leftarrow \ RegressiveSustitution([U\ \ z])$

Algorithm 8: Partial Pivoting LU

input : nxn matrix A, column vector b
output: solution vector x

if *(A is not square) or (size of A and size of b are not computable)* **then**
  break ;
if *det(A) = 0* **then**
  break ;
$PA \leftarrow LU$
$z \leftarrow \ progresiveSustitution([L \quad P * b])$
$x \leftarrow \ RegressiveSustitution([U\ \ z])$

Algorithm 9: SOR

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax
output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D - w * L)^{-1} * ((1 - w) * D + w * U)$
$C \leftarrow w * (D - w * L)^{-1} * b$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance\ and\ counter < nmax$ **do**
  $xact = T * xant + C$
  $error = norm(xant - xact)$
  $xant = xact;$
  $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

Algorithm 10: Gauss-Seidel

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax

output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D - L)^{-1} * U)$
$C \leftarrow (D - L)^{-1} * b)$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance$ *and* $counter < nmax$ **do**
    $xact = T * xant + C$
    $error = norm(xant - xact)$
    $xant = xact;$
    $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

Algorithm 11: Jacobi

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax

output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D)^{-1} * (L + U)$
$C \leftarrow (D)^{-1} * b$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance$ *and* $counter < nmax$ **do**
    $xact = T * xant + C$
    $error = norm(xant - xact)$
    $xant = xact;$
    $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

Algorithm 12: Crout

input : nxn matrix A, column vector b
output: solution vector x

$L \leftarrow I(n)$
$U \leftarrow I(n)$
for $i \leftarrow 1$ **to** $n-1$ **do**
  for $j \leftarrow i$ **to** n **do**
    $L_{j,i} \leftarrow A_{j,i} - dot(L(j, 1:i-1), U(1:i-1, i)')$
  for $j \leftarrow i+1$ **to** n **do**
    $U_{i,j} \leftarrow A_{i,j} - dot(L(i, 1:i-1), U(1:i-1, j)')$
$L[n,n] = A[n,n] - dot(L[n, 1:n-1], U[1:n-1, n]');$
$z = progresiveSustitution([L\ b]);$
$x = regresiveSustitution([U\ z]);$

Algorithm 13: Cholesky

input : nxn matrix A, column vector b, initial approximation X0, weighing factor w, tolerance, maximum iterations Nmax
output: solution vector x
$L \leftarrow I(n)$
$U \leftarrow I(n)$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $L_{i,i} \leftarrow \sqrt{A_{i,i} - dot(L(i, 1:i-1), U(1:i-1, i)')}$
  $U_{i,i} \leftarrow L_{i,i}$
  for $j \leftarrow i+1$ **to** n **do**
  L[j,i]=(A[j,i]-dot(L[j,1:i-1],U[1:i-1,i]'))/U(i,i);
  $U[i,j] = (A[i,j] - dot(L[i, 1:i-1], U[1:i-1, j]'))/L[i,i];$
$L[n,n] = A[n,n] - dot(L[n, 1:n-1], U[1:n-1, n]');$
$z = progressiveSustitution([Lb]);$
$x = regressiveSustitution([Uz]);$

2 Evidence

```
INCREMENTAL SEARCHES

Results:
There is a root of f in [ -2.5 , -2.0 ]
There is a root of f in [ -1.0 , -0.5 ]
There is a root of f in [ 0.5 , 1.0 ]
There is a root of f in [ 2.0 , 2.5 ]
There is a root of f in [ 4.0 , 4.5 ]
There is a root of f in [ 5.0 , 5.5 ]
There is a root of f in [ 7.0 , 7.5 ]
There is a root of f in [ 8.0 , 8.5 ]
There is a root of f in [ 10.0 , 10.5 ]
There is a root of f in [ 11.5 , 12.0 ]
There is a root of f in [ 13.5 , 14.0 ]
There is a root of f in [ 14.5 , 15.0 ]
There is a root of f in [ 16.5 , 17.0 ]
There is a root of f in [ 17.5 , 18.0 ]
There is a root of f in [ 19.5 , 20.0 ]
There is a root of f in [ 21.0 , 21.5 ]
There is a root of f in [ 22.5 , 23.0 ]
There is a root of f in [ 24.0 , 24.5 ]
There is a root of f in [ 26.0 , 26.5 ]
There is a root of f in [ 27.0 , 27.5 ]
There is a root of f in [ 29.0 , 29.5 ]
There is a root of f in [ 30.0 , 30.5 ]
There is a root of f in [ 32.0 , 32.5 ]
There is a root of f in [ 33.5 , 34.0 ]
There is a root of f in [ 35.0 , 35.5 ]
There is a root of f in [ 36.5 , 37.0 ]
There is a root of f in [ 38.5 , 39.0 ]
There is a root of f in [ 39.5 , 40.0 ]
There is a root of f in [ 41.5 , 42.0 ]
There is a root of f in [ 43.0 , 43.5 ]
There is a root of f in [ 44.5 , 45.0 ]
```

Muller Method

```
Muller method
 iter|      x0          |         x1        |       x2          |          Root      |          E
   0 |   -1.000000e+00 |    0.000000e+00  |    1.000000e+00  |                    |
   1 |   -1.000000e+00 |    0.000000e+00  |    1.000000e+00  |    -2.220446e-16  |
   2 |    0.000000e+00 |    1.000000e+00  |   -2.220446e-16  |    -2.220446e-16  |    0.000000e+00
A root aproximation was found at -2.220446049250313e-16
```

Figure 2: Proof of Muller Method

Doolitle

```
24 [0.5251077877274448, 0.25545768216313536, -0.41047969902650905, -0.2816585612350974]
25 [0.5251083414671069, 0.25545801583827094, -0.4104799594870898, -0.281658892623501]
26 [0.5251086734271935, 0.25545821587239254, -0.4104801156299903, -0.2816590912867551]
27 [0.5251088724331645, 0.25545833579037724, -0.41048020923573025, -0.2816592103829217]
28 [0.5251089917347855, 0.25545840767972766, -0.41048026535121496, -0.28165928177960226]
29 [0.5251090632546336, 0.25545845077650514, -0.4104802989917548, -0.28165932458103016]
30 [0.5251091061298989, 0.25545847661249077, -0.41048031915884275, -0.2816593502399575]
```

Figure 3: Proof of Doolitle

Jacobi

```
| 42 |   1.5e-06 | 0.525106  0.255457  -0.410479  -0.281657
| 43 |   1.1e-06 | 0.525107  0.255457  -0.410479  -0.281658
| 44 |   8.7e-07 | 0.525107  0.255457  -0.410479  -0.281658
| 45 |   6.5e-07 | 0.525108  0.255458  -0.410480  -0.281658
| 46 |   4.9e-07 | 0.525108  0.255458  -0.410480  -0.281659
| 47 |   3.7e-07 | 0.525108  0.255458  -0.410480  -0.281659
| 48 |   2.8e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 49 |   2.1e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 50 |   1.6e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 51 |   1.2e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 52 |   9.0e-08 | 0.525109  0.255458  -0.410480  -0.281659
```

Figure 4: Proof of Jacobi

Vandermonde

```
Vandermonde Matrix:
[[-1.  1. -1.  1.]
 [ 0.  0.  0.  1.]
 [27.  9.  3.  1.]
 [64. 16.  4.  1.]]
Polynomial coefficients:  [-1.1417  5.825  -5.5333  3.    ]
Vandermonde polynom:
-1.1416666666666666 *x^ 3 +
5.824999999999999 *x^ 2 +
-5.533333333333332 *x^ 1 +
3.0 *x^ 0 +
```

Figure 5: Proof of Vandermonde

Newton Divided Difference

```
Newton's polynomial coefficients:
[ 15.5    -12.5      3.5417  -1.1417]
Newton's Divided Difference Table
[[ 15.5      0.       0.       0.    ]
 [  3.     -12.5      0.       0.    ]
 [  8.       1.6667   3.5417   0.    ]
 [  1.      -7.      -2.1667  -1.1417]]
 Newton's polynom:
-1.14166666666667*x*(x - 3.0)*(x + 1.0) + 3.54166666666667*x*(x + 1.0) - 12.5*x
+ 3.0
Newton's simple polynom::
-1.14166666666667*x**3 + 5.825*x**2 - 5.53333333333333*x + 3.0
```

Figure 6: Proof of Newton Divided Difference

Lagrange

```
Lagrange interpolating polynomials:
L0 -0.05*x**3 + 0.35*x**2 - 0.6*x
L1 0.0833333333333333*x**3 - 0.5*x**2 + 0.416666666666667*x + 1.0
L2 -0.0833333333333333*x**3 + 0.25*x**2 + 0.333333333333333*x
L3 0.05*x**3 - 0.1*x**2 - 0.15*x
Lagrange polynom
15.5*L0+3.0*L1+8.0*L2+1.0*L3
```

Figure 7: Proof of Lagrange

Spline Linear

```
Lineal tracers coefficients

-12.5 <-> 3.0
1.66667 <-> 3.0
-7.0 <-> 29.0

Lineal tracers

3.0 - 12.5*x
1.66666666666667*x + 3.0
29.0 - 7.0*x
```

Figure 8: Proof of Spline Linear

Spline Square

```
Cuadratic tracers coefficients

0.0 <-> -12.5 <-> 3.0
4.72222 <-> -12.5 <-> 3.0
-22.83333 <-> 152.83333 <-> -245.0

Cuadratic tracers

3.0 - 12.5*x
4.72222222222222*x**2 - 12.5*x + 3.0
-22.8333333333333*x**2 + 152.833333333333*x - 245.0
```

Figure 9: Proof of Spline Square

Spline Cubic

```
Cubic tracers coefficients

2.53333 <-> 7.6 <-> -7.43333 <-> 3.0
-1.52222 <-> 7.6 <-> -7.43333 <-> 3.0
2.03333 <-> -24.4 <-> 88.56667 <-> -93.0

Cubic tracers

2.53333333333333*x**3 + 7.6*x**2 - 7.43333333333333*x + 3.0
-1.52222222222222*x**3 + 7.6*x**2 - 7.43333333333333*x + 3.0
2.03333333333333*x**3 - 24.4*x**2 + 88.5666666666667*x - 93.0
```

Figure 10: Proof of Spline Cubic

Neville

```
[[15.5, None, None, None], [3.0, 3.0 - 12.5*x, None, None], [8.0,
1.66666666666667*x + 3.0, -0.25*(3.0 - 12.5*x)*(x - 3.0) + 0.25*(x +
1.0)*(1.66666666666667*x + 3.0), None], [1.0, 29.0 - 7.0*x, 0.25*x*(29.0 -
7.0*x) - 0.25*(x - 4.0)*(1.66666666666667*x + 3.0), -0.2*(x - 4.0)*(-0.25*(3.0 -
12.5*x)*(x - 3.0) + 0.25*(x + 1.0)*(1.66666666666667*x + 3.0)) + 0.2*(x +
1.0)*(0.25*x*(29.0 - 7.0*x) - 0.25*(x - 4.0)*(1.66666666666667*x + 3.0))]]
-1.14166666666667*x**3 + 5.825*x**2 - 5.53333333333333*x + 3.0
```

Figure 11: Proof of Neville

LU Simple

```
Simple LU
 Stage 0

   4.000   -1.000    0.000    3.000
   1.000   15.500    3.000    8.000
   0.000   -1.300   -4.000    1.100
  14.000    5.000   -2.000   30.000

Stage 1

   4.000   -1.000    0.000    3.000
   0.000   15.750    3.000    7.250
   0.000   -1.300   -4.000    1.100
   0.000    8.500   -2.000   19.500
L:
   1.000    0.000    0.000    0.000
   0.250    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
   3.500    0.000    0.000    1.000
U:
   4.000   -1.000    0.000    3.000
   0.000   15.750    3.000    7.250
   0.000    0.000    0.000    0.000
   0.000    0.000    0.000    0.000

Stage 2

   4.000   -1.000    0.000    3.000
   0.000   15.750    3.000    7.250
   0.000    0.000   -3.752    1.698
   0.000    0.000   -3.619   15.587
L:
   1.000    0.000    0.000    0.000
   0.250    1.000    0.000    0.000
   0.000   -0.083    1.000    0.000
   3.500    0.540    0.000    1.000
U:
   4.000   -1.000    0.000    3.000
   0.000   15.750    3.000    7.250
   0.000    0.000   -3.752    1.698
   0.000    0.000    0.000    0.000

Stage 3

   4.000   -1.000    0.000    3.000
   0.000   15.750    3.000    7.250
   0.000    0.000   -3.752    1.698
   0.000    0.000    0.000   13.949
L:
   1.000    0.000    0.000    0.000
   0.250    1.000    0.000    0.000
   0.000   -0.083    1.000    0.000
```

LU Partial Pivot

```
Partial Pivot
 Stage 0

   4.000   -1.000    0.000    3.000
   1.000   15.500    3.000    8.000
   0.000   -1.300   -4.000    1.100
  14.000    5.000   -2.000   30.000

Stage 1


 L:

   1.000    0.000    0.000    0.000
   0.071    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
   0.286    0.000    0.000    1.000

 A:

  14.000    5.000   -2.000   30.000
   0.000   15.143    3.143    5.857
   0.000   -1.300   -4.000    1.100
   0.000   -2.429    0.571   -5.571

 P:

   0.000    0.000    0.000    1.000
   0.000    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
   1.000    0.000    0.000    0.000

 L:

   1.000    0.000    0.000    0.000
   0.071    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
   0.286    0.000    0.000    1.000

 U:

  14.000    5.000   -2.000   30.000
   0.000   15.143    3.143    5.857
   0.000    0.000    0.000    0.000
   0.000    0.000    0.000   -5.571
Stage 2


 L:

   1.000    0.000    0.000    0.000
   0.071    1.000    0.000    0.000
   0.000   -0.086    1.000    0.000
   0.286   -0.160    0.000    1.000

 A:

  14.000    5.000   -2.000   30.000
   0.000   15.143    3.143    5.857
   0.000    0.000   -3.730    1.603
   0.000    0.000    1.075   -4.632

 P:

   0.000    0.000    0.000    1.000
   0.000    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
```

```
P:

   0.000    0.000    0.000    1.000
   0.000    1.000    0.000    0.000
   0.000    0.000    1.000    0.000
   1.000    0.000    0.000    0.000

L:

   1.000    0.000    0.000    0.000
   0.071    1.000    0.000    0.000
   0.000   -0.086    1.000    0.000
   0.286   -0.160    0.000    1.000

U:

  14.000    5.000   -2.000   30.000
   0.000   15.143    3.143    5.857
   0.000    0.000   -3.730    1.603
   0.000    0.000    0.000   -4.632
Stage 3


L:

   1.000    0.000    0.000    0.000
   0.071    1.000    0.000    0.000
   0.000   -0.086    1.000    0.000
   0.286   -0.160   -0.288    1.000

A:

  14.000    5.000   -2.000   30.000
   0.000   15.143    3.143    5.857
   0.000    0.000   -3.730    1.603
   0.000    0.000    0.000   -4.170

P:
```

SOR

SOR

iter	E				
0		0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
1	6.162414e-01	3.750000e-01	6.048387e-02	-4.044859e-01	-2.680696e-01
2	3.183688e-01	5.117597e-01	3.419762e-01	-4.500492e-01	-3.046960e-01
3	1.453273e-01	5.901442e-01	2.352284e-01	-3.903364e-01	-3.085937e-01
4	1.098156e-01	5.153065e-01	2.815263e-01	-4.443708e-01	-2.712363e-01
5	7.360179e-02	5.280600e-01	2.439088e-01	-3.836051e-01	-2.833615e-01
6	4.309603e-02	5.212175e-01	2.551253e-01	-4.244577e-01	-2.793986e-01
7	2.128919e-02	5.243866e-01	2.580027e-01	-4.037994e-01	-2.822520e-01
8	1.074326e-02	5.270911e-01	2.525138e-01	-4.126297e-01	-2.822292e-01
9	6.899037e-03	5.236550e-01	2.581368e-01	-4.109464e-01	-2.810727e-01
10	5.517942e-03	5.261806e-01	2.536968e-01	-4.091465e-01	-2.821289e-01
11	4.227797e-03	5.244410e-01	2.563803e-01	-4.117903e-01	-2.813184e-01
12	2.849285e-03	5.254053e-01	2.550853e-01	-4.095027e-01	-2.818461e-01
13	1.690326e-03	5.250312e-01	2.555134e-01	-4.110729e-01	-2.815844e-01
14	8.831839e-04	5.250844e-01	2.555475e-01	-4.101965e-01	-2.816734e-01
15	4.363164e-04	5.251707e-01	2.553365e-01	-4.105686e-01	-2.816738e-01
16	2.698699e-04	5.250488e-01	2.555621e-01	-4.104927e-01	-2.816371e-01
17	2.155237e-04	5.251531e-01	2.553888e-01	-4.104310e-01	-2.816789e-01
18	1.666535e-04	5.250830e-01	2.554967e-01	-4.105317e-01	-2.816460e-01
19	1.138496e-04	5.251215e-01	2.554428e-01	-4.104415e-01	-2.816669e-01
20	6.816800e-05	5.251055e-01	2.554613e-01	-4.105042e-01	-2.816562e-01
21	3.592628e-05	5.251084e-01	2.554617e-01	-4.104686e-01	-2.816601e-01
22	1.771948e-05	5.251115e-01	2.554538e-01	-4.104842e-01	-2.816599e-01
23	1.064441e-05	5.251068e-01	2.554626e-01	-4.104806e-01	-2.816585e-01
24	8.426540e-06	5.251109e-01	2.554557e-01	-4.104785e-01	-2.816602e-01
25	6.571071e-06	5.251081e-01	2.554601e-01	-4.104823e-01	-2.816589e-01
26	4.537825e-06	5.251097e-01	2.554579e-01	-4.104788e-01	-2.816597e-01
27	2.747051e-06	5.251090e-01	2.554587e-01	-4.104813e-01	-2.816593e-01
28	1.462035e-06	5.251091e-01	2.554586e-01	-4.104799e-01	-2.816594e-01
29	7.201971e-07	5.251093e-01	2.554583e-01	-4.104805e-01	-2.816594e-01
30	4.207385e-07	5.251091e-01	2.554587e-01	-4.104804e-01	-2.816594e-01
31	3.293455e-07	5.251092e-01	2.554584e-01	-4.104803e-01	-2.816594e-01
32	2.588758e-07	5.251091e-01	2.554586e-01	-4.104804e-01	-2.816594e-01
33	1.807148e-07	5.251092e-01	2.554585e-01	-4.104803e-01	-2.816594e-01
34	1.105975e-07	5.251092e-01	2.554585e-01	-4.104804e-01	-2.816594e-01
35	5.945865e-08	5.251092e-01	2.554585e-01	-4.104803e-01	-2.816594e-01

PS C:\Users\maria\OneDrive - GR Chia S.A.S\ACADEMICO\Numerico\proyecto\NumericalViews\Methods>

Figure 14: Proof of SOR

Cholesky

```
Cholesky
 Stage 0

   4.000   -1.000    0.000    3.000
   1.000   15.500    3.000    8.000
   0.000   -1.300   -4.000    1.100
  14.000    5.000   -2.000   30.000

Stage 1

L:

 (   2.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.500+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)
 (   7.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)
U:

 (   2.000+   0.000j)  (  -0.500+   0.000j)  (   0.000+   0.000j)  (   1.500+   0.000j)
 (   0.000+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)

Stage 2

L:

 (   2.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.500+   0.000j)  (   3.969+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (  -0.328+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)
 (   7.000+   0.000j)  (   2.142+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)
U:

 (   2.000+   0.000j)  (  -0.500+   0.000j)  (   0.000+   0.000j)  (   1.500+   0.000j)
 (   0.000+   0.000j)  (   3.969+   0.000j)  (   0.756+   0.000j)  (   1.827+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)

Stage 3

L:

 (   2.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.500+   0.000j)  (   3.969+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (  -0.328+   0.000j)  (   0.000+   1.937j)  (   0.000+   0.000j)
 (   7.000+   0.000j)  (   2.142+   0.000j)  (   0.000+   1.868j)  (   1.000+   0.000j)
U:

 (   2.000+   0.000j)  (  -0.500+   0.000j)  (   0.000+   0.000j)  (   1.500+   0.000j)
 (   0.000+   0.000j)  (   3.969+   0.000j)  (   0.756+   0.000j)  (   1.827+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   1.937j)  (   0.000+  -0.877j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   1.000+   0.000j)

Stage 4

L:

 (   2.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.500+   0.000j)  (   3.969+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)
 (   0.000+   0.000j)  (  -0.328+   0.000j)  (   0.000+   1.937j)  (   0.000+   0.000j)
 (   7.000+   0.000j)  (   2.142+   0.000j)  (   0.000+   1.868j)  (   3.735+   0.000j)
U:

 (   2.000+   0.000j)  (  -0.500+   0.000j)  (   0.000+   0.000j)  (   1.500+   0.000j)
 (   0.000+   0.000j)  (   3.969+   0.000j)  (   0.756+   0.000j)  (   1.827+   0.000j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   1.937j)  (   0.000+  -0.877j)
 (   0.000+   0.000j)  (   0.000+   0.000j)  (   0.000+   0.000j)  (   3.735+   0.000j)

After applying progresive and regresive sustitution
 X:
 (275993036322204239822896234496.000+   0.000j)
 (170125478496320516419793977344.000+   0.000j)
 (-1408933916339496727540557414440.000+  -0.000j)
 (-3112822222641654692291396239936.000+   0.000j)
```

Figure 15: Proof of Cholesky