Sandu Alexandru-Cristian
Grupa 341
Facultatea de Matematica si Informatica

CALCUL NUMERIC – TEMA 1

```python
import metode_numerice_ecuatii_algebrice as mnea
import numpy as np
import matplotlib.pyplot as plt
# VARIANTA V11


def f(x):
    y = x**3 - 9*x**2 + 24*x - 19
    return y


def ex1():
    print("----------------------EX1----------------------")
    a = 0
    b = 8
    n_noduri = 100
    interval = mnea.cauta_intervale(f, a, b, 10)
    print(f'Intervalele pe care voi efectua cele doua metode sunt:\n{interval}')
    x0 = interval[0]
    x1 = interval[1]
    eps = 10 ** -5
    x_grafic2 = np.linspace(0, 8, n_noduri)
    y_grafic2 = f(x_grafic2)

    # metoda secantei
    print("Metoda secantei:")
    plt.plot(x_grafic2, y_grafic2, linewidth=3)
    plt.grid()
    plt.axvline(0, color='black')
    plt.axhline(0, color='black')
    sol_sec = np.zeros(x0.shape)  # vector in care voi salva solutiile
    nr_sec = np.zeros(x0.shape)  # vectorul in care voi salva numarul de iteratii
    for i in range(len(x0)):
        sol_sec[i], nr_sec[i] = mnea.metoda_secantei(f, a, b, x0[i], x1[i], eps)
        print(f'Solutia pe intervalul [{x0[i]:.2f}, {x1[i]:.2f}] este x{i} ='
                    f' {sol_sec[i]}, calculat dupa {int(nr_sec[i])} iteratii.')
    plt.plot(sol_sec, f(sol_sec), 'o', markerfacecolor="red", markersize=5)
    plt.show()

    # metoda pozitiei false
    print("Metoda pozitiei false:")
    plt.plot(x_grafic2, y_grafic2, linewidth=3)
    plt.grid()
    plt.axvline(0, color='black')
    plt.axhline(0, color='black')
    a = interval[0]
    b = interval[1]
    sol_fpoz = np.zeros(a.shape)  # vector in care voi salva solutiile
    nr_fpoz = np.zeros(a.shape)  # vectorul in care voi salva numarul de iteratii
    for i in range(len(a)):
        sol_fpoz[i], nr_fpoz[i] = mnea.metoda_pozitiei_false(f, a[i], b[i], eps)
```

```
        print(f'Solutia pe intervalul [{a[i]:.2f}, {b[i]:.2f}] este x{i} =
{sol_sec[i]}, calculat dupa {int(nr_sec[i])} iteratii.')
    plt.plot(sol_fpoz, f(sol_fpoz), 'o', markerfacecolor="red", markersize=5)
    plt.show()


def ex2():
    print("---------------------EX2---------------------")
    d = 21
    f2 = -7
    c = -2
    n = 20
    A = np.zeros((n, n))
    b = np.zeros((n, 1))  # vectorul termenilor liberi
    A[0][0] = d
    A[0][1] = f2
    A[n-1][n-2] = c
    A[n-1][n-1] = d
    for i in range(1, n-1):
        A[i][i - 1] = c
        A[i][i] = d
        A[i][i + 1] = f2
    b[0] = b[n-1] = 2
    for i in range(1, n-1):
        b[i] = 1

    tol = 10**-16
    rez = mnea.metoda_gauss_cu_pivotare_totala(A, b, tol)
    print("Solutia este:")
    for i in range(len(rez)):
        print(f'x{i+1}={rez[i][0]}')
    print(f'Verificare: \n{A@rez}')  # verific daca prin A * rezultata obtin b


def ex3():
    print("---------------------EX3---------------------")
    tol = 10**-16
    A = np.array([[10.0, 30.0, 16.0], [2.0, 15.0, 7.0], [2.0, 5.0, 3.0]])
    b = np.array([[118.0], [53.0], [21.0]])
    rez = mnea.metoda_gauss_cu_pivotare_partiala(A, b, tol)
    print("Solutia este:")
    for i in range(len(rez)):
        print(f'x{i+1}={rez[i][0]}')
    print(f'Verificare: \n{A@rez}')

# Aici sunt apelate functiile pentru fiecare exercitiu
ex1()
ex2()
ex3()
```

Functiile folosite din fisierul metode_numerice_ecuatii_algebrice.py

```python
import numpy as np
"""
Funcție care caută intervalele pe care funcția are o soluție.
f(a) * f(b) < 0 -> EXISTENȚA
"""


def cauta_intervale(f, a, b, n):
    """
    :param f: funcția asociată ecuației f(x)=0.
    :param a: capătul din stânga interval.
    :param b: capătul din dreapta interval.
    :param n: nr de subintervale în care împărțim
                intervalul global (a, b).
    :return: Matricea 'intervale' cu 2 linii; prima linie -> capăt st
                interval curent si
                a doua linie -> capat dr si
                un nr de coloane = nr radacini
    """
    # returnează n+1 numere, situate la distanțe egale, din cadrul intervalului [a,
b]
    x = np.linspace(a, b, n + 1)
    for i in range(len(x)):
        if f(x[i]) == 0:  # capetele intervalelor mele nu au voie să fie 0;
                          # tb să avem soluțiile în intervale, nu la capete
            print("Schimbati numarul de intervale")
            exit(0)

    matrice = np.zeros((2, 1000))
    z = 0
    for i in range(n):
        if f(x[i]) * f(x[i + 1]) < 0:  # existență soluție
            matrice[0][z] = x[i]
            matrice[1][z] = x[i + 1]
            z += 1
    matrice_finala = matrice[:, 0:z]  # iau ambele 2 linii și
                                      # doar coloanele de la
                                      # 0 la z (numărat mai sus)
    return matrice_finala


"""
    Metoda secantei (Tema)
"""


def metoda_secantei(f, a, b, x0, x1, eps):
    """
    :param f: functia pentru care cautam f(x) = 0
    :param a: capatul din stanga al intervalului
    :param b: capatul din dreapta al intervalului
```

```python
    :param x0: ales din intervalul [a, b]
    :param x1: ales din intervalul [a, b]
    :param eps: toleranta, eroarea (epsilon)
    :return: x_aprox = aproximarea x_k a solutiei x 'stelat'
                    a ecuatiei f(x) = 0, x apartine [a, b]
             k = numarul de iteratii
    """
    x_k_1 = x0
    x_k = x1
    k = 0  # numarul de iteratii
    while abs(x_k - x_k_1)/abs(x_k_1) >= eps:
        k += 1
        x_k_2 = x_k_1
        x_k_1 = x_k
        x_k = (x_k_2 * f(x_k_1) - x_k_1 * f(x_k_2)) / (f(x_k_1) - f(x_k_2))
        if x_k < a or x_k > b:
            print("Introduceti alte valori pentru x0 si x1")
            exit(0)
    x_aprox = x_k
    return x_aprox, k


"""
    Metoda Pozitiei False (Tema)
"""


def metoda_pozitiei_false(f, a, b, eps):
    """
    :param f: functia pentru care cautam f(x) = 0
    :param a: capatul din stanga al intervalului
    :param b: capatul din dreapta al intervalului
    :param eps: toleranta, eroarea (epsilon)
    :return: aproximarea xk a solutiei exacte x* a ecuatiei
                       f(x) = 0 si numarul de iteratii
    """
    k = 0  # numarul de iteratii
    a0 = a
    b0 = b
    x0 = (a0 * f(b0) - b0 * f(a0)) / (f(b0) - f(a0))
    x_old = x0
    x_new = x_old
    while True:
        k += 1
        if f(x_old) == 0:
            x_new = x_old
            break
        elif f(a0) * f(x_old) < 0:
            b0 = x_old
            x_new = (a0 * f(b0) - b0 * f(a0)) / (f(b0) - f(a0))
        elif f(a0) * f(x_old) > 0:
            a0 = x_old
            x_new = (a0 * f(b0) - b0 * f(a0)) / (f(b0) - f(a0))
        if abs(x_new - x_old) / abs(x_old) < eps:
            break
        x_old = x_new
    return x_new, k
```

```python
"""
    Metoda Substitutiei Descendente
"""


def met_subst_desc(a, b, tol):
    """
    :param a: matricea patratica, superior triunghiulara,
        cu toate elementele de pe diagonala principala nenule
    :param b: vectorul termenilor liberi
    :param tol: valoarea numerica foarte mica in raport cu care
        vom compara numerele apropiate de 0
    :return: solutia sistemului reprezentata printr-un vector cu
        mai multe componente x1, x2, ...
    """
    # Verificam daca matricea este patratica
    m, n = np.shape(a)
    if m != n:
        print("Matricea nu este patratica! Introduceti o alta matrice!")
        return None

    # Verificam daca matricea este superior triunghiulara
    for i in range(m):
        for j in range(i):  # i > j (merge pana la i - 1)
            if abs(a[i][j]) > tol:
                print("Matricea nu este superior triunghiulara!"
                      " Introduceti alta matrice!")
                return None

    # Verificam daca elementele de pe diag principala sunt
    #   nenule(sist comp det, am solutie unica)
    for i in range(n):
        if a[i][i] == 0:
            print("Matricea contine 0 pe diagonala principala"
                  " (sist comp det, am solutie unica)")
            return None

    # Aplic algoritmul
    x = np.zeros((n, 1))
    x[n - 1] = 1 / a[n-1][n-1] * b[n-1]
    k = n - 2
    while k >= 0:
        suma = 0
        for i in range(k+1, n):
            suma += a[k][i] * x[i]
        x[k] = 1/a[k][k]*(b[k] - suma)
        k -= 1

    return x


"""
    Metoda Gauss fara pivotare
"""
```

```python
def metoda_gauss_fara_pivotare(A, b, tol):
    """
    :param A: mat asociata sistemului, mat patratica
    :param b: vectorul termenilor liberi
    :param tol: valoare cu care comparam numerele nenule
    :return: x = solutia sistemului
    """
    # Verificam daca matricea este patratica
    m, n = np.shape(A)
    if m != n:
        print("Matricea nu este patratica! Introduceti o alta matrice!")
        return None

    # Definim matricea A extins
    A_extins = np.concatenate((A, b), axis=1)  # axis = 0 l-ar pune
                                               # pe b ca o linie noua,
                                               # 1 il pune ca coloana
    for k in range(n - 1):
        p = None
        for j in range(k, n):
            if abs(A[j][k]) > tol:  # atentie sa iau valoarea absoluta (abs)
                p = j
                break

        if p is None:
            print("Sistemul nu admite solutie unica")
            return None

        if p != k:
            A_extins[[p, k]] = A_extins[[k, p]]  # swap linia p cu linia k

        for j in range(k + 1, n):
            A_extins[j] = A_extins[j] - (A_extins[j][k] / A_extins[k][k]) *
A_extins[k]

    if abs(A_extins[n - 1][n - 1]) <= tol:
        print("Sistemul nu admite solutie unica")
        return None

    x = met_subst_desc(A_extins[:, 0:n], A_extins[:, n], tol)

    return x


"""
    Metoda Gauss cu pivotare partiala
"""


def metoda_gauss_cu_pivotare_partiala(A, b, tol):
    """
    :param A: mat asociata sistemului, mat patratica
    :param b: vectorul termenilor liberi
    :param tol: valoare cu care comparam numerele nenule
    :return: x = solutia sistemului
    """
```

```python
    # Verificam daca matricea este patratica
    m, n = np.shape(A)
    if m != n:
        print("Matricea nu este patratica! Introduceti o alta matrice!")
        return None

    # Definim matricea A extins
    A_extins = np.concatenate((A, b), axis=1)  # axis = 0 l-ar pune
                                               # pe b ca o linie noua,
                                               # 1 il pune ca coloana

    print(f'Iteratia {0} \nA_extins = \n{A_extins}\n')

    for k in range(n - 1):
        p = k
        maxim = A_extins[k][k]
        for j in range(k + 1, n):
            if abs(A_extins[j][k]) > abs(maxim):
                p = j
                maxim = A_extins[j][k]

        if abs(maxim) <= tol:
            print("Sistem nu admite solutie unica!")
            return None

        if p != k:
            A_extins[[p, k]] = A_extins[[k, p]]  # swap linia p cu linia k

        for j in range(k + 1, n):
            A_extins[j] = A_extins[j] - (A_extins[j][k] / A_extins[k][k]) *
A_extins[k]

        print(f'Iteratia {k+1} \nA_extins = \n{A_extins}\n')

    if abs(A_extins[n - 1][n - 1]) <= tol:
        print("Sistemul nu admite solutie unica, incompatibil/comp. nedet.")
        return None

    x = met_subst_desc(A_extins[:, 0:n], A_extins[:, n], tol)

    return x


"""
    Metoda Gauss cu pivotare totala
"""


def metoda_gauss_cu_pivotare_totala(A, b, tol):
    """
    :param A: mat asociata sistemului, mat patratica
    :param b: vectorul termenilor liberi
    :param tol: valoare cu care comparam numerele nenule
    :return: x = solutia sistemului
    """
    # Verificam daca matricea este patratica
    m, n = np.shape(A)
```

```
    if m != n:
        print("Matricea nu este patratica! Introduceti o alta matrice!")
        return None

    # Definim matricea A extins
    A_extins = np.concatenate((A, b), axis=1)  # axis = 0 l-ar pune
                                               # pe b ca o linie noua,
                                               # 1 il pune ca coloana
    # Definim vectorul index care contine ordinea solutiilor
    index = np.arange(n)
    print(f'index initial= {index}')

    for k in range(n - 1):
        p = k
        m_alg = k  # m folosit in prezentarea algoritmului din curs
        maxim = A_extins[k][k]
        for j in range(k + 1, n):
            for s in range(m_alg+1, n):
                if abs(A_extins[j][s]) > abs(maxim):
                    p = j
                    m_alg = s
                    maxim = A_extins[j][s]

        if abs(maxim) <= tol:
            print("Sistem nu admite solutie unica!")
            return None

        if p != k:
            # schimb linia p cu linia k
            A_extins[[p, k]] = A_extins[[k, p]]

        if m_alg != k:
            # schimb coloana m_alg cu coloana k
            A_extins[:, [m_alg, k]] = A_extins[:, [k, m_alg]]
            index[[m_alg, k]] = index[[k, m_alg]]

        for j in range(k + 1, n):
            A_extins[j] = A_extins[j] - (A_extins[j][k] / A_extins[k][k]) *
A_extins[k]

        # print(f'Iteratia {k+1} \nA_extins = \n{A_extins}\n')

    if abs(A_extins[n - 1][n - 1]) <= tol:
        print("Sistemul nu admite solutie unica, incompatibil/comp. nedet.")
        return None

    x = met_subst_desc(A_extins[:, 0:n], A_extins[:, n], tol)

    # urmeaza sa modificam ordinea solutiilor in functie de vectorul index
    print(f'index final= {index}')
    x_final = np.copy(x)
    for i in range(n):
        x_final[index[i]] = x[i]

    return x_final
```