test-01_matrix

October 23, 2020

1 Analiza i projektiranje računalom: 1. laboratorijska vježba

1.1 Test klase Matrix

1.1.1 Priprema

[1]:
```
import os

CD_KEY = "--MATRIX_TEST_IN_ROOT"
```

[2]:
```
if (
    CD_KEY not in os.environ
    or os.environ[CD_KEY] is None
    or len(os.environ[CD_KEY]) == 0
    or os.environ[CD_KEY] == "false"
):
    %cd ..
else:
    print(os.getcwd())

os.environ[CD_KEY] = "true"
```

```
/mnt/data/projekti/faks/AIPR/dz/dz-01
```

1.1.2 Učitavanje paketa

[3]:
```
from math import log10
from textwrap import dedent

import numpy as np

from src.matrices.matrix import Matrix
from src.matrices.exceptions import MatrixIsSingular, NotSolvable
```

1.1.3 Provjere pristupa

[4]:
```
SHAPE = (5, 5)
DTYPE = float
FILL_VALUE = 17.29
```

[5]:
```
our_matrix = Matrix.full(5, 5, fill_value=FILL_VALUE, dtype=DTYPE)
print(our_matrix)
```

```
[
  [17.290 17.290 17.290 17.290 17.290]
  [17.290 17.290 17.290 17.290 17.290]
  [17.290 17.290 17.290 17.290 17.290]
  [17.290 17.290 17.290 17.290 17.290]
  [17.290 17.290 17.290 17.290 17.290]
]
```

Provjera manipulacije elemenata

[6]:
```
our_matrix[1][3] = 941

print(f"Oblik: {our_matrix.shape}")
print(f"Matrica: {our_matrix}")
```

```
Oblik: (5, 5)
Matrica: [
  [ 17.290  17.290  17.290  17.290  17.290]
  [ 17.290  17.290  17.290 941.000  17.290]
  [ 17.290  17.290  17.290  17.290  17.290]
  [ 17.290  17.290  17.290  17.290  17.290]
  [ 17.290  17.290  17.290  17.290  17.290]
]
```

[7]:
```
our_matrix.int()
print(f"Matrica pretvorena u cijele brojeve: {our_matrix}")
```

```
Matrica pretvorena u cijele brojeve: [
  [ 17  17  17  17  17]
  [ 17  17  17 941  17]
  [ 17  17  17  17  17]
  [ 17  17  17  17  17]
  [ 17  17  17  17  17]
]
```

[8]:
```
our_matrix.float()
print(f"Matrica pretvorena natrag u float: {our_matrix}")
```

```
Matrica pretvorena natrag u float: [
  [ 17.000  17.000  17.000  17.000  17.000]
  [ 17.000  17.000  17.000 941.000  17.000]
  [ 17.000  17.000  17.000  17.000  17.000]
  [ 17.000  17.000  17.000  17.000  17.000]
  [ 17.000  17.000  17.000  17.000  17.000]
]
```

[9]:
```
print(f"Matrica nula: {Matrix.zeros(3, 3)}")
```

```
Matrica nula: [
  [0.000 0.000 0.000]
  [0.000 0.000 0.000]
  [0.000 0.000 0.000]
]
```

[10]:
```
print(f"Jedinična matrica: {Matrix.eye(5, 6, int)}")
```

```
Jedinična matrica: [
  [1 0 0 0 0 0]
  [0 1 0 0 0 0]
  [0 0 1 0 0 0]
  [0 0 0 1 0 0]
  [0 0 0 0 1 0]
]
```

Provjera podmatrica

[11]:
```
MATRIX_TO_USE = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

numpy_matrix_2 = np.array(MATRIX_TO_USE, dtype=float)
our_matrix_2 = Matrix.from_array(MATRIX_TO_USE)
```

[12]:
```
print(our_matrix_2.diagonal)
```

```
[[1 5 9]]
```

[13]:
```
print(our_matrix_2.reverse_diagonal)
```

```
[[3 5 7]]
```

[14]:
```
print(our_matrix_2.row(1))
```

```
[[4 5 6]]
```

[15]:
```
print(our_matrix_2.column(1))
```

```
[
  [2]
  [5]
  [8]
]
```

1.1.4 Provjera aritmetike

Zbrajanje

[16]:

```
base_arithmetic_matrix = Matrix.from_array(
    [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
)
```

[17]:

```
print(base_arithmetic_matrix + 1)
```

```
[
  [ 2  3  4]
  [ 5  6  7]
  [ 8  9 10]
]
```

[18]:

```
print(base_arithmetic_matrix + 3.14)
```

```
[
  [ 4.140  5.140  6.140]
  [ 7.140  8.140  9.140]
  [10.140 11.140 12.140]
]
```

[19]:

```
print(base_arithmetic_matrix + Matrix.eye(3, 3, int))
```

```
[
  [ 2  2  3]
  [ 4  6  6]
  [ 7  8 10]
]
```

Oduzimanje

[20]:

```
print(base_arithmetic_matrix - 1)
```

```
[
  [0 1 2]
  [3 4 5]
  [6 7 8]
]
```

[21]:
```python
print(base_arithmetic_matrix - 3.14)
```

```
[
  [-2.140 -1.140 -0.140]
  [ 0.860  1.860  2.860]
  [ 3.860  4.860  5.860]
]
```

[22]:
```python
print(base_arithmetic_matrix - Matrix.eye(3, 3, int))
```

```
[
  [0 2 3]
  [4 4 6]
  [7 8 8]
]
```

Množenje

[23]:
```python
print(base_arithmetic_matrix * 3)
```

```
[
  [ 3  6  9]
  [12 15 18]
  [21 24 27]
]
```

[24]:
```python
print(base_arithmetic_matrix * 3.14)
```

```
[
  [ 3.140  6.280  9.420]
  [12.560 15.700 18.840]
  [21.980 25.120 28.260]
]
```

[25]:
```python
print(base_arithmetic_matrix * Matrix.eye(3, 3, int))
```

```
[
  [1 0 0]
  [0 5 0]
  [0 0 9]
]
```

Matrično množenje

[26]:
```python
print(base_arithmetic_matrix @ base_arithmetic_matrix)
```

```
[
  [ 30  36  42]
  [ 66  81  96]
  [102 126 150]
]
```

[27]:

```
print(base_arithmetic_matrix @ Matrix.eye(3, 3, int))
```

```
[
  [1 2 3]
  [4 5 6]
  [7 8 9]
]
```

[28]:

```
print(base_arithmetic_matrix @ Matrix.from_array(
    [
        [1],
        [1],
        [1]
    ]
))
```

```
[
  [ 6]
  [15]
  [24]
]
```

[29]:

```
row_matrix = Matrix.from_array(
    [
        [1, 2, 3]
    ]
)

column_matrix = Matrix.from_array(
    [
        [2],
        [3],
        [4]
    ]
)

print(int(row_matrix @ column_matrix))
```

```
20
```

Dijeljenje

[30]:

```
print(base_arithmetic_matrix / 3)
```

```
[
  [0.333 0.667 1.000]
  [1.333 1.667 2.000]
  [2.333 2.667 3.000]
]
```

[31]:

```
print(base_arithmetic_matrix / 3.14)
```

```
[
  [0.318 0.637 0.955]
  [1.274 1.592 1.911]
  [2.229 2.548 2.866]
]
```

[32]:

```
print(base_arithmetic_matrix / base_arithmetic_matrix)
```

```
[
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
]
```

Cjelobrojno dijeljenje

[33]:

```
print(base_arithmetic_matrix // 3)
```

```
[
  [0 0 1]
  [1 1 2]
  [2 2 3]
]
```

[34]:

```
print(base_arithmetic_matrix // 3.14)
```

```
[
  [0 0 0]
  [1 1 1]
  [2 2 2]
]
```

[35]:

```
print(base_arithmetic_matrix // Matrix.full(3, 3, 2))
```

```
[
  [0 1 1]
  [2 2 3]
  [3 4 4]
]
```

Transponiranje

[36]:
```
base_arithmetic_matrix.transpose()
print(base_arithmetic_matrix)
```

```
[
  [1 4 7]
  [2 5 8]
  [3 6 9]
]
```

[37]:
```
print(base_arithmetic_matrix.transposed())
print(base_arithmetic_matrix)
```

```
[
  [1 2 3]
  [4 5 6]
  [7 8 9]
]
[
  [1 4 7]
  [2 5 8]
  [3 6 9]
]
```

[38]:
```
base_arithmetic_matrix = base_arithmetic_matrix.T
print(base_arithmetic_matrix)
```

```
[
  [1 2 3]
  [4 5 6]
  [7 8 9]
]
```

Modul

[39]:
```
print(base_arithmetic_matrix % 3)
```

```
[
  [1 2 0]
  [1 2 0]
  [1 2 0]
]
```

[40]:
```
print(base_arithmetic_matrix % 3.14)
```

```
[
  [1.000 2.000 3.000]
  [0.860 1.860 2.860]
  [0.720 1.720 2.720]
]
```

```
[41]: print(base_arithmetic_matrix % base_arithmetic_matrix.T)
```

```
[
  [0 2 3]
  [0 0 6]
  [1 2 0]
]
```

Eksponencijacija

```
[42]: print(base_arithmetic_matrix ** 3)
```

```
[
  [  1   8  27]
  [ 64 125 216]
  [343 512 729]
]
```

```
[43]: print(base_arithmetic_matrix ** 3.14)
```

```
[
  [  1.000   8.815  31.489]
  [ 77.708 156.591 277.584]
  [450.410 685.019 991.566]
]
```

```
[44]: print(base_arithmetic_matrix ** base_arithmetic_matrix.T)
```

```
[
  [        1        16      2187]
  [       16      3125   1679616]
  [      343    262144 387420489]
]
```

Negacija

```
[45]: print(-base_arithmetic_matrix)
```

```
[
  [-1 -2 -3]
  [-4 -5 -6]
  [-7 -8 -9]
]
```

Apsolucija

[46]:

```
interesting_matrix = base_arithmetic_matrix - base_arithmetic_matrix.T

print(interesting_matrix)
print(abs(interesting_matrix))
```

```
[
  [ 0 -2 -4]
  [ 2  0 -2]
  [ 4  2  0]
]
[
  [0 2 4]
  [2 0 2]
  [4 2 0]
]
```

1.1.5 Provjera proširene aritmetike

[47]:

```
base_extended_arithmetic_matrix = Matrix.from_array(
    [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
)
```

Zbrajanje u mjestu

[48]:

```
base_extended_arithmetic_matrix += 1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 2  3  4]
  [ 5  6  7]
  [ 8  9 10]
]
```

[49]:

```
base_extended_arithmetic_matrix += 1.1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 3.100  4.100  5.100]
  [ 6.100  7.100  8.100]
  [ 9.100 10.100 11.100]
]
```

Oduzimanje u mjestu

[50]:

```
base_extended_arithmetic_matrix -= 1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 2.100  3.100  4.100]
  [ 5.100  6.100  7.100]
  [ 8.100  9.100 10.100]
]
```

[51]:

```
base_extended_arithmetic_matrix -= 1.1
print(base_extended_arithmetic_matrix)
```

```
[
  [1.000 2.000 3.000]
  [4.000 5.000 6.000]
  [7.000 8.000 9.000]
]
```

[52]:

```
base_extended_arithmetic_matrix.int();
```

Množenje u mjestu

[53]:

```
base_extended_arithmetic_matrix *= 2
print(base_extended_arithmetic_matrix)
```

```
[
  [ 2  4  6]
  [ 8 10 12]
  [14 16 18]
]
```

[54]:

```
base_extended_arithmetic_matrix *= 1.1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 2.200  4.400  6.600]
  [ 8.800 11.000 13.200]
  [15.400 17.600 19.800]
]
```

Matrično množenje u mjestu

[55]:

```
base_extended_arithmetic_matrix @= base_extended_arithmetic_matrix
print(base_extended_arithmetic_matrix)
```

```
[
  [145.200 174.240 203.280]
  [319.440 392.040 464.640]
  [493.680 609.840 726.000]
]
```

Dijeljenje u mjestu

[56]:
```
base_extended_arithmetic_matrix /= 2
print(base_extended_arithmetic_matrix)
```

```
[
  [ 72.600  87.120 101.640]
  [159.720 196.020 232.320]
  [246.840 304.920 363.000]
]
```

[57]:
```
base_extended_arithmetic_matrix /= 1.1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 66.000  79.200  92.400]
  [145.200 178.200 211.200]
  [224.400 277.200 330.000]
]
```

Cjelobrojno dijeljenje u mjestu

[58]:
```
base_extended_arithmetic_matrix //= 2
print(base_extended_arithmetic_matrix)
```

```
[
  [ 33.000  39.000  46.000]
  [ 72.000  89.000 105.000]
  [112.000 138.000 165.000]
]
```

[59]:
```
base_extended_arithmetic_matrix //= 1.1
print(base_extended_arithmetic_matrix)
```

```
[
  [ 29.000  35.000  41.000]
  [ 65.000  80.000  95.000]
  [101.000 125.000 149.000]
]
```

[60]:
```
base_extended_arithmetic_matrix += Matrix.from_array(
    [
        [1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]
    ]
)
base_extended_arithmetic_matrix.int()

print(base_extended_arithmetic_matrix)
```

```
[
  [ 30  37  44]
  [ 66  82  98]
  [102 127 152]
]
```

Modul u mjestu

[61]:

```
base_extended_arithmetic_matrix %= 4
print(base_extended_arithmetic_matrix)
```

```
[
  [2 1 0]
  [2 2 2]
  [2 3 0]
]
```

[62]:

```
base_extended_arithmetic_matrix %= 2.5
print(base_extended_arithmetic_matrix)
```

```
[
  [2.000 1.000 0.000]
  [2.000 2.000 2.000]
  [2.000 0.500 0.000]
]
```

Eksponencijacija u mjestu

[63]:

```
base_extended_arithmetic_matrix **= 2.5
print(base_extended_arithmetic_matrix)
```

```
[
  [5.657 1.000 0.000]
  [5.657 5.657 5.657]
  [5.657 0.177 0.000]
]
```

[64]:

```
base_extended_arithmetic_matrix.int()
print(base_extended_arithmetic_matrix)
```

```
[
  [6 1 0]
  [6 6 6]
  [6 0 0]
]
```

[65]:

```
base_extended_arithmetic_matrix **= 2
print(base_extended_arithmetic_matrix)
```

```
[
  [36  1  0]
  [36 36 36]
  [36  0  0]
]
```

1.1.6 Provjera usporedbe

[66]:

```
base_comparison_matrix = Matrix.full(3, 3, 1, int)
```

Jednakost

[67]:

```
print(base_comparison_matrix == 1)
```

```
False
```

[68]:

```
print(base_comparison_matrix == 1.0)
```

```
False
```

[69]:

```
equals_matrix_1 = Matrix.full(3, 3, 1, float)
equals_matrix_2 = Matrix.full(3, 3, 1, int)
equals_matrix_3 = Matrix.full(3, 3, 1, float) + 1e-6
equals_matrix_4 = Matrix.full(3, 3, 1, float) + (base_comparison_matrix.epsilon␣
 ↪/ 10)
```

[70]:

```
print(
f"""\
{base_comparison_matrix} == {equals_matrix_1}

{base_comparison_matrix == equals_matrix_1}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
]

True
```

[71]:

```
print(
f"""\
{base_comparison_matrix} == {equals_matrix_2}

{base_comparison_matrix == equals_matrix_2}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1 1 1]
  [1 1 1]
  [1 1 1]
]

True
```

Sada ćemo istestirati koje su granice usporedbe. Prvo krenimo s malom, ali dovoljno velikom devijacijom.

[72]:

```
print(
f"""\
{base_comparison_matrix} == \
{equals_matrix_3.pretty_print(decimal_precision=6)}

{base_comparison_matrix == equals_matrix_3}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
]

False
```

Sada ćemo istestirati što se događa ako je devijacija premala (u ovom slučaju 10 puta manjoj od dozvoljene).

[73]:

```
needed_precision = int(0.5 - log10(base_comparison_matrix.epsilon) + 1)

print(
f"""\
{base_comparison_matrix} == \
{equals_matrix_4.pretty_print(decimal_precision=needed_precision)}

{base_comparison_matrix == equals_matrix_4}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1.00000000000001 1.00000000000001 1.00000000000001]
  [1.00000000000001 1.00000000000001 1.00000000000001]
  [1.00000000000001 1.00000000000001 1.00000000000001]
]

True
```

Ovu granicu možemo i mijenjati, iako nije preporučljivo. Npr., ako želimo da nam 3. matrica bude jednaka, onda možemo napraviti sljedeće

[74]:

```
base_comparison_matrix.epsilon = 1e-5
```

[75]:

```
print(
f"""\
{base_comparison_matrix} == \
{equals_matrix_3.pretty_print(decimal_precision=6)}

{base_comparison_matrix == equals_matrix_3}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
]
```

```
True
```

Epsilon koji se gleda je uvijek onaj lijevog argumenta, pa tako imamo i ovakav rezultat

[76]:

```
base_comparison_matrix.epsilon = 1e-13
equals_matrix_3.epsilon = 1e-5
```

[77]:

```
print(
f"""\
{base_comparison_matrix} == \
{equals_matrix_3.pretty_print(decimal_precision=6)}

{base_comparison_matrix == equals_matrix_3}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] == [
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
  [1.000001 1.000001 1.000001]
]

False
```

Manje (jednako) od

[78]:

```
lt_matrix_1 = Matrix.full(3, 3, 2, int)
lt_matrix_2 = Matrix.full(3, 3, 2, float)
le_matrix_1 = Matrix.full(3, 3, 1, float)
le_matrix_2 = Matrix.full(3, 3, 1, float) - (base_comparison_matrix.epsilon /␣
 ↪10)
```

[79]:

```
print(
f"""\
{base_comparison_matrix} < {lt_matrix_1}

{base_comparison_matrix < lt_matrix_1}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] < [
  [2 2 2]
  [2 2 2]
  [2 2 2]
]

True
```

[80]:

```
print(
f"""\
{base_comparison_matrix} < {lt_matrix_2}

{base_comparison_matrix < lt_matrix_2}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] < [
  [2.000 2.000 2.000]
  [2.000 2.000 2.000]
  [2.000 2.000 2.000]
]

True
```

Slično radi i operacija $\leq$, pa ćemo samo provjeriti rubne slučajeve

[81]:

```
print(
f"""\
{base_comparison_matrix} <= {le_matrix_1}

{base_comparison_matrix <= le_matrix_1}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] <= [
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
]

True
```

[82]:

```
needed_precision = int(0.5 - log10(base_comparison_matrix.epsilon) + 1)

print(
f"""\
{base_comparison_matrix} <= \
{le_matrix_2.pretty_print(decimal_precision=needed_precision)}

{base_comparison_matrix <= le_matrix_2}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] <= [
  [0.99999999999999 0.99999999999999 0.99999999999999]
  [0.99999999999999 0.99999999999999 0.99999999999999]
  [0.99999999999999 0.99999999999999 0.99999999999999]
]

False
```

Vidimo da ovog puta ne toleriramo ni devijaciju manju od epsilona.

Veće (jednako) od

[83]:

```
gt_matrix_1 = Matrix.full(3, 3, 2, int)
gt_matrix_2 = Matrix.full(3, 3, 2, float)
ge_matrix_1 = Matrix.full(3, 3, 1, float)
ge_matrix_2 = Matrix.full(3, 3, 1, float) + (base_comparison_matrix.epsilon /␣
 ↪10)
```

[84]:

```
print(
f"""\
{base_comparison_matrix} > {gt_matrix_1}

{base_comparison_matrix > gt_matrix_1}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] > [
  [2 2 2]
  [2 2 2]
  [2 2 2]
]

False
```

[85]:

```python
print(
f"""\
{base_comparison_matrix} > {gt_matrix_2}

{base_comparison_matrix > gt_matrix_2}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] > [
  [2.000 2.000 2.000]
  [2.000 2.000 2.000]
  [2.000 2.000 2.000]
]

False
```

Slično kao i prije, testiramo samo rubne slučajeve na veće ili jednako

[86]:

```python
print(
f"""\
{base_comparison_matrix} >= {ge_matrix_1}

{base_comparison_matrix >= ge_matrix_1}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] >= [
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
  [1.000 1.000 1.000]
]

True
```

[87]:

```python
needed_precision = int(0.5 - log10(base_comparison_matrix.epsilon) + 1)
```

```
print(
f"""\
{base_comparison_matrix} >= \
{ge_matrix_2.pretty_print(decimal_precision=needed_precision)}

{base_comparison_matrix >= ge_matrix_2}\
"""
)
```

```
[
  [1 1 1]
  [1 1 1]
  [1 1 1]
] >= [
  [1.00000000000001 1.00000000000001 1.00000000000001]
  [1.00000000000001 1.00000000000001 1.00000000000001]
  [1.00000000000001 1.00000000000001 1.00000000000001]
]

False
```

Ni tu ne prihvaćamo devijaciju manju od epsilona.

1.1.7 Rješavanje matrica

U ove metode spadaju:

- supstitucija unaprijed
- supstitucija unatrag
- LU dekompozicija
- LUP dekompozicija
- izračun determinante matrice
- izračun inverza matrice

[88]:

```
base_matrix = Matrix.from_array(
    [
        [2, -1, 3],
        [1, 5, -1],
        [-1, -1, 1]
    ]
)
eye = Matrix.eye(3, 3, int)

print(base_matrix.forward_substitute(eye.row(0)))
```

```
[[ 1 -1  0]]
```

[89]:

```
base_row = Matrix.from_array(
    [
        [1, 2, 3]
    ]
)
base_invertible_matrix = Matrix.from_array(
    [
        [1, 2, 0],
        [3, 5, 4],
        [5, 6, 3]
    ]
)
base_singular_matrix = Matrix.from_array(
    [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
)
```

Supstitucija unaprijed

[90]:

```
forward_row = base_invertible_matrix.forward_substitute(base_row)
print(forward_row)
```

```
[[ 1 -1  4]]
```

Supstitucija unaprijed radi i za singularne matrice

[91]:

```
forward_row_2 = base_singular_matrix.forward_substitute(base_row)
print(forward_row_2)
```

```
[[ 1 -2 12]]
```

Supstitucija unatrag Ovu supstituciju radimo nad prethodno dobivenim retcima

[92]:

```
backward_row = base_invertible_matrix.backward_substitute(forward_row)
print(backward_row)
```

```
[[ 3.533 -1.267  1.333]]
```

[93]:

```
try:
    backward_row_2 = base_singular_matrix.backward_substitute(forward_row_2)
    print(backward_row_2)
except NotSolvable as ns:
    print(ns)
```

```
[[ 1.000 -2.000  1.333]]
```

LU dekompozicija

[94]:

```
lu_matrix = base_invertible_matrix.lu()
print(lu_matrix)
```

```
[
  [  1   2   0]
  [  3  -1   4]
  [  5   4 -13]
]
```

Ovu matricu možemo i razdvojiti na L i U matricu:

[95]:

```
l_matrix, u_matrix = Matrix.split_lu_matrix(lu_matrix)
print(l_matrix)
print(u_matrix)
```

```
[
  [1 0 0]
  [3 1 0]
  [5 4 1]
]
[
  [  1   2   0]
  [  0  -1   4]
  [  0   0 -13]
]
```

Ako pokušamo izvršiti LU dekompoziciju na singularnoj matrici, metoda treba vratiti iznimku **NotSolvable**

[96]:

```
try:
    lu_matrix_2 = base_singular_matrix.lu()
except NotSolvable as ns:
    print(ns)
```

```
Encountered a zero pivot in method Matrix.lu: Matrix[2][2] is the culprit (in [
  [ 1  2  3]
  [ 4 -3 -6]
  [ 7  2  0]
]).
```

LUP dekompozicija

[97]:

```
lu_matrix_3, p_matrix_3 = base_invertible_matrix.lup()
print(lu_matrix_3)
print(p_matrix_3)
```

```
[
  [ 5.000  6.000  3.000]
  [ 0.600  1.400  2.200]
  [ 0.200  0.571 -1.857]
]
[
  [0 0 1]
  [0 1 0]
  [1 0 0]
]
```

Slično kao i prije, ni LUP dekompozicija ne bi trebala biti otporna na singularne matrice

[98]:

```python
try:
    lu_matrix_4, p_matrix_4 = base_singular_matrix.lup()
    print(lu_matrix_4)
    print(p_matrix_4)
except NotSolvable as ns:
    print(ns)
```

```
Encountered a zero pivot in method Matrix.lup: Matrix[2][2] is the culprit (in [
  [7.000 8.000 9.000]
  [0.143 0.857 1.714]
  [0.571 0.500 0.000]
]).
```

Izračun determinante matrice

[99]:

```python
print(
    f"Determinanta {base_invertible_matrix} = "
    f"{base_invertible_matrix.determinant}"
)
```

```
Determinanta [
  [1 2 0]
  [3 5 4]
  [5 6 3]
] = 13
```

[100]:

```python
print(
    f"Determinanta {base_singular_matrix} = "
    f"{base_singular_matrix.determinant}"
)
```

```
Determinanta [
  [1 2 3]
  [4 5 6]
  [7 8 9]
] = 0
```

Izračun inverza matrice

[101]:

```
base_invertible_matrix_inverse = base_invertible_matrix.float().inverse
print(
    f"Inverz {base_invertible_matrix} = "
    f"{base_invertible_matrix_inverse}"
)
```

```
Inverz [
  [1.000 2.000 0.000]
  [3.000 5.000 4.000]
  [5.000 6.000 3.000]
] = [
  [-0.692 -0.462  0.615]
  [ 0.846  0.231 -0.308]
  [-0.538  0.308 -0.077]
]
```

Možemo provjeriti da je matrica stvarno inverz jednim matričnim množenjem

[102]:

```
print((base_invertible_matrix @ base_invertible_matrix_inverse).float())
```

```
[
  [1.000 0.000 0.000]
  [0.000 1.000 0.000]
  [0.000 0.000 1.000]
]
```

Ako pokušamo invertirati singularnu matricu, dignut će nam se **MatrixIsSingular** iznimka

[103]:

```
try:
    print(
        f"Inverz {base_singular_matrix} = "
        f"{base_singular_matrix.inverse}"
    )
except MatrixIsSingular as mis:
    print(mis)
```

```
Matrix [
  [1 2 3]
  [4 5 6]
  [7 8 9]
] is singular.
```