Мататематички методи за машинско учење 2023

Домаћи задатак број 1

```python
import numpy as np
import numpy.random as rndm
import matplotlib as mplb
import matplotlib.pyplot as plt
```

Задатак 1. Дат је низ случајно генерисаних бројева $v = [x_1 \quad x_2 \quad \dots \quad x_n)$ (генерише се кодном ћелијом испод). Два елемента x_i и x_j у низу v образују инверзију уколико је $i < j \wedge x_i > x_j$. Написати ефикасан код којим се одређује број инверзија у низу v.

(10 поена)

```python
import time

np.random.seed(0)
n = 12345
min_val, max_val = -10, 10
v = np.random.uniform(min_val, max_val, n)

def broj_inverzija_neoptimalno(v):
    count = 0
    for i in range(n):
        for j in range(i+1, n):
            if v[i] > v[j]:
                count += 1
    return count

start_time = time.time()
count = broj_inverzija_neoptimalno(v)
end_time = time.time()

print(f"Broj inverzija: {count}")
print(f"Vreme izvršenja: {end_time - start_time:.6f} sekundi")
```

```
Broj inverzija: 38287110
Vreme izvršenja: 65.321677 sekundi
```

```python
def broj_inverzija_numpy(v):
    count = sum(np.sum(v[i] > v[i+1:]) for i in range(n - 1))
    return count

start_time = time.time()
count = broj_inverzija_numpy(v)
end_time = time.time()
```

```python
print(f"Broj inverzija: {count}")
print(f"Vreme izvršenja: {end_time - start_time:.6f} sekundi")
```

```
Broj inverzija: 38287110
Vreme izvršenja: 0.657578 sekundi
```

```python
def merge(arr, temp_arr, left, mid, right):
    i, j, k = left, mid, 0
    inv_count = 0

    while i <= mid - 1 and j <= right:
        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            k += 1
            i += 1
        else:
            temp_arr[k] = arr[j]
            inv_count += (mid - i)
            k += 1
            j += 1

    while i <= mid - 1:
        temp_arr[k] = arr[i]
        k += 1
        i += 1

    while j <= right:
        temp_arr[k] = arr[j]
        k += 1
        j += 1

    for i in range(left, right + 1):
        arr[i] = temp_arr[i - left]

    return inv_count

def merge_sort(arr, temp_arr, left, right):
    inv_count = 0
    if left < right:
        mid = (left + right) // 2
        inv_count += merge_sort(arr, temp_arr, left, mid)
        inv_count += merge_sort(arr, temp_arr, mid + 1, right)
        inv_count += merge(arr, temp_arr, left, mid + 1, right)

    return inv_count

def broj_inverzija_merge_sort(v):
    temp_arr = np.zeros(len(v))
    return merge_sort(v, temp_arr, 0, len(v) - 1)
```

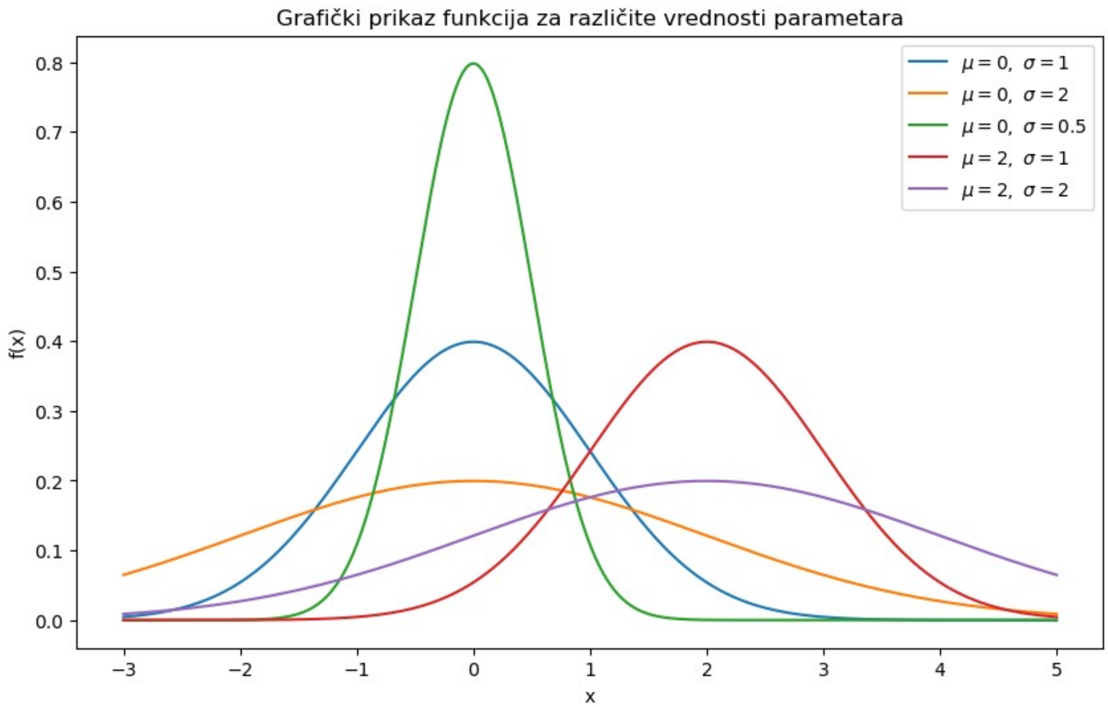
```
start_time = time.time()
count = broj_inverzija_merge_sort(v)
end_time = time.time()
print(f"Broj inverzija: {count}")
print(f"Vreme izvršenja: {end_time - start_time:.6f} sekundi")
```

```
Broj inverzija: 0
Vreme izvršenja: 0.341677 sekundi
```

Задатак 2. На сегменту $[-3,5]$ приказати графике функција $f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$ за комбинације вредности параметара $(\mu,\sigma)\in\{(0,1),(0,2),(0,1/2),(2,1),(2,2)\}$. Добијени график треба да садржи легенду са вредностима параметара за сваку од функција.

(5 поена)

```
def f(x, mu, sigma):
    return 1 / (sigma * np.sqrt(2 * np.pi)) * np.exp(-0.5 * ((x - mu) / sigma)**2)

x = np.linspace(-3, 5, 1000)
parametri = [(0, 1), (0, 2), (0, 1/2), (2, 1), (2, 2)]

plt.figure(figsize=(10, 6))

for mu, sigma in parametri:
    plt.plot(x, f(x, mu, sigma), label=f'$\mu={mu},\ \sigma={sigma}$')

plt.title("Grafički prikaz funkcija za različite vrednosti parametara")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.show()
```



Задатак 3. а) Направити низ kocke од $n \in \{1000, 10000\}$ вредности који симулира резултат n бацања фер коцке за игру. Употербити NumPy функцију `randint` за добијање овог низа.

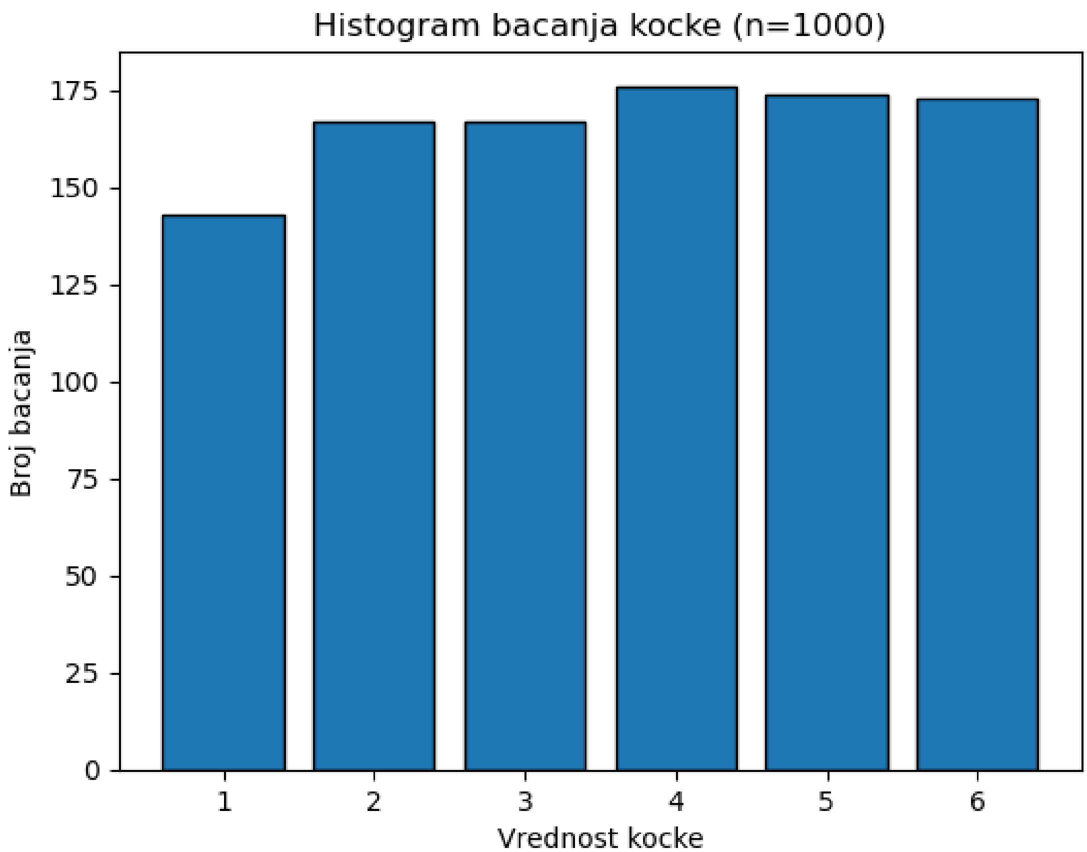
б) Креирати хистограм резултата бацања за сваку од могућих вредности коцке.

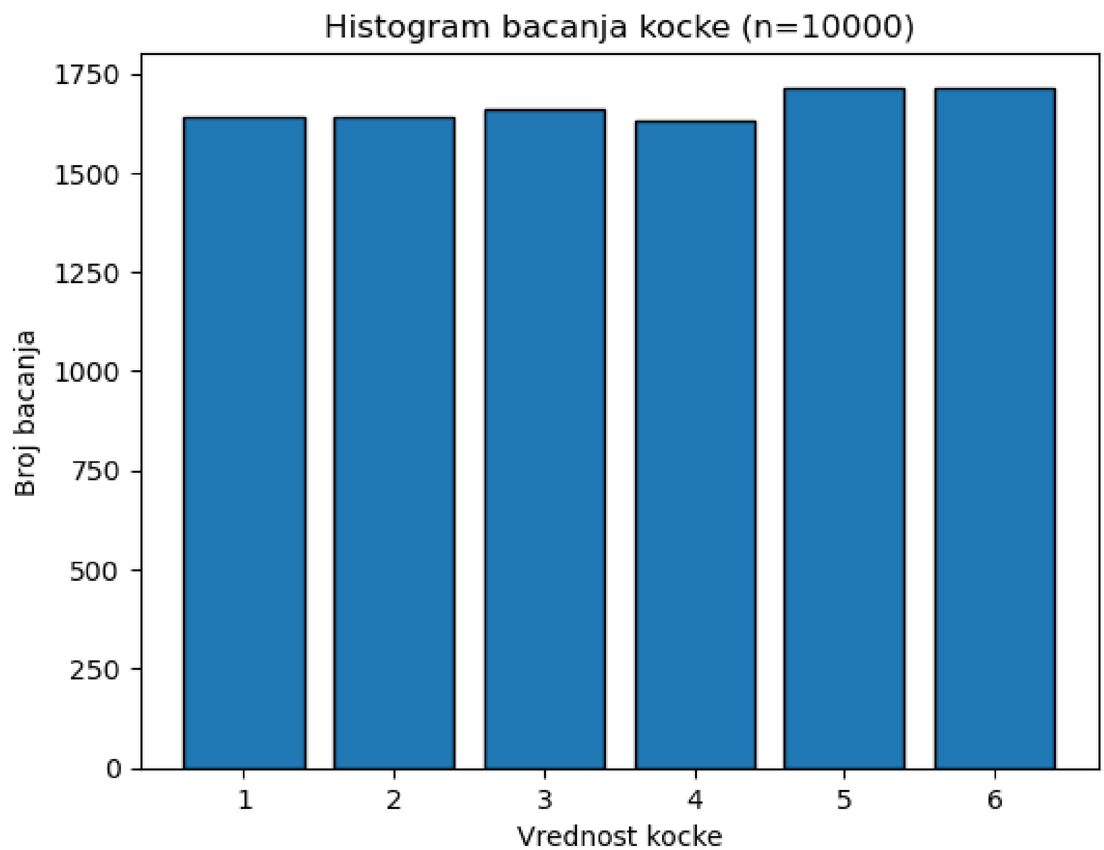
в) Одредити суму свих вредности бачене коцке, као и средњу вредност свих бацања (аритметичку средину).

(5 поена)

```
#a)
n1 = 1000
n2 = 10000

kocke_1000 = np.random.randint(1, 7, n1)
kocke_10000 = np.random.randint(1, 7, n2)

#b)
def prikaz_histograma(kocke, naslov):
    plt.hist(kocke, bins=[1, 2, 3, 4, 5, 6, 7], edgecolor='black',
align='left', rwidth=0.8)
    plt.title(naslov)
    plt.xlabel('Vrednost kocke')
    plt.ylabel('Broj bacanja')
    plt.xticks(range(1, 7))
    plt.show()
```

```
prikaz_histograma(kocke_1000, 'Histogram bacanja kocke (n=1000)')
prikaz_histograma(kocke_10000, 'Histogram bacanja kocke (n=10000)')


#c)
def statistika(kocke):
    suma = np.sum(kocke)
    srednja_vrednost = np.mean(kocke)
    return suma, srednja_vrednost

suma_1000, srednja_vrednost_1000 = statistika(kocke_1000)
suma_10000, srednja_vrednost_10000 = statistika(kocke_10000)

print(f"Za n=1000: suma = {suma_1000}, srednja vrednost =
{srednja_vrednost_1000:.2f}")
print(f"Za n=10000: suma = {suma_10000}, srednja vrednost =
{srednja_vrednost_10000:.2f}")
```






```
Za n=1000: suma = 3590, srednja vrednost = 3.59
Za n=10000: suma = 35283, srednja vrednost = 3.53
```

Задатак 4. На основу скупа података $S=\{(x_k, f(x_k)) \vee k=0,1,\ldots,n, x_i \neq x_j, i \neq j\}$ о функцији $f_4(x)=\dfrac{1-2x^2}{\cos(e^{2x})+2}$ у чворовима
$x_k \in \{-0.66, -0.54, -0.47, -0.33, -0.21, -0.12, 0.03, 0.11, 0.26, 0.37, 0.41, 0.81\}$
написати модификацију Невиловог алгоритма за израчунавање извода интерполационог полинома $P'(0.5)$ на основу шеме \begin{align} p_{ii}'(a)&=0,\ quad p_{ii}(a)=f(x_i)=f_i\
p_{ij}'(a)&=\dfrac{p_{i,j-1}(a)-p_{i+1,j}(a)+(a-x_j)p_{i,j-1}'(a)-(a-x_i)p_{i+1,j}'(a)}{x_i-x_j},\ quad i<j,\ p_{ij}(a)&=\dfrac{(a-x_j)p_{i,j-1}(a)-(a-x_i)p_{i+1,j}(a)}{x_i-x_j},,\quad i<j. \ end{align}

(10 поена)

```python
import numpy as np
import numpy as np

# Čvorovi i vrednosti funkcije f u tim čvorovima
x_cvorovi = np.array([-0.66, -0.54, -0.47, -0.33, -0.21, -0.12, 0.03, 0.11, 0.26, 0.37, 0.41, 0.81])
```

```python
f_x = (1 - 2 * x_cvorovi**2) / (np.cos(np.exp(2 * x_cvorovi)) + 2)

# Funkcija koja računa izvod polinoma interpolacije
def neville_izvod(x_cvorovi, f_x, a):
    n = len(x_cvorovi)
    p = np.zeros((n, n))
    p_izvod = np.zeros((n, n))

    p[:, 0] = f_x

    # Računanje polinoma interpolacije i izvoda
    for j in range(1, n):
        for i in range(n - j):
            p[i, j] = ((a - x_cvorovi[j + i]) * p[i, j - 1] - (a - 
x_cvorovi[i]) * p[i + 1, j - 1]) / (x_cvorovi[i] - x_cvorovi[i + j])
            p_izvod[i, j] = (p[i, j - 1] - p[i + 1, j - 1] + (a - 
x_cvorovi[j + i]) * p_izvod[i, j - 1] - (a - x_cvorovi[i]) * p_izvod[i 
+ 1, j - 1]) / (x_cvorovi[i] - x_cvorovi[i + j])

    # Vrednost izvoda u tački a je poslednja vrednost u poslednjem 
redu matrice izvoda
    izvod_a = p_izvod[0, -1]
    return izvod_a

# Tačka u kojoj se računa izvod
a = 0.5

# Računanje izvoda i ispisivanje rezultata
izvod = neville_izvod(x_cvorovi, f_x, a)
print(f"Izvod polinoma interpolacije u tački {a} je {izvod}")
```

```
Izvod polinoma interpolacije u tački 0.5 je -0.7208417928907618
```

Задатак 5. Претпоставимо да је познат сегмент $[a,b]$ који садржи нулу α глатке функције $f(x)$. Нула је детектована променом знака вредности функције на крајевима интервала, $f(a)f(b)<0$.

Желимо да добијемо ужи сегмент који (највероватније) и даље садржи нулу α функције $f(x)$. То постижемо половљењем почетног сегмента. Наиме α се налази у неком од подсегмената $\left[a,\frac{a+b}{2}\right)$ или $\left[\frac{a+b}{2},b\right)$. У ком је, препознаћемо поново на основу промене знака функције $f(x)$ на једном од тих сегмената, или се α поклапа баш са $\frac{a+b}{2}$. Уколико је нула детектована, тј. $\alpha=\frac{a+b}{2}$ прекидамо потрагу. Ако није сужење интервала можемо да наставимо даљим половљењем кроз итерације. Ово представља метод половљења интервала.

У методу половљења интервала у свакој итерацији се постојећи сегмент $[a_k, b_k]$ дели на два подинтервала. Директна генерализација овог метода јесте дељење постојећег интервала на више делова $p \geq 2$:

$$a_k = c_0 < c_1 < \ldots < c_p = b_k, c_{j+1} - c_j = \frac{b-a}{p}.$$

За нов сегмент $[a_{k+1}, b_{k+1}]$ којим се изолује нула узима се сегмент $[c_j, c_{j+1}]$ на коме се детектује промена знака функције f.

Написати програмски код којим се реализује оваква претрага са вишеструком поделом интервала који користи векторизацију израчунавања над NumPy низовима. Изабрати самостално вредност параметра $p > 2$. Код применити на изоловање нуле функције $f(x) = \frac{x^2}{2} - \cos x$, полазећи од сегмента $[-2, 2]$. Метод вршити док ширина интервала не буде мања од 10^{-6}. Укључити могућност да функција има више нула на изабраном сегменту, тј. да је могуће да више подинтервала детектује промену знака функције.

(20 поена)

```
import numpy as np

def f(x):
    return 0.5 * x ** 2 - np.cos(x)

def find_zeros(a, b, p, tol=1e-6):
    interval_width = b - a
    zeros = []

    while interval_width > tol:
        c_values = np.linspace(a, b, p + 1)
        f_values = f(c_values)
        sign_changes = np.where(np.diff(np.sign(f_values)))[0]

        if len(sign_changes) > 0:
            a = c_values[sign_changes[0]]
            b = c_values[sign_changes[0] + 1]
            interval_width = b - a
            zeros.append((a + b) / 2)
        else:
            break

    return zeros
```

```python
a, b = -2, 2
p = 5  # Broj delova na koje se interval deli u svakoj iteraciji

zeros = find_zeros(a, b, p)
print("Nule funkcije f(x) su:")
for zero in zeros:
    print(f"{zero:.10f}")
```

```
Nule funkcije f(x) su:
-0.8000000000
-0.9600000000
-1.0240000000
-1.0240000000
-1.0214400000
-1.0216960000
-1.0216960000
-1.0216857600
-1.0216898560
-1.0216898560
```