Ex.4

Fie $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ un polinom de grad n. Să se construiască un algoritm și să se implementeze în Python care să calculeze valoarea polinomului $P_n(x)$ folosind înmulțirea imbricată. Indicație: Se vor deduce, pe rând, formulele pentru $P_2, P_3, P_4, ..., P_n$ și se va observa o regulă.

Exemplu:

$$y = P_2(x) = a_2 x^2 + a_1 x + a_0 = ((a_2 x + a_1)x + a_0)$$

se inițializează $y = a_2$ și se actualizează de două ori valoarea y evaluând rând pe rând parantezele, pornind de la cea inferioară către cea superioară.

Procedura va avea ca parametri de intrare $x, a_0, a_2, ..., a_n$, iar ca parametru de ieșire valoarea polinomului $y = P_n(x)$.

Să se evalueze și să se afișeze $P_3(r), P_3(v)$, dacă $r = 3$ și $v = [1,2,3,4]$ este un vector.

Codul scris:

```
def polinom(x, *lista_coeficienti):
    rez = 0
    for ind, elem in enumerate(reversed(lista_coeficienti)):
        rez += elem
        if ind < len(lista_coeficienti) - 1:
            rez *= x

    return rez

# primul parametru este valoarea lui x in care dorim sa fie evaluata
functia
# pentru urmatorii parametri = 1, 2, 3, 4
# se va calcula 4x^3 + 3x^2 + 2x + 1
if __name__ == '__main__':
    print(polinom(3, 1, 2, 3, 4))
```

Output la rulare:

```
rotak@rtkUBUNTU:~/aaa_work/Calcul Numeric/Tema1/p4$ python main.py
142
```

Ex 6.

Fie ecuaţia $x^3 - 7x^2 + 14x - 6 = 0$

a. Să se construiască în Python procedura cu sintaxa **Bisectie**(f, a, b, ε) (ε fiind un parmetru impus cu valoarea implicită 10^{-3}), care implementează algoritmul metodei bisecţiei. Procedura **Bisectie** va returna, atât soluţia aproximativă x_{aprox} cât şi numărul de iteraţii N necesar pentru obţinerea soluţiei aproximative cu eroarea ε.

b. Să se construiască în Python graficul funcţiei $f(x) = x^3 - 7x^2 + 14x - 6$ pe intervalul $[0, 4]$. Să se calculeze toate rădăcinile de pe intervalul dat şi să se construiască rădăcinile pe grafic.

c. Să se afișeze un tabel cu următoarele coloane

```
+-----+-----+----------+---------------------+---------------+
|  k  | x_k | |f(x_k)| | |x_k - x_k-1|/|x_k| | (b-a)/2^(k+1) |
+-----+-----+----------+---------------------+---------------+
| ... | ... |   ...    |         ...         |      ...      |
| ... | ... |   ...    |         ...         |      ...      |
| ... | ... |   ...    |         ...         |      ...      |
| ... | ... |   ...    |         ...         |      ...      |
+-----+-----+----------+---------------------+---------------+
```

care va fi completat cu valorile mărimilor din prima linie la fiecare iterație.

Pentru construcția tabelelor se va folosi următoarea secvență

```
from prettytable import PrettyTable
Tabel_date = PrettyTable(['k','x_k','|f(x_k)|','|x_k - x_k-1|/|x_k|'])
for k in range(1,len(x)):
  Tabel_date.add_row([   '...','...' ,'...' , '...'])

```

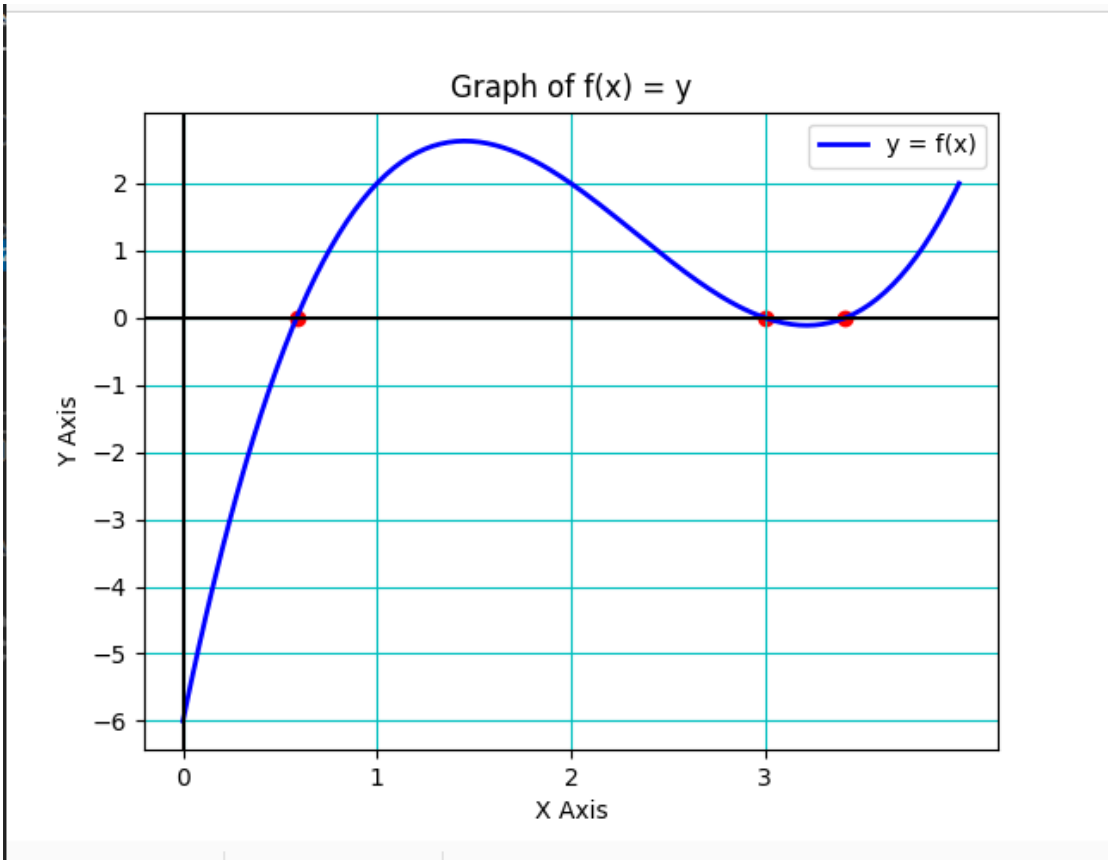
Codul scris:

```
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable

# Formula pentru eroare relativa
relative_error = lambda prevx, x : abs((x - prevx) / x)

fun = lambda x : x**3 - 7*x**2 + 14 * x - 6

def Bisectie(f, a, b, eps = 10**-3, out_ptable = None):
    counter = 0
    xold = -1

    # Daca din start valorile sunt pe aceeasi parte a axei Ox, ori nu
avem o radacina, ori avem mai multe.
    # it's not good either way, returnam none
    if f(a) * f(b) > 0:
```

```
        return None

    while True:
        x = a + (b - a) / 2

        # adaugam datele la PrettyTable, daca acesta exista
        if out_ptable != None:
            y = fun(x)
            err = relative_error(xold, x) if counter != 0 else 'nil'
            log = (float)(b - a) / (2 ** (counter))

            out_ptable.add_row([counter + 1, x, y, err, log])

        if counter != 0 and relative_error(xold, x) < eps:
            return {'x':x, 'it':counter}

        if f(a) * f(x) < 0:
            b = x
        else:
            a = x

        counter += 1
        xold = x

if __name__ == '__main__':
    result_points = []
    result_tables = []
    start_interval = 0
    end_interval = 4
    jump_interval = 0.05

    for i in np.linspace(start_interval, end_interval,
(int)((end_interval - start_interval) / jump_interval)):
        table = PrettyTable(['k', 'x_k', 'f(x_k)', 'rel err',
'(b-a)/2^(k-1)'])
        pt = Bisectie(fun, i, i + jump_interval, 10**-6, table)
        if pt:
            result_points.append(pt['x'])
            result_tables.append(table)

    print("Radacinile polinomului sunt:")
    print(result_points)

    for (index, table) in enumerate(result_tables):
        print(f"Tabel pentru x = {result_points[index]}:")
        print(table)
```

```
        print("\n")

    fig = plt.figure(1)
    ax = plt.axes()
    x = np.linspace(start_interval, end_interval, (end_interval -
start_interval) * 50)
    y = fun(x)
    ax.plot(x, y, linestyle='-', lw = 2, color = 'b', label = 'y =
f(x)')
    ax.scatter(result_points, np.zeros(len(result_points)), color="r")
    ax.grid(True, color = 'c')
    plt.xticks(np.arange(start_interval,end_interval,(int)((end_interval
- start_interval) / 4)))
    plt.yticks(np.arange(min(y),max(y),(int)((end_interval -
start_interval) / 4)))
    ax.legend(loc='best')
    plt.xlabel('X Axis')
    plt.ylabel('Y Axis')
    plt.title('Graph of f(x) = y')
    ax.axhline(y=0,color='k')
    ax.axvline(x=0,color='k')
    plt.show()
```

Graficul Trasat



Rezultatele Numerice Afișate

```
Radacinile polinomului sunt:
[0.5857862593252446, 2.999998937679242, 3.4142159763770765]

Tabel pentru x = 0.5857862593252446:
+----+--------------------+--------------------------+-----------------------+-----------------------+
| k  |        x_k         |          f(x_k)          |        rel err        |     (b-a)/2^(k-1)     |
+----+--------------------+--------------------------+-----------------------+-----------------------+
| 1  | 0.5819620253164557 |   -0.02619145623495278   |          nil          |  0.05000000000000044  |
| 2  | 0.5944620253164556 |   0.058846679618175024   |  0.021027415491083237 |  0.012500000000000011 |
| 3  | 0.5882120253164557 |   0.016532118094893633   |  0.010625420309347643 |  0.003124999999999989 |
| 4  | 0.5850870253164557 |  -0.004778450916474775   | 0.0053410857954160175 | 0.0007812499999999972 |
| 5  | 0.5866495253164556 |   0.005889626683506677   | 0.0026634300933884737 | 0.0001953124999999584 |
| 6  | 0.5858682753164557 |  0.0005587875876029713   | 0.0013334908765591802 | 4.8828124999997224e-05 |
| 7  | 0.5854776503164557 |  -0.002109031559600183   | 0.0006671902843581503 | 1.2207031249999306e-05 |
| 8  | 0.5856729628164556 |  -0.0007749219821420184  | 0.0003334838935720898 | 3.0517578124998265e-06 |
| 9  | 0.5857706190664557 | -0.0001080171990999191903 | 0.0001667141485444545 9 | 7.629394531251735e-07 |
| 10 | 0.5858194471914557 |  0.00022539769344476213  | 8.335012644964098e-05 | 1.9073486328129337e-07 |
| 11 | 0.5857950331289556 |  5.8693372015028444e-05  | 4.167680010819477e-05 | 4.768371582037755e-08 |
| 12 | 0.5857828260977056 | -2.4661132327352675e-05  | 2.0838834302063232e-05 | 1.1920928955067283e-08 |
| 13 | 0.5857889296133306 |  1.701631514716695e-05   | 1.0419308587909013e-05 | 2.9802322387803733e-09 |
| 14 | 0.5857858778555181 | -3.8223597638165074e-06  | 5.2096814346885265e-06 | 7.450580596950933e-10 |
| 15 | 0.5857874037344244 |  6.596989897467154e-06   | 2.604833932261538e-06  | 1.8626451492716146e-10 |
| 16 | 0.5857866407949712 |  1.387318118162284e-06   | 1.302418662422932e-06  | 4.656612873348443e-11 |
| 17 | 0.5857862593252446 | -1.2175200598818492e-06  | 6.512097552853351e-07  | 1.1641532183371108e-11 |
+----+--------------------+--------------------------+-----------------------+-----------------------+


Tabel pentru x = 2.999998937679242:
+----+--------------------+--------------------------+-----------------------+-----------------------+
| k  |        x_k         |          f(x_k)          |        rel err        |     (b-a)/2^(k-1)     |
+----+--------------------+--------------------------+-----------------------+-----------------------+
| 1  | 3.0123417721518986 |   -0.01203525358161528   |          nil          |  0.04999999999999982  |
| 2  | 2.999841772151899  |  0.0001582779162419 0013 |  0.004166886439158094 |  0.012499999999999956 |
| 3  | 3.0060917721518985 |  -0.006017326712232318   | 0.0020791115088032086 | 0.0031249999999999334 |
| 4  | 3.0029667721518987 |  -0.0029491425651571035  | 0.0010406375551603175 | 0.0007812499999999556 |
| 5  | 3.0014042721518988 |  -0.001400325422146409   |  0.000520589650150546 | 0.0001953124999999889 |
| 6  | 3.000623022151899  |  -0.0006222455968725171  | 0.0002603625961116 1004 | 4.8828124999997224e-05 |
| 7  | 3.000232397151899  | -0.00023228912247219569  | 0.0001301982474327344 | 1.2207031250002776e-05 |
| 8  | 3.0000370846518987 |  -3.708190131135325e-05  | 6.510336188825113e-05 | 3.051757812500694e-06 |
| 9  | 2.9999394284018988 |  6.057893571664863e-05   | 3.255274059047711e-05 | 7.629394531243061e-07 |
| 10 | 2.9999882565268985 |  1.174374892087826e-05   | 1.627610537924642e-05 | 1.9073486328107653e-07 |
| 11 | 3.0000126705893986 |  -1.267026831186513e-05  | 8.137986462334607e-06 | 4.768371582048597e-08 |
| 12 | 3.0000004635581483 |  -4.635577184330941e-07  | 4.0690097880146035e-06 | 1.1920928955121493e-08 |
| 13 | 2.9999943600425234 |  5.6400210937113116e-06  | 2.0345090331518715e-06 | 2.980232238726163e-09 |
| 14 | 2.9999974118003356 |  2.5882130643140044e-06  | 1.017253481696222e-06  | 7.450580596815408e-10 |
| 15 | 2.999998937679242  |  1.0623230153328223e-06  |  5.08626482221096e-07  | 1.8626451493393772e-10 |
+----+--------------------+--------------------------+-----------------------+-----------------------+
```

Tabel pentru x = 3.4142159763770765:

k	x_k	f(x_k)	rel err	(b-a)/2^(k-1)
1	3.417405063291139	0.0037721369087933	nil	0.04999999999999982
2	3.404905063291139	-0.010625422764640291	0.0036711743110739875	0.012499999999999956
3	3.411155063291139	-0.0035529501618967174	0.0018322239487905264	0.0031250000000000444
4	3.414280063291139	7.792501221359771e-05	0.000915273481399172	0.0007812500000000111
5	3.412717563291139	-0.0017454182210627778	0.00045784626797346794	0.00019531250000001665
6	3.413498813291139	-0.0008357244464889391	0.00022887074017961418	4.88281250000111e-05
7	3.413889438291139	-0.0003793943564573965	0.00011442227613429114	1.2207031250002776e-05
8	3.4140847507911394	-0.00015085835430284078	5.720786514013955e-05	3.051757812500694e-06
9	3.4141824070411393	-3.6497594386730725e-05	2.860311440844891e-05	7.629394531243061e-07
10	3.414231235166139	2.0705977725299363e-05	1.4301352672546075e-05	1.9073486328107653e-07
11	3.4142068211036394	-7.897741078011222e-06	7.1507274687457e-06	4.768371582005229e-08
12	3.414219028134889	6.403635140372899e-06	3.5753509511927213e-06	1.1920928954904653e-08
13	3.4142129246192643	-7.471737646369547e-07	1.7876786713856796e-06	2.980232238726163e-09
14	3.4142159763770765	2.8282004862489885e-06	8.938385366797607e-07	7.450580596815408e-10

Ex 7

Fie ecuaţia $f(x) = 0$ pe intervalul $[a, b]$.

a. Să se construiască în Python o procedură cu sintaxa **MetSecantei**$(f, x_0, x_1, \varepsilon)$ conform algoritmului metodei secantei. Procedura **MetSecantei** va returna soluţia aproximativă şi numărul de iteraţii.

b. Să se construiască în Python o procedură cu sintaxa **MetPozFalse**(f, a, b, ε) conform algoritmului metodei poziţiei false. Procedura **MetPozFalse** va returna soluţia aproximativă şi numărul de iteraţii.

c. Aflaţi toate rădăcinile funcţiei $f(x)$ pe intervalul $[a, b]$ folosind metodele secantei şi falsei poziţii cu eroarea de aproximare $\varepsilon = 10^{-5}$. Să se construiască atât graficul funcţiei, cât şi rădăcinile pe grafic. Formataţi figurile.

V1. $f(x) = 2x \cdot cos(2x) - (x+1)^2,\ [a, b] = [-5, 5]$

V2. $f(x) = 2x + 3cos(2x) - x,\ [a, b] = [-2, 3]$

V3. $f(x) = x + 1 - 2sin(\pi x),\ [a, b] = [-4, 2]$

V4. $f(x) = 2x \cdot cos(2x) - x + 2,\ [a, b] = [-4, 3]$

V5. $f(x) = sinx - e^{-x}\ , [a, b] = [0, 10]$

V6. $12cos(\pi x) = 2^x, [a, b] = [1, 3]$

S-a ales ecuația 5.

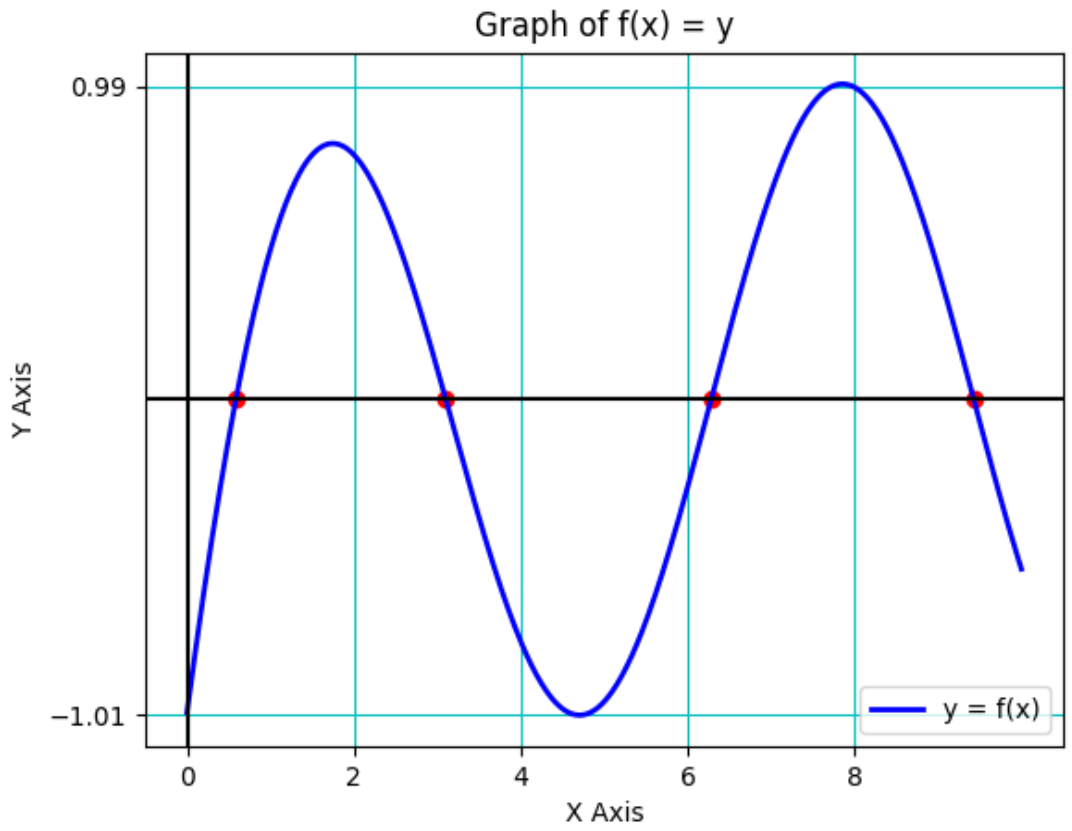
Codul Scris:

```
import math
import numpy as np
import matplotlib.pyplot as plt

fun = lambda x : math.sin(x) - (math.e ** (-x))
relative_error = lambda prevx, x : abs((x - prevx) / x)

def Secanta(f, x0, x1, a, b, eps = 10**-5):
    if f(a) * f(b) > 0:
        return None

    vx = [x0, x1]
    k = 1

    while relative_error(vx[k-1], vx[k]) >= eps:
        k += 1
        xk = (vx[k-2] * f(vx[k-1]) - vx[k-1] * f(vx[k-2])) / (f(vx[k-1])
- f(vx[k-2]))
        vx.append(xk)

        if xk < a or xk > b:
            return None
```

```
    return {'x': vx[k], 'nr_it': k - 1}

def PozitieFalsa(f, _a, _b, eps = 10**-5):

    if f(_a) * f(_b) > 0:
        return None

    k = 0
    a = [_a]
    b = [_b]
    x = [(_a*f(_b) - _b * f(_a)) / (f(_b) - f(_a))]

    k = 0

    while k == 0 or relative_error(x[k-1], x[k]) >= eps:
        k += 1
        if f(x[k-1]) == 0:
            x.append(x[k-1])
            break

        elif f(a[k-1])*f(x[k-1]) < 0:
            a.append(a[k-1])
            b.append(x[k-1])
            x.append((a[k]*f(b[k]) - b[k] * f(a[k])) / (f(b[k]) -
f(a[k])))
        else:
            a.append(x[k-1])
            b.append(b[k-1])
            x.append((a[k]*f(b[k]) - b[k] * f(a[k])) / (f(b[k]) -
f(a[k])))

    return {'x': x[k], 'nr_it': k}

if __name__ == '__main__':

    result_points1 = []
    result_points2 = []
    start_interval = 0
    end_interval = 10
    jump_interval = 0.5

    for i in np.linspace(start_interval, end_interval,
(int)((end_interval - start_interval) / jump_interval)):

        pt1 = Secanta(fun, i, i + jump_interval, i, i + jump_interval,
```

```
10**-5)
        if pt1:
            result_points1.append(pt1)

        pt2 = PozitieFalsa(fun, i, i + jump_interval, 10**-5)
        if pt2:
            result_points2.append(pt2)

    print("Metoda Secantei:")
    print(result_points1)

    print("Metoda Pozitiei False:")
    print(result_points2)


    fig = plt.figure(1)
    ax = plt.axes()
    x = np.linspace(start_interval, end_interval, (end_interval -
start_interval) * 50)
    y = np.vectorize(fun)(x)
    ax.plot(x, y, linestyle='-', lw = 2, color = 'b', label = 'y =
f(x)')
    ax.scatter([x['x'] for x in result_points1],
np.zeros(len(result_points1)), color="r")
    ax.grid(True, color = 'c')
    plt.xticks(np.arange(start_interval,end_interval,(int)((end_interval
- start_interval) / 4)))
    plt.yticks(np.arange(min(y),max(y),(int)((end_interval -
start_interval) / 4)))
    ax.legend(loc='best')
    plt.xlabel('X Axis')
    plt.ylabel('Y Axis')
    plt.title('Graph of f(x) = y')
    ax.axhline(y=0,color='k')
    ax.axvline(x=0,color='k')
    plt.show()
```

Graficul Rezultant



Rezultatele Numerice Afișate

```
Metoda Secantei:
[{'x': 0.5885327439817414, 'nr_it': 5}, {'x': 3.0963639327379577,
'nr_it': 3}, {'x': 6.285049273382586, 'nr_it': 3}, {'x':
9.424697254738508, 'nr_it': 3}]
Metoda Pozitiei False:
[{'x': 0.5885327485539672, 'nr_it': 4}, {'x': 3.0963639278622517,
'nr_it': 2}, {'x': 6.285049273382586, 'nr_it': 2}, {'x':
9.424697254738508, 'nr_it': 2}]
```

Ex 9

Metoda Gauss - Jordan: Această metodă, la iterația k, transformă în zerouri elementele de pe coloana k a matricei extinse $[A|b]$, situate atât pe liniile sub pivot, cât și pe liniile deasupra pivotului. Se obține în felul acesta o matrice diagonala, iar sistemul echivalent asociat poate fi rezolvat imediat. Să se construiască un nou algoritm în baza algortmului Gauss fără pivotare care să rezolve sistemele de ecuații liniare în baza metodei Gauss-Jordan.

$$\left[\begin{array}{cccc:c} a_{11}^{(1)} & 0 & \cdots & 0 & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & \ddots & \vdots & a_{2,n+1}^{(2)} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & a_{nn}^{(n)}) & a_{n,n+1}^{(n)} \end{array}\right]$$

$$x_i = \frac{a_{i,n+1}^{(i)}}{a_{ii}^{(i)}},$$

$i = 1, 2, ..., n.$

Obs.: Elementul $a_{ij}^{(k)}$ reprezintă elementul matricei extinse la iterația k.

Să se implementeze în Python metoda Gauss-Jordan. Să se testeze algoritmul pe sistemul rezolvat de la ex. 8.

Codul Scris:

```
import numpy as np

def Verificare(A, b, tol = 10 ** -16):
    # Verificam ca A sa fie matrice bidimensionala
    if A.ndim != 2:
        print("Vectorul nu este bidimensional")
        return False

    # Verificam ca A sa fie matrice patratica
    if A.shape[0] != A.shape[1]:
        print("Matricea nu este patratica")
        return False

    n = A.shape[0] # dimensiunea sistemului

    # Verificam ca b sa fie unidimensional
    if b.ndim != 1:
        print("Vectorul b nu este unidimensional")
        return False
    # Verificam ca b sa aiba dimensiunea corecta
```

```
    if len(b) != n:
        print("Vectorul b nu are dimensiunea corecta")
        return False

    return True

def readLinearSystem(path):
    dim = 0
    mat = ""
    with open(path, 'r') as file:
        dim = int(file.readline())
        mat = np.zeros((dim, dim+1))

        for i in range(dim):
            mat[i] = [float(x) for x in file.readline().split()]

    return mat[:, :dim], mat[:, dim:].T[0]

def GaussJordan(A, b, tol = 10 ** -16):
    # Verificam ca datele introduse sunt corecte
    if Verificare(A, b, tol) == False:
        return None

    # Construim matricea extinsa
    Aext = np.c_[A, b]

    # Dimensiunea sistemului
    n = A.shape[0]

    for k in range(n):

        # Aflam prima linie pe care Apk este nenul
        p = k
        while p < n and abs(Aext[p][k]) < tol:
            p += 1

        # Daca nu exista Apk nenul, sistemul nu e determinat
        if abs(Aext[p][k]) < tol:
            print("Sistemul nu este determinat")
            return None

        # Daca apk nu e de forma akk, interschimbam liniile
        if k != p:
            Aext[[k, p]] = Aext[[p, k]]

        # Reducem liniile pentru a forma matricea triunghiulara
```

```
        for l in range(n):
            if l == k:
                continue
            m = Aext[l][k] / Aext[k][k]
            Aext[l] = Aext[l] - Aext[k] * m

    # Daca matricea nu e corecta, sistemul nu e determinat
    if abs(Aext[n-1][n-1]) < tol:
        print("Sistemul nu este determinat")
        return None

    return Aext

def SolveDiag(diag):
    dim = diag.shape[0]
    res = np.zeros(dim)
    for i in range(dim):
        res[i] = diag[i][dim] / diag[i][i]
    return res

if __name__ == "__main__":
    A, b = readLinearSystem("in.txt")
    Diag = GaussJordan(A, b)
    print(Diag)
    res = SolveDiag(Diag)
    print(res)
```

In.txt

```
3
-3 2 -1 -1
6 -6 7 -7
3 -4 4 -6
```

Rezultate:

```
rotak@rtkUBUNTU:~/aaa_work/Calcul Numeric/Tema1/p9$ python main.py
[[-3.  0.  0. -6.]
 [ 0. -2.  0. -4.]
 [ 0.  0. -2.  2.]]
[ 2.  2. -1.]
```