5. Interpolacija

1. Date su tačke:

```
x = np.array([0.7854    1.9635    3.1416    4.3197    5.4978])
fX = np.array([0.7071    0.9239    0.0000   -0.9239   -0.7071])
```

2. Nacrtati poznate tačke:

```
plt.scatter(x, fX)
```

3. Naći polinom koji zadovoljava poznate tačke:

```
order = x.size – 1 # red polinoma mora biti za 1 manji od broja tačaka
p = np.polyfit(x, fX, order)
```

Rezultat:

```
p =
   -0.0000    0.1163   -1.0958    2.4970   -0.6344
```

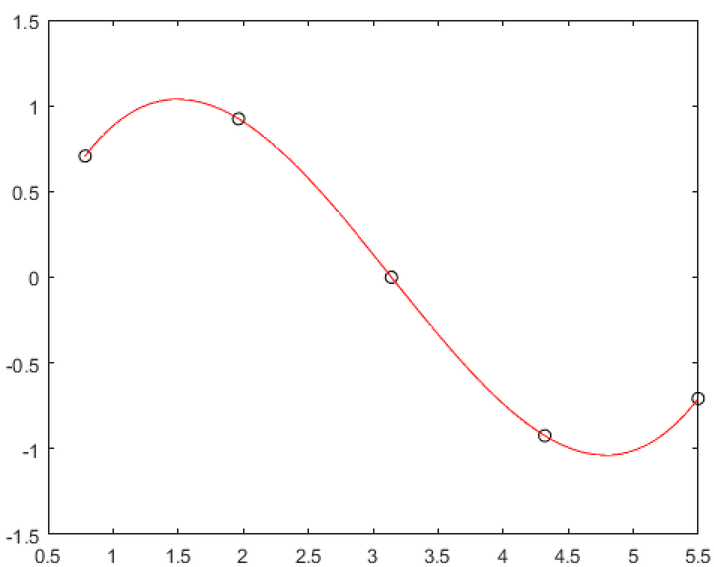
Pronađeni polinom se čita na sledeći način:

$$p(x) = -0.0000\,x^4 + 0.1163\,x^3 - 1.0958\,x^2 + 2.4970\,x - 0.6344$$

4. Nacrtati pronađeni polinom:

```
x = np.linspace(np.min(x), np.max(x), 100)
pX = np.polyval(p, x) # računanje vrednosti polinoma p za svaku vrednost vektora x
plt.plot(x, pX, 'red')
```

Rezultat:

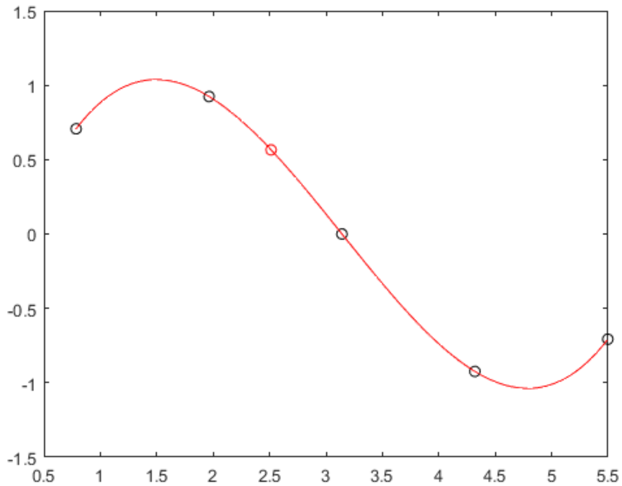


Slika 1. Polinom

5. Interpolacijom naći nepoznatu vrednost u tački $x_1 = \frac{4\pi}{5}$, a zatim na istom grafiku nacrtati dobijenu tačku:

```
x1 = 4 * np.pi / 5
pX1 = np.polyval(p, x1)
plt.scatter(x1, pX1, c='red')
```

Rezultat:
```
pX1 =
    0.5653
```

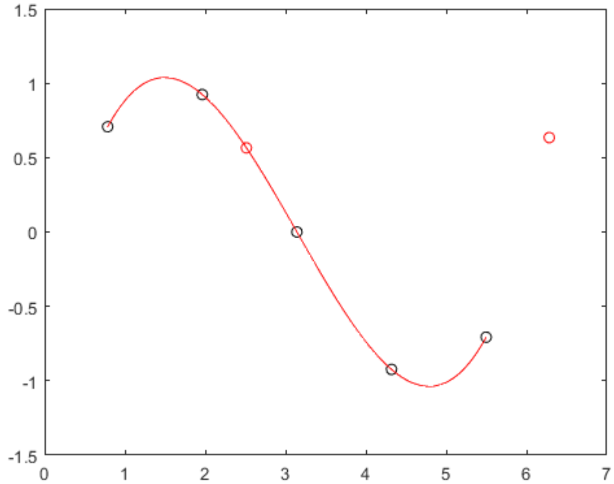


Slika 2. Interpolacija

6. Ekstrapolacijom naći nepoznatu vrednost u tački $x_2 = 2\pi$, a zatim na istom grafiku nacrtati dobijenu tačku:

```
x2 = 2 * np.pi
pX2 = np.polyval(p, x2)
plt.scatter(x2, pX2, c='red')
```

Rezultat:
```
pX2 =
    0.6344
```



Slika 3. Ekstrapolacija

7. Stvarna funkcija koja zadovoljava tačke je $f(x)=\sin x$. Na istom grafiku nacrtati funkciju na intervalu $x \in [0,3\pi]$, a zatim naći apsolutnu grešku u odnosu na stvarne vrednosti funkcije u tim tačkama:

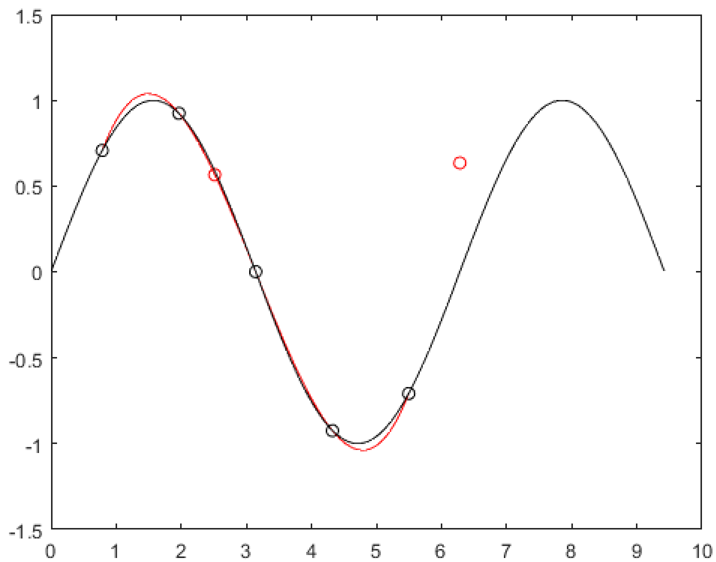
```
f = lambda x: np.sin(x)

x = np.linspace(0, 3*pi, 100)
fX = f(x)
plt.plot(x, fX, 'black')

errAbs1 = np.abs(np.sin(x1) - pX1)
errAbs2 = np.abs(np.sin(x2) - pX2)
```

Rezultat:
```
errAbs1 =
    0.0225

errAbs2 =
    0.6344
```



Slika 4. Stvarna funkcija

Ekstrapolacija je rezultovala daleko većom greškom od interpolacije!

Množenje (konvolucija) polinoma

Primer 1
Data su dva polinoma:

$$p_1(x) = x - 1$$
$$p_2(x) = x - 2$$

Njihov proizvod je:

$$p_1(x)\,p_2(x) = (x-1)(x-2) = x^2 - 2x - x + 2 = x^2 - 3x + 2$$

Prethodni rezultat se mogao naći ugrađenom Numpy funkcijom `np.conv(p1, p2)`

```
p1 = np.array([1, -1])
p2 = np.array([1, -2])

np.convolve(p1, p2)
```

Rezultat:

```
[1    -3     2]
```

Lagranžova interpolacija

Zadatak 1. Napisati metodu za traženje Lagranžovog polinoma koji zadovoljava proizvoljan broj poznatih tačaka.
Pokušati prvo ručno nalaženje koeficijenata Lagranžovog polinoma za skup tačaka:

```
x = np.array([0.7854,    1.9635,    3.1416,    4.3197,    5.4978])
fX = np.array([0.7071,    0.9239,    0.0000,   -0.9239,   -0.7071]
```

S obzirom na to da je dato 5 tačaka, traženi Lagranžov polinom je 4. reda:

$$p(x) = L_1 f(x_1) + L_2 f(x_2) + L_3 f(x_3) + L_4 f(x_4) + L_5 f(x_5) p(x) = \frac{(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)} f(x_1)$$
$$\frac{+(x-x_1)(x-x_3)(x-x_4)(x-x_5)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)(x_2-x_5)} f(x_2)$$
$$\frac{+(x-x_1)(x-x_2)(x-x_4)(x-x_5)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)(x_3-x_5)} f(x_3)$$
$$\frac{+(x-x_1)(x-x_2)(x-x_3)(x-x_5)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)(x_4-x_5)} f(x_4)$$
$$\frac{+(x-x_1)(x-x_2)(x-x_3)(x-x_4)}{(x_5-x_1)(x_5-x_2)(x_5-x_3)(x_5-x_4)} f(x_5)$$

Primetiti da je x jedina nepoznata koja figuriše u jednačini. Sve ostale vrednosti (x_1, $f(x_1)$, x_2, $f(x_2)$, x_3, $f(x_3)$, x_4, $f(x_4)$, x_5 i $f(x_5)$,) se mogu uvrstiti.

Primetiti da i u imeniocu i u brojiocu razlomka koji množe svaki $f(x_i)$ ne figurišu članovi koji sadrže x_i.

1. Red polinoma se izračunava na sledeći način:

```
order = x.size – 1
```

Posmatrati npr. 2. element sume:

$$\frac{(x-x_1)(x-x_3)(x-x_4)(x-x_5)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)(x_2-x_5)}f(x_2)$$

On se može izračunati sledećim Python izrazom:

```
lNumer = np.convolve([1, -x[0]], \
                  np.convolve([1, -x[2]], \
                        np.convolve([1, -x[3]], [1, -x[4]])))  # krecemo od zadnjeg

lDenom = (x[1] - x[0])*(x[1] - x[2])*(x[1] - x[3])*(x[1] - x[4])
print(lNumer/lDenom*fX[1])
```

Rezultat:

```
[-0.0799    1.0987   -5.1775    9.3914   -4.6841]
```

Uz napomenu da su brojilac (`lNumer`) i imenilac (`lDenom`) kumulativini proizvodi, izrazi se mogu razviti:

```
lNumer = 1
lDenom = 1
lNumer = np.convolve(lNumer, [1, -x[0]])
lDenom = lDenom*(x[1] - x[0])
lNumer = conv(lNumer, [1 -x[2]])
lDenom = lDenom*(x[1] - x[2])
lNumer = conv(lNumer, [1 -x[3]])
lDenom = lDenom*(x[1] - x[3])
lNumer = conv(lNumer, [1 -x[4]])
lDenom = lDenom*(x[1] – x[4])
print(lNumer/lDenom*fX[1])
```

Primetiti šta je fiksno, a šta promenljivo. Promenljivi indeksi rastu od 0 do fiksnog indeksa i od fiksnog indeksa + 1 do reda polinoma + 1. Ovo se može zapisati sa 2 *for* petlje pri čemu promenljivi indeksi zavise od indeksa *for* petlji:

```
lNumer = 1
lDenom = 1
for itX in range(0, 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[1] - x[itX])

for itX in range(2, order + 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[1] – x[itX])

print(lNumer/lDenom*fX[1])
```

Ceo polinom se može izračunati kao kumulativna suma svih elemenata:

```
p = 0

lNumer = 1
lDenom = 1
for itX in range(0, 0):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[0] - x[itX])

for itX in range(0 + 1, order + 1):
    lNumer = np.convolve(lNumer, [1 -x(itX)])
    lDenom = lDenom*(x[0] - x[itX])

p = p + lNumer/lDenom*fX[0]

lNumer = 1
lDenom = 1
for itX in range(0, 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[1] - x[itX])

for itX in range(1 + 1, order + 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[1] - x[itX])

p = p + lNumer/lDenom*fX[1]

lNumer = 1
lDenom = 1
for itX in range(0, 2):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[2] - x[itX])

for itX in range(2 + 1, order + 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[2] - x[itX])

p = p + lNumer/lDenom*fX[2]

lNumer = 1
lDenom = 1
for itX In range(0, 3):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[3] - x[itX])

for itX in range(3 + 1, order + 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[3] – x[itX])

p = p + lNumer/lDenom*fX(3)

lNumer = 1
lDenom = 1
for itX in range(0, 4):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[4] - x[itX])

for itX in range(4 + 1, order + 1):
    lNumer = np.convolve(lNumer, [1, -x[itX]])
    lDenom = lDenom*(x[4] - x[itX])

p = p + lNumer/lDenom*fX[4]
```

Rezultat:

```
p =

    0.0000    0.1163   -1.0958    2.4971   -0.6345
```

2. Primetiti šta je promenljivo. Promenljivi indeksi rastu od 0 do reda polinoma + 1. Prethodni par *for* petlji se može ugnjezditi u novu *for* petlju pri čemu promenljivi indeksi zavise od indeksa nove *for* petlje:

```
order = x.size - 1
p = 0
for itFX in range(order + 1):
   lNumer = 1
   lDenom = 1
   for itX in range(itFX):
       lNumer = np.convolve(lNumer, [1, -x[itX]])
       lDenom = lDenom*(x[1] - x[itX])

   for itX in range(itFX + 1, order + 1):
       lNumer = np.convolve(lNumer, [1, -x[itX]])
       lDenom = lDenom*(x[1] – x[itX])

p = p + lNumer / lDenom * fX[itFX]
```

3. Sada je moguće definisati funkciju koja sadrži prethodni algoritam:

```
def lagrange_interpolation(x, fX):
    order = x.size - 1

    .
    .
    .
```

4. Testirati funkciju na primeru:

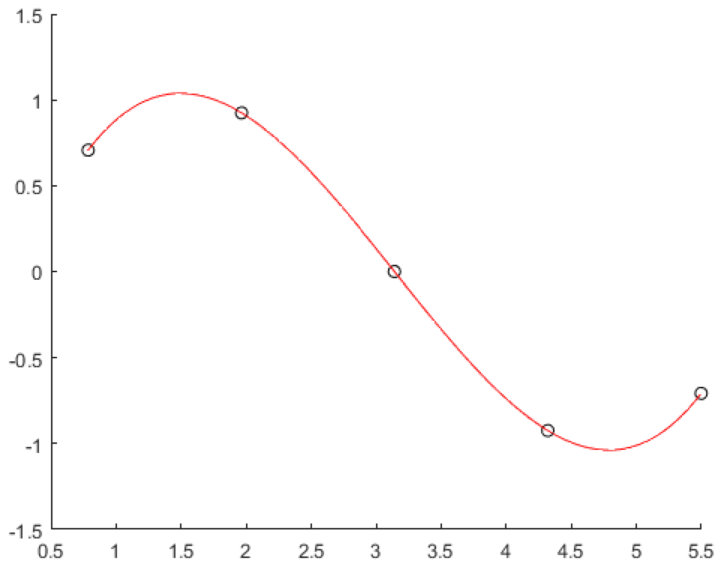
```
x = np.array([0.7854,    1.9635,    3.1416,    4.3197,    5.4978])
fX = np.array([0.7071,    0.9239,    0.0000,   -0.9239,   -0.7071])
plt.scatter(x, fX, c='black')

p = lagrange_interpolation(x, fX)

x = np.linspace(np.min(x), np.max(x), 100)
pX = np.polyval(p, x)
plt.plot(x, pX, 'red')
```

Rezultat:

```
p =
    0.0000    0.1163   -1.0958    2.4971   -0.6345
```



Slika 5. Lagranžova interpolacija

5. Testirati funkciju na primeru:

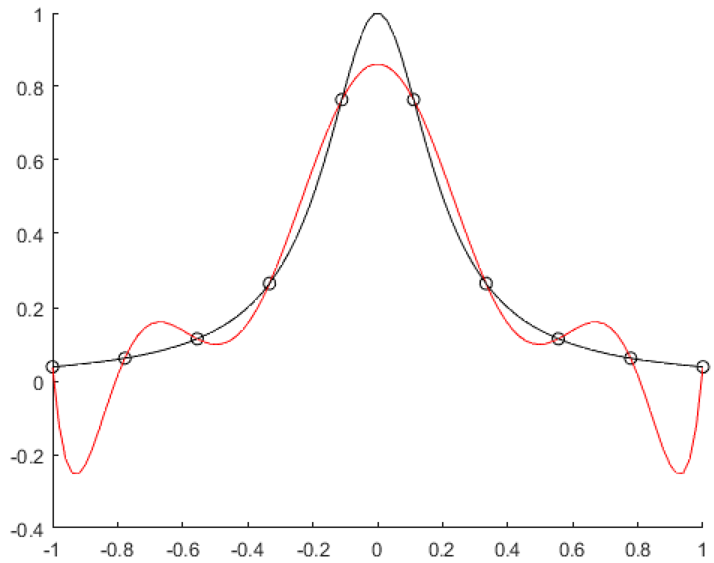
```
x = np.linspace(-1.0, 1.0, 10)
f = lambda x: 1.0 / (1 + 25*x ** 2)
fX = f(x)
plt.scatter(x, fX, c='black')

p = lagrange_interpolation(x, fX)

x = np.linspace(np.min(x), np.max(x), 100)
pX = np.polyval(p, x)
plt.plot(x, pX, 'red', x, f(x), 'black')
```

Rezultat:

```
p =
    0.0000   21.6248   -0.0000  -44.9155   -0.0000   30.7285   -0.0000   -8.2609   -0.0000    0.8615
```



Slika 6. Oscilacije

U zavisnosti od funkcije i od broja tačaka, može da dođe do neželjenih oscilacija. Tada polinom ne aproskimira funkciju dovoljno dobro. Problem se rešava *spline* interpolacijom ili upotrebom regresije.

* **Zadatak 2.** Napisati metodu za traženje Njutnovog polinoma koji zadovoljava proizvoljan broj poznatih tačaka.