3. SLAJ, iterativne metode

Dat je sistem jednačina:

$$9x_1+3x_2+1x_3=33$$
$$7x_1+8x_2+9x_3=54$$
$$4x_1+x_2+9x_3=13$$

matrični oblik:

$$\begin{bmatrix}9 & 3 & 1\\ 7 & 8 & 9\\ 4 & 1 & 9\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}33\\ 54\\ 13\end{bmatrix}$$

ili:

$$Ax=b$$

, gde je A matrica množilaca rešenja sistema x, a b vektor slobodnih članova.

1 Definisati matricu A i vektor slobodnih članova b :

```
A = np.array([
     [9,     3,     1],
     [7,     8,     9],
     [4,     1,     9]])
```

```
b = np.array([33,    54,    13])
```

2 Do vektora x se može doći upotrebom metode `np.linalg.solve()`:

```
x = np.linalg.solve(A, b)
```

Rezultat:

```
x =  [2.00   5.00   0.00]
```

3 Proveriti tačnost jednakosti:

```
np.dot(A, x)
```

Rezultat:

```
[33. 54. 13.]
```

1. *Jacobi* metoda

Zadatak 1. Napisati *Jacobi* iterativnu metodu da za rešavanje sistema linearnih algebarskih jednačina. Pokušati prvo ručno jednu *Jacobi* iteraciju vrstu po vrstu:

```
A = np.array([                  b = np.array(                 x0 = np.array(
    [9,     3,     1],          [33,    54,    13])           [0,     0,     0], ‘d’)
    [7,     8,     9],
    [4,     1,     9]])
```

Postaviti tekuće $x^{(1)}$ na početno $x^{(0)}$:

```
x = x0.copy()
```

Rezultat:

```
 x = [0     0     0]    A = [[9    3    1]     x0 = [0     0     0]   b = [33    54    13]
                             [7    8    9]
                             [4    1    9]]
```

Transformisati 1. vrstu iz oblika $Ax=b$ u $x=Tx+c$:

$$x_1 a_{11}+x_2 a_{12}+x_3 a_{13}=b_1$$

$$x_1 a_{11}=b_1-x_2 a_{12}-x_3 a_{13}$$

$$x_1=\frac{1}{a_{11}}(b_1-x_2 a_{12}-x_3 a_{13})$$

$$x_1=\frac{1}{a_{11}}\left(b_1-\begin{bmatrix}a_{12} & a_{13}\end{bmatrix}\begin{bmatrix}x_2\\ x_3\end{bmatrix}\right)$$

$$x_1^{(k)}=\frac{1}{a_{11}}\left(b_1-\begin{bmatrix}a_{12} & a_{13}\end{bmatrix}\begin{bmatrix}x_2^{(k-1)}\\ x_3^{(k-1)}\end{bmatrix}\right)$$

```
x[0] = 1 / A[0, 0] * (b[0] – np.dot(A[0, 1:], x0[1:])
```

Rezultat:

```
 x = [3.6667  0    0]   A = [[9    3    1]     x0 = [0     0     0]   b = [33    54    13]
                             [7    8    9]
                             [4    1    9]]
```

Transformisati 2. vrstu iz oblika $Ax=b$ u $x=Tx+c$:

$$x_1 a_{21}+x_2 a_{22}+x_3 a_{23}=b_2$$

$$x_2 a_{22}=b_2-x_1 a_{21}-x_3 a_{23}$$

$$x_2=\frac{1}{a_{22}}(b_2-x_1 a_{21}-x_3 a_{23})$$

$$x_2^{(k)}=\frac{1}{a_{22}}(b_2-x_1^{(k-1)} a_{21}-x_3^{(k-1)} a_{23})$$

```
x[1] = 1 / A[1, 1] * (b[1] – np.dot(A[1, :1], x0[:1]) – np.dot(A[1, 2:], x0[2:]))
```

Rezultat:

```
 x = [3.6667  6.75  0]   A = [[9    3    1]    x0 = [0    0    0]      b = [33   54   13]
                             [7    8    9]
                             [4    1    9]]
```

$$x_1 a_{31} + x_2 a_{32} + x_3 a_{33} = b_3$$
$$x_3 a_{33} = b_3 - x_1 a_{31} - x_2 a_{32}$$
$$x_3 = \frac{1}{a_{33}} (b_3 - x_1 a_{31} - x_2 a_{32})$$
$$x_3 = \frac{1}{a_{33}} \left(b_3 - [a_{31} \quad a_{32}] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$
$$x_3^{(k)} = \frac{1}{a_{33}} \left(b_3 - [a_{31} \quad a_{32}] \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \end{bmatrix}\right)$$

```
x[2] = 1 / A[2, 2] * (b[2] – np.dot(A[2, :2], x0[:2]))
```

Rezultat:

```
 x = [3.6667 6.75 1.44]  A = [[9    3    1]    x0 = [0     0     0]    b = [33   54   13]
                             [7    8    9]
                             [4    1    9]]
```

Pripremiti sledeću iteraciju. Postaviti sledeće x^2 na tekuće x^1 :

```
x0 = x.copy()
```

Rezultat:

```
 x = [3.6667 6.75 1.44]  A = [[9    3    1]    x = [3.6667 6.75 1.44]  b = [33   54   13]
                             [7    8    9]
                             [4    1    9]]
```

Uporediti korake:

```
x[0] = 1 / A[0, 0] * (b[0]                                  –  np.dot(A[0, 1:], x0[1:]))
x[1] = 1 / A[1, 1] * (b[1] – np.dot(A[1, :0], x0[:0])       –  np.dot(A[1, 2:], x0[2:]))
x[2] = 1 / A[2, 2] * (b[2] – np.dot(A[2, :1], x0[:1])                                 )
x0 = x.copy()
```

Dopuniti nedostajuće elemente:

```
x[0] = 1/A[0, 0]*(b[0] – np.dot(A[0, :0], x0[:0])   - np.dot(A[0, 1:], x0[1:]))
x[1] = 1/A[1, 1]*(b[1] – np.dot(A[1, :1], x0[:1])   - np.dot(A[1, 2:], x0[2:]))
x[2] = 1/A[2, 2]*(b[2] – np.dot(A[2, :2], x0[:2])   - np.dot(A[2, 3:], x0[3:]))
x0 = x.copy()
```

Proširiti indekse:

```
x[0] = 1/A[0,0]*(b[0]–np.dot(A[0,:0]*x0[:0])–np.dot(A[0,(0 + 1):]*x0[(0 + 1):]))
x[1] = 1/A[1,1]*(b[1]–np.dot(A[1,:1]*x0[:1])–np.dot(A[1,(1 + 1):]*x0[(1 + 1):]))
x[2] = 1/A[2,2]*(b[2]–np.dot(A[2,:2]*x0[:2])–np.dot(A[2,(2 + 1):]*x0[(2 + 1):]))
x0 = x.copy()
```

1 Pogledati šta je fiksno, a šta promenljivo. Primetiti da promenljivi indeksi rastu od 0, do dimenzije matrice - 1. Ovo se može zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje.

```
for r in range(rows):
    x[r] = 1/A[r,r]*(b[r]-np.dot(A[r, :r],x0[:r])-np.dot(A[r,r+1:],x0[r+1:]))
x0 = x.copy()
```

Ako se prethodni postupak ponavlja:

nakon 1. ponavljanja:
```
X = [3.6667    6.7500    1.4444]                x0 = [3.6667    6.7500    1.4444]
```

nakon 2. ponavljanja:
```
X = [1.2562    1.9167   -0.9352]                x0 = [1.2562    1.9167   -0.9352]
```

.

.

.

nakon 53. ponavljanja:
```
x = [2.0001    5.0001    0.0001]                x0 = [2.0001    5.0001    0.0001]
```

nakon 54. ponavljanja:
```
x = [1.9999    4.9999   -0.0000]                x0 = [1.9999    4.9999   -0.0000]
```

nakon 55. ponavljanja:
```
x = [2.0000    5.0001    0.0000]                x0 = [2.0000    5.0001    0.0000]
```

nakon 56. ponavljanja:
```
x = [2.0000    4.9999   -0.0000]                x0 = [2.0000    4.9999   -0.0000]
```

nakon 57. ponavljanja:
```
x = [2.0000    5.0001    0.0000]                x0 = [2.0000    5.0001    0.0000]
```

nakon 58. ponavljanja:
```
x = [2.0000    4.9999   -0.0000]                x0 = [2.0000    4.9999   -0.0000]
```

nakon 59. ponavljanja:
```
x = [2.0000    5.0000    0.0000]                x0 = [2.0000    5.0000    0.0000]
```

nakon 60. ponavljanja:
```
x = [2.0000    5.0000   -0.0000]
```
… tekuće i prethodno rešenje će postati blisko ili jednako.

2 Prethodna *for* petlja se može ugnjezditi u beskonačnu *while* petlju:

```
x = x0.copy()
while True:
  for r in range(rows):
      x[r] = 1/A[r,r]*(b[r]-np.dot(A[r, :r],x0[:r])-np.dot(A[r,r+1:],x0[r+1:]))

  x0 = x.copy()
```

3 Potrebno je definisati uslov za prekid iteracije u zavisnosti od tražene preciznosti:

```
if np.linalg.norm(x0 – x, np.Inf) < err_max:
  break
```

4 Za slučaj da metod **divergira** ili da je iz drugih razloga potrebno ograničiti broj iteracija, potrebno je definisati njihov maksimalni broj. Umesto *while* petljom, to se može iskazati *for* petljom:

```
x = x0.copy()
for it in range(it_max)
    for r in range(rows)
   .
   .
   .
```

5 Sada je moguće definisati funkciju koja sadrži prethodni algoritam:

```
def jacobi(A, b, x0, err_max, it_max)
    rows = A.shape[0]
    x = x0.copy()
    .
    .
    .
    return x, it + 1
```

6 Testirati funkciju `jacobi` na primeru. Izračunati tačnu vrednost upotrebom operatora `np.linalg.solve()` i izračunati apsolutnu grešku:

```
A = np.array([                  b = np.array(                 x0 = np.array(
    [9,     3,     1],              [33,     54,     13])          [0,     0,     0], ‘d’)
    [7,     8,     9],
    [4,     1,     9]])

x, it = jacobi(A, b, x0, 10e-05, 100)
xt = np.linalg.solve(A, b)
abs_err = abs(xt - x)
```

Rezultat:

```
x = [2.0000    5.0000    0.0000]

it = 71
```

```
xt = [2.0000    5.0000    0.0000]

abs_err = 1.0e-05 * [0.1938    0.4259    0.1617]
```

2. *Gauss-Seidel* metoda

Zadatak 2. Konvergencija *Jacobi* metode se može **ubrzati**. Napisati *Gauss-Seidel* iterativnu metodu za rešavanje sistema linearnih algebarskih jednačina.

1 *Gauss-Seidel* za izračunavanje rešenja x_i^k koristi već izračunate vrednosti ostalih rešenja $x_{j<i}^k$ iz tekuće iteracije:

```
def gs(A, b, x0, err_max, it_max)
    .
    .
    .
         x[r] = 1/A[r,r]*(b[r]-np.dot(A[r, :r],x[:r])-np.dot(A[r,r+1:],x0[r+1:]))
    .
    .
    .
    return x, it + 1
```

2 Testirati funkciju *gs* na primeru, pa uporediti rezultat sa metodom *jacobi.* Izračunati tačnu vrednost upotrebom operatora `np.linalg.solve()` i izračunati apsolutnu grešku:

```
A = np.array([                  b = np.array(                  x0 = np.array(
    [9,     3,     1],              [33,    54,    13])             [0,    0,    0], 'd')
    [7,     8,     9],
    [4,     1,    9]])

x_jacobi, it_jacobi = jacobi(A, b, x0, 1e-5, 100)
x_gs, it_gs = gs(A, b, x0, 1e-5, 100)
xt = np.linalg.solve(A, b)
err_jacobi = abs(xt - x_jacobi)
err_gs = abs(xt – x_gs)
```

Rezultat:

```
x_jacobi = [2.0000    5.0000    0.0000]

it_jacobi = 71

x_gs = [2.0000    5.0000   -0.0000]

it_gs = 16

xt = [2.0000    5.0000    0.0000]

err_jacobi = 1.0e-05 * [0.1938    0.4259    0.1617]

err_gs = 1.0e-05 * [0.0651    0.2953    0.0039]
```

Zadatak 3. Isprobati *Gauss-Seidel* metodu na sledećem primeru:

```
A = np.array([                    b = np.array(                     x0 = np.array(
    [2,      5,      6],              [69,    100,     67])              [0,     0,     0], ‘d’)
    [5,      4,      9],
    [6,      2,      3]])
x, it = gs(A, b, x0, 1e-5, 100)
```

Rezultat:

```
x = 1.0e+81 * [0.8944   -0.6344   -1.3659]

it = 100
```

Metoda nije uspela da pronađe rešenje u ograničenom broju iteracija.
Postoje sistemi kod kojih **nije zagarantovana konvergencija** iterativnih metoda. Odabir drugačijeg početnog rešenja može da reši problem, ali ne uvek.

* **Zadatak 4**. Napisati *Successive Over-Relaxation* metodu za rešavanje sistema linearnih algebarskih jednačina.