2. Gausova eliminacija

Dat je sistem jednačina:

$$5x_1+8x_2+5x_3=48$$
$$4x_1+2x_2+7x_3=43$$
$$8x_1+5x_2+8x_3=69$$

matrični oblik:

$$\begin{bmatrix} 5 & 8 & 5 \\ 4 & 2 & 7 \\ 8 & 5 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 48 \\ 43 \\ 69 \end{bmatrix}$$

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([
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
```

```
b = np.array([48, 43, 69])
```

2 Do vektora x se može doći upotrebom operatora `linalg.solve` iz biblioteke *numpy*:

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

Rezultat:

```
x = [5 1 3]
```

3 Proveriti tačnost jednakosti:

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

Rezultat:

```
[48 43 69]
```

Zadatak 1. Definisati funkciju *upper_triangular* koja proizvoljni sistem jednačina svodi na gornji trougaoni oblik.
Pokušati prvo ručno svođenje sistema na gornji trougaoni oblik.

```
A =
     5      8      5
     4      2      7
     8      5      8
```

Potrebno je eliminisati sve elemente ispod glavne dijagonale.

1 Odabrati 1. element 2. vrste, podeliti ga 1. elementom 1. vrste, a zatim dobijenim množiocem pomnožiti 1. vrstu i oduzeti je od 2. vrste:

```
A[1, :] = A[1, :] - A[0, :] * A[1, 0] / A[0, 0]
```

Rezultat:

```
A =
    5.0000    8.0000    5.0000
         0   -4.4000    3.0000
    8.0000    5.0000    8.0000
```

Svi elementi ispod glavne dijagonale u 2. vrsti su sada eliminisani.

2 Odabrati 1. element 3. vrste, podeliti ga 1. elementom 1. vrste, a zatim dobijenim množiocem pomnožiti 1. vrstu i oduzeti je od 3. vrste:

```
A[2, :] = A[2, :] - A[0, :] * A[2, 0] / A[0, 0]
```

Rezultat:

```
A =
    5.0000    8.0000    5.0000
         0   -4.4000    3.0000
         0   -7.8000         0
```

1. kolona u 3. vrsti je sada eliminisana. Preostaje još 2. kolona.

3 Odabrati 2. element 3. vrste, podeliti ga 2. elementom 2. vrste, a zatim dobijenim množiocem pomnožiti 2. vrstu i oduzeti je od 3. vrste:

```
A[2, :] = A[2, :] - A[1, :] * A[2, 1] / A[1, 1]
```

Rezultat:

```
A =
    5.0000    8.0000    5.0000
         0   -4.4000    3.0000
         0         0   -5.3182
```

Svi elementi ispod glavne dijagonale u 3. vrsti su sada eliminisani. Matrica je svedena na gornji trougaoni oblik.

Potrebno je definisati algoritam koji će ovo uraditi za proizvoljnu kvadratnu matricu.

1 Definisati praznu funkciju *upper_triangular*:

```
def upper_triangular(A):

   return A
```

Uporediti korake:

- za 1. vrstu:
  ```
  A[1, :] = A[1, :] - A[0, :] * A[1, 0] / A[0, 0]
  A[2, :] = A[2, :] - A[0, :] * A[2, 0] / A[0, 0]
  ```
- za 2. vrstu:
  ```
  A[2, :] = A[2, :] - A[1, :] * A[2, 1] / A[1, 1]
  ```
- za 3. vrstu nema elemenata ispod glavne dijagonale

Odabrati 1. vrstu i pogledati šta je fiksno, a šta promenljivo. Primetiti da promenljivi indeksi rastu od indeksa vrste + 1 do dimenzije matrice. Ovo se može zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje:

```
for it2 in range(0 + 1, rows)
    A[it2, :] = A[it2, :] - A[0, :] * A[it2, 0] / A[0, 0]
end
```

Posmatrati ovu *for* petlju za sve vrste:

```
for it2 in range(0 + 1, rows)
    A[it2, :] = A[it2, :] - A[0, :] * A[it2, 0] / A[0, 0]
end

for it2 in range(1 + 1, rows)
    A[it2, :] = A[it2, :] - A[1, :] * A[it2, 1] / A[1, 1]
end
```

Pogledati šta je fiksno, a šta promenljivo. Primetiti da promenljivi indeksi rastu od 0, do dimenzije matrice - 2.

2 Prethodna *for* petlja se može ugnjezditi u novu *for* petlju pri čemu promenljivi indeksi zavise od indeksa nove for petlje. U funkciji definisati par *for* petlji:

```
for it1 in range(0, rows-1):
    for it2 in range(it1 + 1, rows):
        A[it2, :] = A[it2, :] - A[it1, :] * A[it2, it1] / A[it1, it1]
```

3 **Pre for petlji** odrediti dimenziju matrice:

```
rows = A.shape[0]
```

4 Testirati funkciju *upper_triangular* na primeru:

```
A = np.array([
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
upper_triangular(A)
```

Rezultat:

```
[[ 5.          8.          5.    ]
 [ 0.         -4.4         3.    ]
 [ 0.          0.         -5.3182]]
```

Zadatak 2. Dopuniti funkciju *upper_triangular* tako da transformiše i vektor kolone slobodnih članova b, da bi sistem jednačina ostao ekvivalentan:

1 Modifikovati zaglavlje funkcije *upper_triangular*, tako da prihvata vektor kolone slobodnih članova b. Dodati povratnu vrednost koja predstavlja modifikovani vektor kolone slobodnih članova b:

```
def upper_triangular(A, b)
    .
    .
    .
   return A, b
```

2 Da bi sistem ostao ekvivalentan, svaka transformacija svake vrste matrice A mora se odraziti na odgovarajuću vrstu vektora b, uz napomenu da vektor b ima samo vrste, ali ne i kolone:

```
A[it2, :] = A[it2, :] - A[it1, :] * A[it2, it1] / A[it1, it1]
b[it2] = b[it2] – b[it1] * A[it2, it1] / A[it1, it1]
```

3 Primetiti da se množilac izračunava 2 puta. Postupak je moguće optimizovati:

```
m = A[it2, it1] / A[it1, it1]
A[it2, :] = A[it2, :] - A[it1, :] * m
b[it2] = b[it2] - b[it1]*m
```

4 Testirati funkciju *upper_triangular* na primeru:

```
A = np.array([
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
A, b = upper_triangular(A, b)
```

```
b = np.array([48, 43, 69])
```

Rezultat:

```
A =[
    [5.0000    8.0000    5.0000]
    [     0   -4.4000    3.0000]
    [     0         0   -5.3182]]
```

```
b = [48.0000    4.6000   -15.9545]
```

Zadatak 3. Definisati funkciju *solve_upper_triangular* koja za sistem jednačina sveden na gornji trougaoni oblik (zadan kvadratnom matricom A i vektorom b) nalazi vektor rešenja x:

Pokušati prvo ručno izračunavanje vektora x.

1 Inicijalizovati vektor rešenja x:

```
x = np.array([0, 0, 0])
```

Ako je:

```
A =
    5.0000    8.0000    5.0000
         0   -4.4000    3.0000
         0         0   -5.3182
```

```
b =
   48.0000
    4.6000
  -15.9545
```

$5.0000\, x_1 + 8.0000\, x_2 + 5.0000\, x_3 = 48.0000$

$\cancel{0.0000\, x_1} - 4.4000\, x_2 + 3.0000\, x_3 = 4.6000$

$\cancel{0.0000\, x_1 + 0.0000\, x_2} - 5.3182\, x_3 = -15.9545$

2 x_3, a zatim x_2 i na kraju x_1 se mogu izračunati direktno:

```
x[2] = b[2] / A[2, 2]
x[1] = (b[1] - A[1, 2] * x[2]) / A[1, 1]
x[0] = (b[0] - A[0, 2] * x[2] - A[0, 1] * x[1]) / A[0, 0]
```

Rezultat:

```
x = [5.0000    1.0000    3.0000]
```

Potrebno je definisati algoritam koji će ovo uraditi za proizvoljnu kvadratnu matricu.

1 Definisati praznu funkciju *solve_upper_triangular*:

```
def solve_upper_triangular(A, b):

   return x
```

Uporediti korake:
```
x[2] = (b[2]                                  ) / A[2, 2]
x[1] = (b[1]                  - A[1, 2] * x[2]) / A[1, 1]
x[0] = (b[0] - A[0, 1] * x[1] - A[0, 2] * x[2]) / A[0, 0]
```

Pretvoriti u vektorski zapis:
```
x[2] = (b[2] - [                ] * [         ]) / A[2, 2]
x[1] = (b[1] - [        A[1, 2]] * [     x[2]]) / A[1, 1]
x[0] = (b[0] - [A[0, 1] A[0, 2]] * [x[1] x[2]]) / A[0, 0]
```

Zapisati vektore kao izraze:
```
x[2] = (b[2] - np.matmul(A[2, 3:], x[3:])) / A[2, 2]
x[1] = (b[1] - np.matmul(A[1, 2:], x[2:])) / A[1, 1]
x[0] = (b[0] - np.matmul(A[0, 1:], x[1:])) / A[0, 0]
```

Proširiti indekse:
```
x[2] = (b[2] – np.matmul(A[2, (2 + 1):], x[(2 + 1):])) / A[2, 2]
x[1] = (b[1] - np.matmul(A[1, (1 + 1):], x[(1 + 1):])) / A[1, 1]
x[0] = (b[0] - np.matmul(A[0, (0 + 1):], x[(0 + 1):])) / A[0, 0]
```

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

```
for it in range(rows-1, -1, -1):
   x[it] = (b[it] - np.matmul(A[it, (it + 1):], x[(it + 1):])) / A[it, it]
```

3 **Pre for petlje** odrediti dimenziju matrice i inicijalizovati vektor *x*:

```
rows = A.shape[0]
x = np.zeros(rows)
```

4 Testirati funkciju *solve_upper_triangular.m* na primeru:
```
A = np.array([                                  b = np.array([48, 43, 69])
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
A, b = upper_triangular(A, b)
x = solve_upper_triangular(A, b)
```

Rezultat:
```
x = [5.0000    1.0000    3.0000]
```

Zadatak 4. Objediniti pozive funkcija *upper_triangular* i *solve_upper_triangular* u jedan poziv funkcije *gauss*.

1 Definisati funkciju *gauss*:

```
def gauss(A, b)
   A, b = upper_triangular(A, b)
   x = solve_upper_triangular(A, b)
   return x
```

2 Testirati funkciju *gauss* na primeru, izračunati tačnu vrednost upotrebom operatora `linalg.solve` iz biblioteke *numpy* i izračunati apsolutnu grešku:

```
A = np.array([                              b = np.array([48, 43, 69])
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
x = gauss(A, b)
xt = np.linalg.solve(A, b)
diff = np.abs(x - xt)
```

Rezultat:

```
x = [5.0000    1.0000    3.0000]

xt = [5     1     3]

diff = [0.0000000000000000000    0.0000000000000000000    0.0000000000000000000]
```

Zadatak 5. Definisati funkciju *gauss_pp* koja rešava sistem jednačina Gausovom eliminacijom, uvodeći parcijalni *pivoting*.
Pokušati prvo ručno pronalaženje i zamenu pivotske vrste.

```
A = [                                       b = [48    43    69]
     5      8      5
     4      2      7
     8      5      8]
```

Za eliminaciju elemenata u 1. koloni ispod 1. vrste najpogodniji je pivotski element u 3. vrsti jer ima najveću apsolutnu vrednost.

1 Izolovati 1. kolonu matrice:

```
A[:, 0]
```

Rezultat:

```
[5. 4. 8.]
```

2 Naći apsolutne vrednosti 1. kolone matrice:

```
np.abs(A[:, 0])
```

Rezultat:
```
[5. 4. 8.]
```

3 Naći indeks maksimalnog elementa po apsolutnoj vrednosti 1. kolone matrice:

```
mi = np.argmax(np.abs(A[:, 0]))
```

Rezultat:
```
mi = 2
```

```
A =
     5     8     5
     4     2     7
     8     5     8
```

Za eliminaciju elemenata u 2. koloni ispod 2. vrste najpogodniji je pivotski element u 3. vrsti jer ima najveću apsolutnu vrednost.

4 Naći indeks maksimalnog elementa po apsolutnoj vrednosti 2. kolone matrice ispod 2. vrste matrice:

```
mi = np.argmax(np.abs(A[1:, 1]))
```

Rezultat:
```
mi = 1
```

Indeks 1 jeste indeks po apsolutnoj vrednosti najvećeg elementa 2. kolone **ako bi se indeksi brojali od 2. vrste**, tj. ako bi 2. vrsta imala indeks 0 (to se dešava jer je u izrazu kolona efektivno odsečena do 2. vrste).

5 Korigovati indeks po apsolutnoj vrednosti najvećeg elementa 2. kolone dodajući mu indeks 2. vrste, tako da bude validan za celu kolonu:

```
mi = np.argmax(np.abs(A[1:, 1]))
mi = mi + 1
```

Rezultat:
```
mi = 2
```

6 Primeniti korigovani izraz za ponovno pronalaženje indeksa pivotskog elementa u 1. koloni, a zatim zameniti 1. vrstu sa vrstom pronađenog pivotskog elementa matrice A i primeniti istu transformaciju na vektor b da bi sistem ostao ekvivalentan:

```
mi = np.argmax(np.abs(A[0:, 0]))
mi = mi + 0
A[[0, mi],:] = A[[mi, 0],:]
b[[0, mi]] = b[[mi, 0]]
```

Rezultat:

```
A = [
     [8,     5,     8],
     [4,     2,     7],
     [5,     8,     5]]

b = [69    43    48]
```

Uporediti korake:

- za 1. vrstu:
```
mi = np.argmax(np.abs(A[0:, 0]))
mi = mi + 0
A[[0, mi],:] = A[[mi, 0],:]
b[[0, mi]] = b[[mi, 0]]
```
- za 2. vrstu:
```
mi = np.argmax(np.abs(A[1:, 1]))
mi = mi + 1
A[[1, mi],:] = A[[mi, 1],:]
b[[1, mi]] = b[[mi, 1]]
```
- za 3. vrstu nema elemenata ispod glavne dijagonale

Primetiti šta je promenljivo. Promenljivi indeksi rastu od 0 do dimenzije matrice - 2.

1 Napraviti kopiju funkcije *upper_triangular* i nazvati je *upper_triangular_PP*. Uklopiti blok za odabir i zamenu pivotske vrste u spoljašnjoj *for* petlji funkcije *upper_triangular_PP*, pre eliminacije, tako da promenljivi indeksi zavise od indeksa spoljašnje for petlje:

```
def upper_triangular_pp(A, b)
    .
    .
    .
   for it1 in range(0, rows-1):
       mi = np.argmax(np.abs(A[it1:, it1]))
       mi = mi + it1
       A[[it1, mi],:] = A[[mi, it1],:]
       b[[it1, mi]] = b[[mi, it1]]
       for it2 in range(it1 + 1, rows):
    .
    .
    .
```

2 Napraviti kopiju funkcije *gauss* i nazvati je *gauss_pp*. U njoj umesto funkcije *upper_triangular* pozvati funkciju *upper_triangular_pp*:

```
def gauss_pp(A, b)
   A, b = upper_triangular_pp(A, b)
   x = solve_upper_triangular(A, b)
   return x
```

3 Testirati funkciju *gauss_pp* na primeru, izračunati tačnu vrednost upotrebom upotrebom operatora *`linalg.solve`* iz biblioteke *numpy* i izračunati apsolutnu grešku:

```
A = np.array([
    [5, 8, 5],
    [4, 2, 7],
    [8, 5, 8]])
x = gauss_pp(A, b)
xt = np.linalg.solve(A, b)
diff = np.abs(x - xt)
```

```
b = np.array([48, 43, 69])
```

Rezultat:

Rezultat:

```
x = [5    1    3]

xt = [5     1     3]

diff = [0    0    0]
```

* **Zadatak 6**. Definisati funkciju *gauss_CP* koja rešava sistem jednačina Gausovom eliminacijom, uvodeći kompletan *pivoting*.

* **Zadatak 7**. Definisati funkciju *gauss_LT* koja rešava sistem jednačina Gausovom eliminacijom, svodeći sistem na donji trougaoni oblik.