Final proyect

Second delivery

Santiago Alzate Cardona
Alejandro Castaño.
Esteban Sierra Patiño.
Sebatian Urrego.

November 10, 2021

Abstract

We made some of the mathematical methods for the numerical analysis project, we created the pseudocode, also two formal codes in python (main) and MathLab, that represents some of the uses you can take in those methods.

1 Introduction

Due to the fact that in the final project we will have to present many methods and have an application development in this first delivery we will deliver several methods in implemented code and its respective pseudo-code with their tests to show an advance phase and not to be left behind in later deliveries, as we rather show what we are capable of developing from the scratch.

In this document we only show the methods with their code and tests, however you can find all the files in the following Github.

2 Tests

2.1 Python

2.1.1 Simple LU

```python
def lusimpl(A, b):
    n = A.shape[0]
    L = np.eye(n)
    U = np.zeros((n, n))
    M = A
    M = M.astype('float')
    stack = []
    etapa = 0
    # Todas las columnas de A - 1
    for i in range(0, n - 1):
        print("Etapa", etapa)
        print(M)
        print()
        etapa += 1
        # Por cada columna iterar por cada fila debajo de la diagonal
        for j in range(i+1, n):
            if M[j][i] != 0:
                # Operacion de subfila
                L[j][i] = M[j][i]/M[i][i]
                M[j][i:n+1] = M[j][i:] - (M[j][i]/M[i][i])*M[i][i:]

        U[i, i:n] = M[i, i:n]
        U[i+1, i+1:n] = M[i+1, i+1:n]

    U[n-1][n-1] = M[n-1][n-1]

    print("Etapa", etapa)
    print(M, '\n')
    stack = np.array(stack, dtype=int).reshape(-1,2)
    # Sustitucion regresiva
    #print("Despues de aplicar sustitucion regresiva\n")
    z = sustprog(np.column_stack((L, b)))
    x = sustreg(np.column_stack((U, z)))
    return x
```

Figure 1: Code of a simple LU method in python.

```
Etapa 0
[[ 4.   -1.    0.    3. ]
 [ 1.   15.5  3.    8. ]
 [ 0.   -1.3 -4.    1.1]
 [14.    5.   -2.   30. ]]

Etapa 1
[[ 4.    -1.     0.     3.  ]
 [ 0.    15.75  3.     7.25]
 [ 0.    -1.3  -4.     1.1 ]
 [ 0.     8.5  -2.    19.5 ]]

Etapa 2
[[ 4.          -1.           0.           3.        ]
 [ 0.          15.75         3.           7.25      ]
 [ 0.           0.          -3.75238095  1.6984127 ]
 [ 0.           0.          -3.61904762 15.58730159]]

Etapa 3
[[ 4.          -1.           0.           3.        ]
 [ 0.          15.75         3.           7.25      ]
 [ 0.           0.          -3.75238095  1.6984127 ]
 [ 0.           0.           0.          13.94923858]]

X:
 [ 0.52510917  0.25545852 -0.41048035 -0.28165939]
```

Figure 2: Simple LU test.

2.1.2 LU with partial pivot

```python
def lupar(A, b):
    n = A.shape[0]
    L = np.eye(n)
    U = np.zeros((n, n))
    P = np.eye(n)
    M = A
    M = M.astype('float')

    stack = []
    etapa = 0
    # Todas las columnas de A - 1
    for i in range(0, n - 1):
        print("Etapa", etapa)
        print(M)
        print()
        etapa += 1
        col = np.abs(M[i+1:, i])
        aux0 = np.max(col) # Maximo en columna
        aux1 = np.argmax(col) # Indice subcolumna

        if aux0 > abs(M[i][i]):
            aux2 = M[i+aux1+1, i:n]
            aux3 = P[i+aux1+1, :]
            M[[aux1+i+1, i], i:n] = M[[i, i+aux1+1], i:n]
            P[[i, i+aux1+1],:] = P[[i+aux1+1, i], :]
            if i>0:
                L[[i, i+aux1+1], 0:i] = L[[i+aux1+1, i], 0:i]

        # Por cada columna iterar por cada fila debajo de la diagonal
        for j in range(i+1, n):
            if M[j][i] != 0:
                # Operacion de subfila
                L[j][i] = M[j][i]/M[i][i]
                M[j][i:n+1] = M[j][i:] - (M[j][i]/M[i][i])*M[i][i:]

        U[i, i:n] = M[i, i:n]
        U[i+1, i+1:n] = M[i+1, i+1:n]

    U[n-1][n-1] = M[n-1][n-1]

    print("Etapa", etapa)
    print(M, '\n')
    stack = np.array(stack, dtype=int).reshape(-1,2)

    z = sustprog(np.column_stack((L, P.dot(b))))
    x = sustreg(np.column_stack((U, z)))
    return x
```

Figure 3: Code of a partial LU method in python.

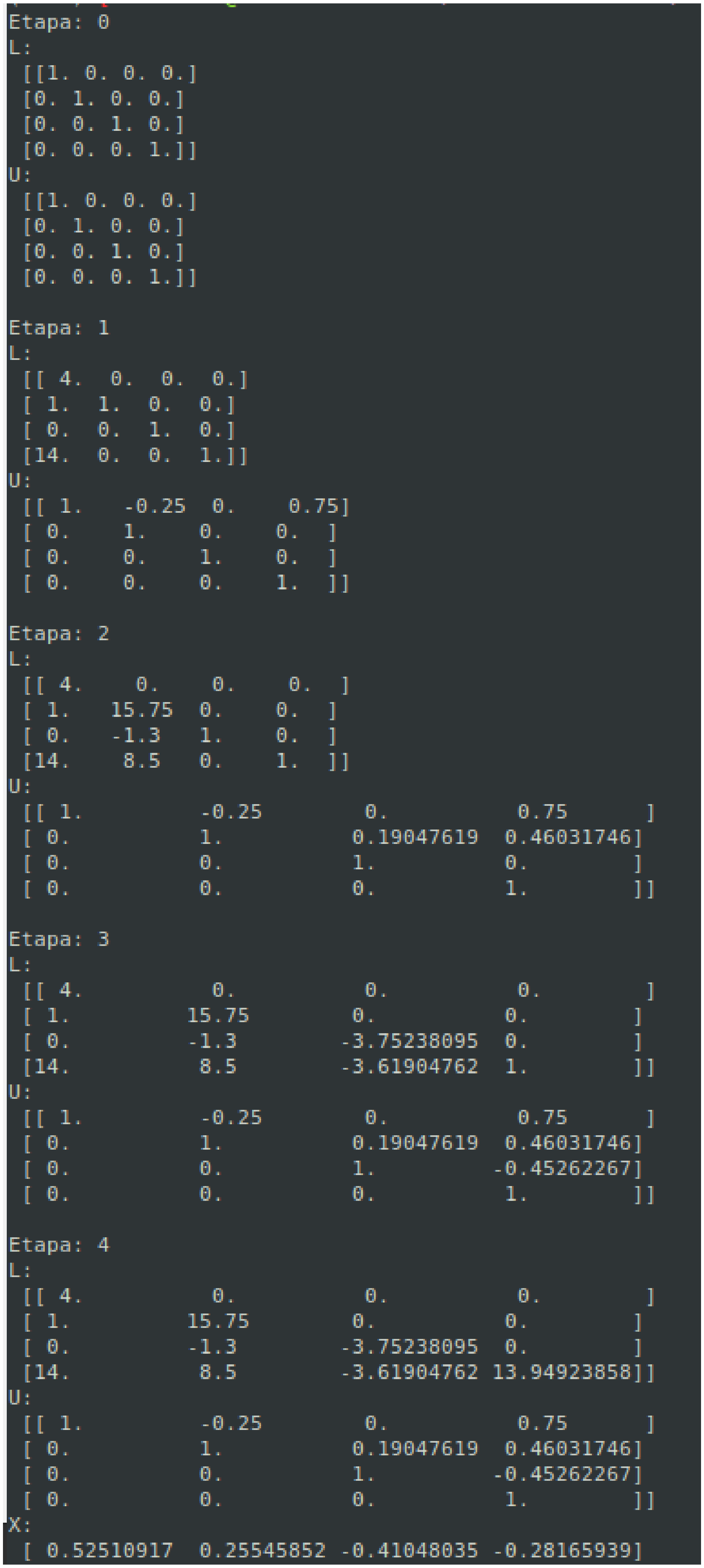
```
Etapa 0
[[ 4.   -1.    0.    3. ]
 [ 1.   15.5   3.    8. ]
 [ 0.   -1.3  -4.    1.1]
 [14.    5.   -2.   30. ]]

Etapa 1
[[14.          5.          -2.          30.        ]
 [ 0.         15.14285714   3.14285714   5.85714286]
 [ 0.         -1.3         -4.           1.1       ]
 [ 0.         -2.42857143   0.57142857  -5.57142857]]

Etapa 2
[[14.          5.          -2.          30.        ]
 [ 0.         15.14285714   3.14285714   5.85714286]
 [ 0.          0.          -3.73018868   1.60283019]
 [ 0.          0.           1.0754717   -4.63207547]]

Etapa 3
[[14.          5.          -2.          30.        ]
 [ 0.         15.14285714   3.14285714   5.85714286]
 [ 0.          0.          -3.73018868   1.60283019]
 [ 0.          0.           0.          -4.16995448]]

X:
 [ 0.52510917  0.25545852 -0.41048035 -0.28165939]
```

Figure 4: Partial LU test.

2.1.3 Crout

```python
def crout(A, b):
    n = np.array(A.shape[0])
    L = np.eye(n)
    U = np.eye(n)

    for i in range(0, n-1):

        for j in range(i, n):
            L[j][i] = A[j][i] - L[j, 0:i].dot( np.matrix.transpose(U[0:i,i]))
            print("L:\n", L)

        for j in range(i+1, n):
            U[i][j] = np.divide((A[i][j] - L[i, 0:i].dot( np.matrix.transpose(U[0:i,j]))), L[i][i])
            print("U:\n", U)

    L[n-1][n-1] = A[n-1][n-1] - L[n-1, 0:n-1].dot( np.matrix.transpose(U[0:n-1,n-1]))
    print("L:\n", L)
    z= sustprog(np.column_stack((L, b)))
    x = sustreg(np.column_stack((U, z)))

    return x
```

Figure 5: Code of crout method in python.



```
Etapa: 0
L:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
U:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Etapa: 1
L:
 [[ 4.  0.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [14.  0.  0.  1.]]
U:
 [[ 1.   -0.25  0.    0.75]
 [ 0.    1.    0.    0.  ]
 [ 0.    0.    1.    0.  ]
 [ 0.    0.    0.    1.  ]]

Etapa: 2
L:
 [[ 4.    0.    0.    0.  ]
 [ 1.   15.75  0.    0.  ]
 [ 0.   -1.3   1.    0.  ]
 [14.    8.5   0.    1.  ]]
U:
 [[ 1.         -0.25        0.          0.75      ]
 [ 0.          1.          0.19047619  0.46031746]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]

Etapa: 3
L:
 [[ 4.          0.          0.          0.        ]
 [ 1.         15.75        0.          0.        ]
 [ 0.         -1.3        -3.75238095  0.        ]
 [14.          8.5        -3.61904762  1.        ]]
U:
 [[ 1.         -0.25        0.          0.75      ]
 [ 0.          1.          0.19047619  0.46031746]
 [ 0.          0.          1.         -0.45262267]
 [ 0.          0.          0.          1.        ]]

Etapa: 4
L:
 [[ 4.          0.          0.          0.        ]
 [ 1.         15.75        0.          0.        ]
 [ 0.         -1.3        -3.75238095  0.        ]
 [14.          8.5        -3.61904762 13.94923858]]
U:
 [[ 1.         -0.25        0.          0.75      ]
 [ 0.          1.          0.19047619  0.46031746]
 [ 0.          0.          1.         -0.45262267]
 [ 0.          0.          0.          1.        ]]
X:
 [ 0.52510917  0.25545852 -0.41048035 -0.28165939]
```

Figure 6: Crout method test.

2.1.4 Doolittle

```python
def doolittle(A, b):
    n = np.array(A.shape[0])
    L = np.eye(n)
    U = np.eye(n)

    for i in range(0, n-1):
        for j in range(i, n):
            print("U:\n", U)
            U[i][j] = A[i][j] - L[i, 0:i].dot(np.matrix.transpose(U[0:i,j]))

        for j in range(i+1, n):
            print("L:\n", L)
            L[j][i] = np.divide((A[j][i] - L[j, 0:i].dot( np.matrix.transpose(U[0:i,i]))), U[i][i])

    print(L)
    print(U)
    U[n-1][n-1] = A[n-1][n-1] - L[n-1, 0:n-1].dot(np.matrix.transpose(U[0:n-1,n-1]))

    z= sustprog(np.column_stack((L, b)))
    x = sustreg(np.column_stack((U, z)))
    return x
```

Figure 7: Code of doolittle method in python.

```
Etapa: 0
L:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
U:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Etapa: 1
L:
 [[1.   0.   0.   0.  ]
 [0.25 1.   0.   0.  ]
 [0.   0.   1.   0.  ]
 [3.5  0.   0.   1.  ]]
U:
 [[ 4. -1.  0.  3.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
Etapa: 2
L:
 [[ 1.          0.          0.          0.        ]
 [ 0.25        1.          0.          0.        ]
 [ 0.         -0.08253968  1.          0.        ]
 [ 3.5         0.53968254  0.          1.        ]]
U:
 [[ 4.   -1.    0.    3.  ]
 [ 0.   15.75  3.    7.25]
 [ 0.    0.    1.    0.  ]
 [ 0.    0.    0.    1.  ]]
Etapa: 3
L:
 [[ 1.          0.          0.          0.        ]
 [ 0.25        1.          0.          0.        ]
 [ 0.         -0.08253968  1.          0.        ]
 [ 3.5         0.53968254  0.96446701  1.        ]]
U:
 [[ 4.         -1.          0.          3.        ]
 [ 0.         15.75        3.          7.25      ]
 [ 0.          0.         -3.75238095  1.6984127 ]
 [ 0.          0.          0.          1.        ]]
Etapa: 4
L:
 [[ 1.          0.          0.          0.        ]
 [ 0.25        1.          0.          0.        ]
 [ 0.         -0.08253968  1.          0.        ]
 [ 3.5         0.53968254  0.96446701  1.        ]]
U:
 [[ 4.         -1.          0.          3.        ]
 [ 0.         15.75        3.          7.25      ]
 [ 0.          0.         -3.75238095  1.6984127 ]
 [ 0.          0.          0.         13.94923858]]
X:
 [ 0.52510917  0.25545852 -0.41048035 -0.28165939]
```

Figure 8: Doolittle method test.

2.1.5 Cholesky

```python
def cholesky(A, b):
    n = np.array(A.shape[0])
    L = np.eye(n, dtype=complex)
    U = np.eye(n, dtype=complex)
    etapa = 0
    print("Etapa:", etapa)
    print("L:\n", L)
    print("U:\n", U)
    etapa += 1

    for i in range(0, n-1):
        L[i][i] = cmath.sqrt(A[i][i] - L[i, 0:i].dot( np.matrix.transpose(U[0:i,i])))
        U[i][i] = L[i][i]

        for j in range(i+1, n):
            L[j][i] = np.divide((A[j][i] - L[j, 0:i].dot( np.matrix.transpose(U[0:i,i]))), U[i][i])

        for j in range(i+1, n):
            U[i][j] = np.divide((A[i][j] - L[i, 0:i].dot( np.matrix.transpose(U[0:i,j]))), L[i][i])

        print("Etapa:", etapa)
        print("L:\n", L)
        print("U:\n", U)
        etapa += 1

    L[n-1][n-1] = cmath.sqrt(A[n-1][n-1] - L[n-1, 0:n-1].dot( np.matrix.transpose(U[0:n-1,n-1])))
    U[n-1][n-1] = L[n-1][n-1]

    print("Etapa:", etapa)
    print("L:\n", L)
    print("U:\n", U)

    z= sustprog(np.column_stack((L, b)))
    x = sustreg(np.column_stack((U, z)))
    return x
```

Figure 9: Code of the Cholesky method in python.

```
Etapa: 0
L:
 [[1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j]]
U:
 [[1.+0.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 1.+0.j]]
Etapa: 1
L:
 [[2. +0.j 0. +0.j 0. +0.j 0. +0.j]
 [0.5+0.j 1. +0.j 0. +0.j 0. +0.j]
 [0. +0.j 0. +0.j 1. +0.j 0. +0.j]
 [7. +0.j 0. +0.j 0. +0.j 1. +0.j]]
U:
 [[ 2. +0.j -0.5+0.j  0. +0.j  1.5+0.j]
 [ 0. +0.j  1. +0.j  0. +0.j  0. +0.j]
 [ 0. +0.j  0. +0.j  1. +0.j  0. +0.j]
 [ 0. +0.j  0. +0.j  0. +0.j  1. +0.j]]
Etapa: 2
L:
 [[ 2.        +0.j  0.        +0.j  0.        +0.j  0.        +0.j]
 [ 0.5       +0.j  3.96862697+0.j  0.        +0.j  0.        +0.j]
 [ 0.        +0.j -0.32756921+0.j  1.        +0.j  0.        +0.j]
 [ 7.        +0.j  2.14179868+0.j  0.        +0.j  1.        +0.j]]
U:
 [[ 2.        +0.j -0.5       +0.j  0.        +0.j  1.5       +0.j]
 [ 0.        +0.j  3.96862697+0.j  0.75592895+0.j  1.82682829+0.j]
 [ 0.        +0.j  0.        +0.j  1.        +0.j  0.        +0.j]
 [ 0.        +0.j  0.        +0.j  0.        +0.j  1.        +0.j]]
Etapa: 3
L:
 [[ 2.        +0.j          0.        +0.j          0.        +0.j
   0.        +0.j        ]
 [ 0.5       +0.j          3.96862697+0.j          0.        +0.j
   0.        +0.j        ]
 [ 0.        +0.j         -0.32756921+0.j          0.        +1.93710633j
   0.        +0.j        ]
 [ 7.        +0.j          2.14179868+0.j          0.        +1.86827515j
   1.        +0.j        ]]
U:
 [[ 2.        +0.j         -0.5       +0.j          0.        +0.j
   1.5       +0.j        ]
 [ 0.        +0.j          3.96862697+0.j          0.75592895+0.j
   1.82682829+0.j        ]
 [ 0.        +0.j          0.        +0.j          0.        +1.93710633j
   0.        -0.87677825j]
 [ 0.        +0.j          0.        +0.j          0.        +0.j
   1.        +0.j        ]]
Etapa: 4
L:
 [[ 2.        +0.j          0.        +0.j          0.        +0.j
   0.        +0.j        ]
 [ 0.5       +0.j          3.96862697+0.j          0.        +0.j
   0.        +0.j        ]
 [ 0.        +0.j         -0.32756921+0.j          0.        +1.93710633j
   0.        +0.j        ]
 [ 7.        +0.j          2.14179868+0.j          0.        +1.86827515j
   3.73486795+0.j        ]]
U:
 [[ 2.        +0.j         -0.5       +0.j          0.        +0.j
   1.5       +0.j        ]
 [ 0.        +0.j          3.96862697+0.j          0.75592895+0.j
   1.82682829+0.j        ]
 [ 0.        +0.j          0.        +0.j          0.        +1.93710633j
   0.        -0.87677825j]
 [ 0.        +0.j          0.        +0.j          0.        +0.j
   3.73486795+0.j        ]]
X:
 [ 0.52510917+0.j  0.25545852+0.j -0.41048035-0.j -0.28165939+0.j]
```

Figure 10: Cholesky method test (Can not handle negative roots yet).

2.1.6 Jacobi

```python
def jacobi(A, b, x0, tol, Nmax):
    det = np.linalg.det(A) # Determinant of matrix
    if det == 0:
        print("Determinant of matrix is 0")
        return
    D = np.diag(np.diag(A))
    L = -np.tril(A)+D
    U = -np.triu(A)+D

    T = np.linalg.inv(D).dot(L+U)
    C = np.linalg.inv(D).dot(b)

    xant = x0
    E = 1000
    cont = 0

    while E>tol and cont<Nmax:
        np.set_printoptions(precision=8, suppress=True)
        print("{i:2d} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        xact = T.dot(xant) + C
        E = np.linalg.norm(xant-xact)
        xant = xact
        cont += 1

    if E>tol or cont == Nmax:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        print("This method can not get solution to the matrix")
    else:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
```

Figure 11: Code of the Jacobi method in python.

```
0 | X0 = [0 0 0 0] | E = 1.0E+03
1 | X1 = [ 0.25        0.06451613 -0.25        0.03333333] | E = 3.6E-01
2 | X2 = [ 0.24112903  0.07956989 -0.26180108 -0.11075269] | E = 1.5E-01
3 | X3 = [ 0.35295699  0.15679327 -0.3063172  -0.1099086 ] | E = 1.4E-01
4 | X4 = [ 0.37162977  0.15775893 -0.33118268 -0.17793329] | E = 7.5E-02
5 | X5 = [ 0.4228897   0.19647642 -0.35020331 -0.18846589] | E = 6.8E-02
6 | X6 = [ 0.44046853  0.20228693 -0.36568296 -0.22010815] | E = 4.0E-02
7 | X7 = [ 0.46565284  0.22048036 -0.37627299 -0.230312  ] | E = 3.4E-02
8 | X8 = [ 0.47785409  0.22617175 -0.38499192 -0.24580292] | E = 2.2E-02
9 | X9 = [ 0.49089513  0.23506742 -0.39110162 -0.25302666] | E = 1.8E-02
10 | X10 = [ 0.49853685  0.23913697 -0.39597924 -0.2610024 ] | E = 1.3E-02
11 | X11 = [ 0.50553605  0.24370452 -0.39949518 -0.26557197] | E = 1.0E-02
12 | X12 = [ 0.51010511  0.24629195 -0.40223626 -0.26983392] | E = 7.3E-03
13 | X13 = [ 0.51394843  0.24872742 -0.40424921 -0.27258013] | E = 5.7E-03
14 | X14 = [ 0.51661695  0.25028647 -0.40579595 -0.27491378] | E = 4.2E-03
15 | X15 = [ 0.51875696  0.25161814 -0.40694439 -0.27652205] | E = 3.2E-03
16 | X16 = [ 0.52029607  0.25253243 -0.40781946 -0.27781923] | E = 2.4E-03
17 | X17 = [ 0.52149753  0.25327201 -0.40847333 -0.2787482 ] | E = 1.8E-03
18 | X18 = [ 0.52237915  0.25380052 -0.40896916 -0.27947574] | E = 1.4E-03
19 | X19 = [ 0.52305693  0.25421511 -0.409341   -0.2800083 ] | E = 1.0E-03
20 | X20 = [ 0.52356     0.25451822 -0.40962219 -0.28041849] | E = 7.7E-04
21 | X21 = [ 0.52394342  0.2547519  -0.40983351 -0.28072252] | E = 5.8E-04
22 | X22 = [ 0.52422986  0.25492498 -0.40999306 -0.28095448] | E = 4.4E-04
23 | X23 = [ 0.52444711  0.25505711 -0.4101131  -0.28112764] | E = 3.3E-04
24 | X24 = [ 0.52461001  0.2551557  -0.41020366 -0.28125904] | E = 2.5E-04
25 | X25 = [ 0.52473321  0.25523054 -0.41027184 -0.28135753] | E = 1.9E-04
26 | X26 = [ 0.52482578  0.25528662 -0.41032324 -0.28143204] | E = 1.4E-04
27 | X27 = [ 0.52489568  0.25532905 -0.41036196 -0.28148802] | E = 1.1E-04
28 | X28 = [ 0.52494828  0.25536092 -0.41039115 -0.28153029] | E = 8.0E-05
29 | X29 = [ 0.52498795  0.255385   -0.41041313 -0.28156209] | E = 6.0E-05
30 | X30 = [ 0.52501782  0.25540311 -0.4104297  -0.28158609] | E = 4.6E-05
31 | X31 = [ 0.52504034  0.25541677 -0.41044218 -0.28160415] | E = 3.4E-05
32 | X32 = [ 0.5250573   0.25542706 -0.41045159 -0.28161777] | E = 2.6E-05
33 | X33 = [ 0.52507009  0.25543481 -0.41045868 -0.28162802] | E = 1.9E-05
34 | X34 = [ 0.52507972  0.25544065 -0.41046402 -0.28163576] | E = 1.5E-05
35 | X35 = [ 0.52508698  0.25544506 -0.41046805 -0.28164158] | E = 1.1E-05
36 | X36 = [ 0.52509245  0.25544837 -0.41047108 -0.28164597] | E = 8.3E-06
37 | X37 = [ 0.52509657  0.25545087 -0.41047336 -0.28164928] | E = 6.3E-06
38 | X38 = [ 0.52509968  0.25545276 -0.41047509 -0.28165177] | E = 4.7E-06
39 | X39 = [ 0.52510202  0.25545418 -0.41047638 -0.28165365] | E = 3.6E-06
40 | X40 = [ 0.52510378  0.25545525 -0.41047736 -0.28165506] | E = 2.7E-06
41 | X41 = [ 0.52510511  0.25545605 -0.4104781  -0.28165613] | E = 2.0E-06
42 | X42 = [ 0.52510611  0.25545666 -0.41047865 -0.28165693] | E = 1.5E-06
43 | X43 = [ 0.52510686  0.25545712 -0.41047907 -0.28165754] | E = 1.1E-06
44 | X44 = [ 0.52510743  0.25545746 -0.41047939 -0.28165799] | E = 8.7E-07
45 | X45 = [ 0.52510786  0.25545772 -0.41047962 -0.28165834] | E = 6.5E-07
46 | X46 = [ 0.52510818  0.25545792 -0.4104798  -0.2816586 ] | E = 4.9E-07
47 | X47 = [ 0.52510843  0.25545806 -0.41047994 -0.28165879] | E = 3.7E-07
48 | X48 = [ 0.52510861  0.25545818 -0.41048004 -0.28165894] | E = 2.8E-07
49 | X49 = [ 0.52510875  0.25545826 -0.41048012 -0.28165905] | E = 2.1E-07
50 | X50 = [ 0.52510885  0.25545832 -0.41048017 -0.28165913] | E = 1.6E-07
51 | X51 = [ 0.52510893  0.25545837 -0.41048022 -0.2816592 ] | E = 1.2E-07
52 | X52 = [ 0.52510899  0.25545841 -0.41048025 -0.28165924] | E = 9.0E-08
```

Figure 12: Jacobi method test.

2.1.7 Gauss-Seidel

```python
def gseidel(A, b, x0, tol, Nmax):
    det = np.linalg.det(A) # Determinant of matrix
    if det == 0:
        print("Determinant of matrix is 0")
        return
    D = np.diag(np.diag(A))
    L = -np.tril(A)+D
    U = -np.triu(A)+D


    T = np.linalg.inv(D-L).dot(U)
    C = np.linalg.inv(D-L).dot(b)

    xant = x0
    E = 1000
    cont = 0

    while E>tol and cont<Nmax:
        np.set_printoptions(precision=8, suppress=True)
        print("{i:2d} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        xact = T.dot(xant) + C
        E = np.linalg.norm(xant-xact)
        xant = xact
        cont += 1

    if E>tol or cont == Nmax:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        print("This method can not get solution to the matrix")
    else:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
```

Figure 13: Code of the Gauss-Seidel method in python.

```
 0 | X0 = [0 0 0 0] | E = 1.0E+03
 1 | X1 = [ 0.25        0.0483871  -0.26572581 -0.1091129 ] | E = 3.8E-01
 2 | X2 = [ 0.34393145  0.15007414 -0.32878014 -0.17409904] | E = 1.7E-01
 3 | X3 = [ 0.41809282  0.19103484 -0.35996356 -0.21761336] | E = 1.0E-01
 4 | X4 = [ 0.46096873  0.21676315 -0.3802917  -0.24326538] | E = 6.0E-02
 5 | X5 = [ 0.48663982  0.23228118 -0.39238936 -0.25863807] | E = 3.6E-02
 6 | X6 = [ 0.50204885  0.24156283 -0.39963339 -0.26785883] | E = 2.1E-02
 7 | X7 = [ 0.51128483  0.24712813 -0.40397782 -0.27338613] | E = 1.3E-02
 8 | X8 = [ 0.51682163  0.25046457 -0.40658217 -0.27669967] | E = 7.7E-03
 9 | X9 = [ 0.52014089  0.25246471 -0.40814344 -0.2786861 ] | E = 4.6E-03
10 | X10 = [ 0.52213075  0.25366376 -0.4090794  -0.27987694] | E = 2.8E-03
11 | X11 = [ 0.52332364  0.25438258 -0.4096405  -0.28059083] | E = 1.7E-03
12 | X12 = [ 0.52403877  0.25481351 -0.40997687 -0.2810188 ] | E = 1.0E-03
13 | X13 = [ 0.52446748  0.25507184 -0.41017852 -0.28127536] | E = 6.0E-04
14 | X14 = [ 0.52472448  0.25522671 -0.41029941 -0.28142917] | E = 3.6E-04
15 | X15 = [ 0.52487856  0.25531955 -0.41037188 -0.28152138] | E = 2.1E-04
16 | X16 = [ 0.52497092  0.25537521 -0.41041532 -0.28157665] | E = 1.3E-04
17 | X17 = [ 0.52502629  0.25540857 -0.41044137 -0.28160979] | E = 7.7E-05
18 | X18 = [ 0.52505948  0.25542858 -0.41045698 -0.28162965] | E = 4.6E-05
19 | X19 = [ 0.52507938  0.25544057 -0.41046634 -0.28164156] | E = 2.8E-05
20 | X20 = [ 0.52509131  0.25544776 -0.41047195 -0.2816487 ] | E = 1.7E-05
21 | X21 = [ 0.52509847  0.25545206 -0.41047531 -0.28165298] | E = 1.0E-05
22 | X22 = [ 0.52510275  0.25545465 -0.41047733 -0.28165555] | E = 6.0E-06
23 | X23 = [ 0.52510532  0.2554562  -0.41047854 -0.28165709] | E = 3.6E-06
24 | X24 = [ 0.52510686  0.25545713 -0.41047926 -0.28165801] | E = 2.1E-06
25 | X25 = [ 0.52510779  0.25545768 -0.4104797  -0.28165856] | E = 1.3E-06
26 | X26 = [ 0.52510834  0.25545802 -0.41047996 -0.28165889] | E = 7.7E-07
27 | X27 = [ 0.52510867  0.25545822 -0.41048012 -0.28165909] | E = 4.6E-07
28 | X28 = [ 0.52510887  0.25545834 -0.41048021 -0.28165921] | E = 2.8E-07
29 | X29 = [ 0.52510899  0.25545841 -0.41048027 -0.28165928] | E = 1.7E-07
30 | X30 = [ 0.52510906  0.25545845 -0.4104803  -0.28165932] | E = 1.0E-07
```

Figure 14: Gauss-Seidel method test.

2.1.8 SOR

```python
def sor(A, b, x0, w, tol, Nmax):
    det = np.linalg.det(A) # Determinant of matrix
    if det == 0:
        print("Determinant of matrix is 0")
        return
    D = np.diag(np.diag(A))
    L = -np.tril(A)+D
    U = -np.triu(A)+D


    T = np.linalg.inv(D -(w*L)).dot((1-w)*D + w*U)
    C = w*np.linalg.inv(D- (w*L)).dot(b)

    xant = x0
    E = 1000
    cont = 0

    while E>tol and cont<Nmax:
        np.set_printoptions(precision=8, suppress=True)
        print("{i:2d} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        xact = T.dot(xant) + C
        E = np.linalg.norm(xant-xact)
        xant = xact
        cont += 1

    if E>tol or cont == Nmax:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
        print("This method can not get solution to the matrix")
    else:
        print("{i} | X{i} = {xant} | E = {Err:.1E}".format(i=cont, xant=xant, Err=E))
```

Figure 15: Code of the SOR method in python.

```
 0 | X0 = [0 0 0 0] | E = 1.0E+03
 1 | X1 = [ 0.375       0.06048387 -0.40448589 -0.26806956] | E = 6.2E-01
 2 | X2 = [ 0.5117597   0.34197623 -0.45004916 -0.30469599] | E = 3.2E-01
 3 | X3 = [ 0.59014422  0.23522845 -0.39033638 -0.30859371] | E = 1.5E-01
 4 | X4 = [ 0.51530648  0.28152633 -0.4443708  -0.27123634] | E = 1.1E-01
 5 | X5 = [ 0.52806002  0.24390875 -0.38360511 -0.28336154] | E = 7.4E-02
 6 | X6 = [ 0.52121751  0.25512531 -0.42445767 -0.27939858] | E = 4.3E-02
 7 | X7 = [ 0.52438664  0.25800267 -0.40379938 -0.28225196] | E = 2.1E-02
 8 | X8 = [ 0.52709114  0.25251377 -0.41262971 -0.28222923] | E = 1.1E-02
 9 | X9 = [ 0.52365497  0.25813679 -0.41094639 -0.2810727 ] | E = 6.9E-03
10 | X10 = [ 0.5261806   0.25369679 -0.40914648 -0.28212891] | E = 5.5E-03
11 | X11 = [ 0.52444102  0.25638029 -0.41179033 -0.28131836] | E = 4.2E-03
12 | X12 = [ 0.52540526  0.25508527 -0.40950273 -0.28184609] | E = 2.8E-03
13 | X13 = [ 0.5250312   0.2555134  -0.41107293 -0.28158444] | E = 1.7E-03
14 | X14 = [ 0.52508442  0.25554748 -0.41019651 -0.2816734 ] | E = 8.8E-04
15 | X15 = [ 0.52517067  0.25533652 -0.41056857 -0.28167376] | E = 4.4E-04
16 | X16 = [ 0.52504884  0.25556209 -0.41049266 -0.2816371 ] | E = 2.7E-04
17 | X17 = [ 0.5251531   0.25538879 -0.41043101 -0.28167892] | E = 2.2E-04
18 | X18 = [ 0.52508303  0.2554967  -0.41053169 -0.28164601] | E = 1.7E-04
19 | X19 = [ 0.52512151  0.25544277 -0.41044149 -0.28166689] | E = 1.1E-04
20 | X20 = [ 0.52510554  0.25546126 -0.41050422 -0.28165617] | E = 6.8E-05
21 | X21 = [ 0.52510839  0.25546165 -0.41046862 -0.28166007] | E = 3.6E-05
22 | X22 = [ 0.5251115   0.25545384 -0.41048422 -0.2816599 ] | E = 1.8E-05
23 | X23 = [ 0.52510682  0.2554626  -0.41048062 -0.28165854] | E = 1.1E-05
24 | X24 = [ 0.52511092  0.25545573 -0.41047851 -0.28166015] | E = 8.4E-06
25 | X25 = [ 0.52510811  0.25546007 -0.41048235 -0.28165885] | E = 6.6E-06
26 | X26 = [ 0.52510968  0.25545785 -0.41047881 -0.28165969] | E = 4.5E-06
27 | X27 = [ 0.52510901  0.25545865 -0.41048131 -0.28165925] | E = 2.7E-06
28 | X28 = [ 0.52510915  0.25545862 -0.41047987 -0.28165942] | E = 1.5E-06
29 | X29 = [ 0.52510926  0.25545834 -0.41048052 -0.28165941] | E = 7.2E-07
30 | X30 = [ 0.52510908  0.25545868 -0.41048035 -0.28165936] | E = 4.2E-07
31 | X31 = [ 0.52510924  0.2554584  -0.41048028 -0.28165942] | E = 3.3E-07
32 | X32 = [ 0.52510913  0.25545858 -0.41048043 -0.28165937] | E = 2.6E-07
33 | X33 = [ 0.52510919  0.25545849 -0.41048029 -0.2816594 ] | E = 1.8E-07
34 | X34 = [ 0.52510916  0.25545852 -0.41048039 -0.28165938] | E = 1.1E-07
35 | X35 = [ 0.52510917  0.25545852 -0.41048033 -0.28165939] | E = 5.9E-08
```

Figure 16: SOR method test

2.1.9 Vandermonde

```python
def vandermonde(X, Y):
    n = X.size
    A = np.zeros((n, n)) # (shape)
    X = np.matrix.transpose(X)

    for i in range(n):
        A[:,i] = np.matrix.transpose(np.power(X, (n-i-1)))

    print("A:\n", A)
    coef = gausstot(A, np.matrix.transpose(Y))
    print("Coef:\n", coef)
    return coef
```

Figure 17: Code of vandermonde method in python.

```
A:
 [[-1.  1. -1.  1.]
 [ 0.  0.  0.  1.]
 [27.  9.  3.  1.]
 [64. 16.  4.  1.]]
Etapa 0
[[-1.   1.  -1.   1.  15.5]
 [ 0.   0.   0.   1.   3. ]
 [27.   9.   3.   1.   8. ]
 [64.  16.   4.   1.   1. ]]

Cambio de fila
[[64.  16.   4.   1.   1. ]
 [ 0.   0.   0.   1.   3. ]
 [27.   9.   3.   1.   8. ]
 [-1.   1.  -1.   1.  15.5]]

Etapa 1
[[64.       16.        4.        1.        1.      ]
 [ 0.        0.        0.        1.        3.      ]
 [ 0.        2.25      1.3125    0.578125  7.578125]
 [ 0.        1.25     -0.9375    1.015625 15.515625]]

Cambio de fila
[[64.       16.        4.        1.        1.      ]
 [ 0.        2.25      1.3125    0.578125  7.578125]
 [ 0.        0.        0.        1.        3.      ]
 [ 0.        1.25     -0.9375    1.015625 15.515625]]

Etapa 2
[[64.         16.          4.          1.          1.        ]
 [ 0.          2.25        1.3125      0.578125    7.578125  ]
 [ 0.          0.          0.          1.          3.        ]
 [ 0.          0.         -1.66666667  0.69444444 11.30555556]]

Cambio de fila
[[64.         16.          4.          1.          1.        ]
 [ 0.          2.25        1.3125      0.578125    7.578125  ]
 [ 0.          0.         -1.66666667  0.69444444 11.30555556]
 [ 0.          0.          0.          1.          3.        ]]

Etapa 3
[[64.         16.          4.          1.          1.        ]
 [ 0.          2.25        1.3125      0.578125    7.578125  ]
 [ 0.          0.         -1.66666667  0.69444444 11.30555556]
 [ 0.          0.          0.          1.          3.        ]]

Despues de aplicar sustitucion regresiva

X antes del cambio de columnas:
[-1.14166667  5.825      -5.53333333  3.        ]
X despues del cambio de columnas:
[-1.14166667  5.825      -5.53333333  3.        ]
Coef:
 [-1.14166667  5.825      -5.53333333  3.        ]
```

Figure 18: Vandermonde method test.

2.1.10 Newton

```python
def difdivididas(X, Y):
    n = X.size
    D = np.zeros((n, n)) # (shape)
    X = np.matrix.transpose(X)

    D[:,0] = np.matrix.transpose(Y)
    for i in range(1, n):
        aux0 = D[i-1:n, i-1]
        aux = np.diff(aux0)
        aux2 = X[i:n] - X[0:n-i]
        D[i:n, i] = np.divide(aux, aux2)

    coef = np.diag(D)
    return coef
```

Figure 19: Code of the Newton method in python.

```
Etapa: 0
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

Etapa: 1
[[15.5  0.   0.   0. ]
 [ 3.   0.   0.   0. ]
 [ 8.   0.   0.   0. ]
 [ 1.   0.   0.   0. ]]

Etapa: 2
[[ 15.5         0.          0.          0.        ]
 [  3.        -12.5        0.          0.        ]
 [  8.          1.66666667  0.          0.        ]
 [  1.         -7.         0.          0.        ]]

Etapa: 3
[[ 15.5         0.          0.          0.        ]
 [  3.        -12.5        0.          0.        ]
 [  8.          1.66666667  3.54166667  0.        ]
 [  1.         -7.        -2.16666667  0.        ]]

Etapa: 4
[[ 15.5         0.          0.          0.        ]
 [  3.        -12.5        0.          0.        ]
 [  8.          1.66666667  3.54166667  0.        ]
 [  1.         -7.        -2.16666667 -1.14166667]]

Coef:
 [ 15.5       -12.5         3.54166667 -1.14166667]
```

Figure 20: Newton method test.

2.1.11 Lagrange

```python
def lagrange(X, Y):
    n = X.size
    L = np.zeros((n, n)) # (shape)

    for i in range(n):
        aux0 = np.setdiff1d(X, X[i])
        aux = [1, -aux0[0]]
        for j in range(1, n-1):
            aux = np.convolve(aux, [1, -aux0[j]])
        L[i,:] = aux / np.polyval(aux, X[i])

    coef = Y.dot(L)
    return L, coef
```

Figure 21: Code of the lagrange method in python.

```
Etapa: 0
L:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

Etapa: 1
L:
 [[-0.05  0.35 -0.6  -0.  ]
 [ 0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.  ]]

Etapa: 2
L:
 [[-0.05        0.35       -0.6        -0.        ]
 [ 0.08333333 -0.5         0.41666667  1.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]]

Etapa: 3
L:
 [[-0.05        0.35       -0.6        -0.        ]
 [ 0.08333333 -0.5         0.41666667  1.        ]
 [-0.08333333  0.25        0.33333333 -0.        ]
 [ 0.          0.          0.          0.        ]]

Etapa: 4
L:
 [[-0.05        0.35       -0.6        -0.        ]
 [ 0.08333333 -0.5         0.41666667  1.        ]
 [-0.08333333  0.25        0.33333333 -0.        ]
 [ 0.05       -0.1        -0.15        0.        ]]

Etapa: 5
L:
 [[-0.05        0.35       -0.6        -0.        ]
 [ 0.08333333 -0.5         0.41666667  1.        ]
 [-0.08333333  0.25        0.33333333 -0.        ]
 [ 0.05       -0.1        -0.15        0.        ]]
Coef:
 [-1.14166667  5.825      -5.53333333  3.        ]
```

Figure 22: Lagrange method test.

2.1.12 Line plotter

```python
def trazlin(X, Y):
    n = X.size
    m = 2*(n-1)
    A = np.zeros((m, m)) # (shape)
    b = np.zeros(m)
    coef = np.zeros((n-1, 2))

    for i in range(n - 1):
        A[i+1, [2*i, 2*i+1]] = np.array([X[i+1], 1])
        b[i+1] = Y[i+1]

    A[0, [0, 1]] = np.array([X[0], 1])
    b[0] = Y[0]
    print(A)
    print(b)

    for i in range(1, n-1):
        A[n+i-1, 2*i-2:2*i+2] = np.array([X[i], 1, -X[i], -1])
        b[n+i-1] = 0

    Saux = gausstot(A, b)

    for i in range(n-1):
        coef[i,:] = Saux[[2*i, 2*i+1]]
    return coef
```

Figure 23: Code of Line plotter method in python.

```
Etapa: 0
A:
 [[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
b:
 [0. 0. 0. 0. 0. 0.]

Etapa: 1
A:
 [[-1.  1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.]
 [ 0.  0.  3.  1.  0.  0.]
 [ 0.  0.  0.  0.  4.  1.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]]
b:
 [15.5  3.   8.   1.   0.   0. ]

Etapa: 2
A:
 [[-1.  1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.]
 [ 0.  0.  3.  1.  0.  0.]
 [ 0.  0.  0.  0.  4.  1.]
 [ 0.  1.  0. -1.  0.  0.]
 [ 0.  0.  3.  1. -3. -1.]]
b:
 [15.5  3.   8.   1.   0.   0. ]

INICIO - GAUSSTOT
```

Figure 24: First image of the line plotter method test.

```
X despues del cambio de columnas:
[-12.5          3.           1.66666667   3.          -7.
  29.        ]
TERMINA - GAUSSTOT
Coef:
 [[-12.5          3.        ]
 [  1.66666667   3.        ]
 [ -7.          29.        ]]
```

Figure 25: Second image of the line plotter method test.

2.1.13 Cuadratic plotter

```python
def trazcuad(X, Y):
    n = X.size
    m = 3*(n-1)
    A = np.zeros((m, m)) # (shape)
    b = np.zeros(m)
    coef = np.zeros((n-1, 3))

    # Condiciones de interpolacion
    for i in range(n - 1):
        A[i+1, 3*i:3*i+3] = np.array([X[i+1]**2, X[i+1], 1])
        b[i+1] = Y[i+1]

    A[0, 0:3] = np.array([X[0]**2, X[0], 1])
    b[0] = Y[0]

    # Condiciones de continuidad
    for i in range(1, n-1):
        A[n+i-1, 3*i-3:3*i+3] = np.array([X[i]**2, X[i], 1, -X[i]**2, -X[i], -1])
        b[n+i-1] = 0

    # Condiciones de suavidad
    for i in range(1, n-1):
        A[2*n+i-3, 3*i-3:3*i+3] = np.array([2*X[i], 1, 0, -2*X[i], -1, 0])
        b[2*n+i-3] = 0

    A[m-1, 0] = 2
    b[m-1] = 0

    Saux = gausstot(A, b)

    for i in range(n-1):
        coef[i,:] = Saux[3*i:3*i+3]
    return coef
```

Figure 26: Code of a cuadratic plotter method in python.

```
Etapa: 0
A:
 [[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]
b:
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]

Etapa: 1
A:
 [[ 1. -1.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 16.  4.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]]
b:
 [15.5  3.   8.   1.   0.   0.   0.   0.   0. ]

Etapa: 2
A:
 [[ 1. -1.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 16.  4.  1.]
 [ 0.  0.  1.  0.  0. -1.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1. -9. -3. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]]
b:
 [15.5  3.   8.   1.   0.   0.   0.   0.   0. ]

Etapa: 3
A:
 [[ 1. -1.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 16.  4.  1.]
 [ 0.  0.  1.  0.  0. -1.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1. -9. -3. -1.]
 [ 0.  1.  0.  0. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  6.  1.  0. -6. -1.  0.]
 [ 2.  0.  0.  0.  0.  0.  0.  0.  0.]]
b:
 [15.5  3.   8.   1.   0.   0.   0.   0.   0. ]

INICIO - GAUSSTOT
```

Figure 27: First image of a cuadratic plotter method test.

```
    12.5          3.          152.83333333 -245.        ]
X despues del cambio de columnas:
[   0.          -12.5           3.            4.72222222  -12.5
    3.          -22.83333333  152.83333333 -245.        ]
TERMINA - GAUSSTOT
Etapa: 4
A:
 [[ 1. -1.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 16.  4.  1.]
 [ 0.  0.  1.  0.  0. -1.  0.  0.  0.]
 [ 0.  0.  0.  9.  3.  1. -9. -3. -1.]
 [ 0.  1.  0.  0. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  6.  1.  0. -6. -1.  0.]
 [ 2.  0.  0.  0.  0.  0.  0.  0.  0.]]
b:
 [15.5  3.   8.   1.   0.   0.   0.   0.   0. ]

Coef:
 [[   0.          -12.5           3.        ]
 [   4.72222222  -12.5           3.        ]
 [ -22.83333333  152.83333333 -245.        ]]
```

Figure 28: Second image of a cuadratic plotter method test.