Мататематички методи за машинско учење 2023

Домаћи задатак број 2

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

Задатак 1. Проверити ортогоналност матрице $A = \begin{bmatrix} I_n & O_{n\times m} \\ O_{m\times n} & Q \end{bmatrix}$, где је $Q \in M_{m\times m}$ ортогонална матрица, $I_n \in M_{n\times n}$ јединична матрица и O нула-матрице одговарајућих димензија.

(5 поена)

```
# matrica je ortogonalna ako je njena inverzna matrica jednaka
transponovanoj matrici, proizvod matrice i njene transponovane matrice
mora biti jedinicnna matrica.
def is_orthogonal(matrix):
    product = np.dot(matrix, matrix.T)
    identity = np.identity(matrix.shape[0])
    return np.allclose(product, identity)

# Definisanje dimenzija
n = 3
m = 2

# Kreiranje matrica I_n, O_nxm, O_mxn, i Q
I_n = np.identity(n)
O_nxm = np.zeros((n, m))
O_mxn = np.zeros((m, n))
Q_orthogonal = np.array([[1, 0], [0, -1]])  # Primer ortogonalne
matrice 2x2
Q_not_orthogonal = np.array([[2, 1], [1, 2]])  # Primer matrice koja
nije ortogonalna

# Kreiranje matrice A koristeći blok-matricu sa ortogonalnom matricom
Q
A_orthogonal = np.block([[I_n, O_nxm], [O_mxn, Q_orthogonal]])

# Kreiranje matrice A koristeći blok-matricu sa matricom Q koja nije
ortogonalna
A_not_orthogonal = np.block([[I_n, O_nxm], [O_mxn, Q_not_orthogonal]])

# Provera ortogonalnosti
orthogonal = is_orthogonal(A_orthogonal)
print(f"Matrica A_orthogonal {'je' if orthogonal else 'nije'}
```

```
ortogonalna.")

orthogonal = is_orthogonal(A_not_orthogonal)
print(f"Matrica A_not_orthogonal {'je' if orthogonal else 'nije'} ortogonalna.")
```

```
Matrica A_orthogonal je ortogonalna.
Matrica A_not_orthogonal nije ortogonalna.
```

Задатак 2. Дописати недостајуће елементе матрица A, B и C тако да је матрица A симетрична, B косо-симетрична и C ортогонална.

$$A=\left[\begin{array}{ccc}5 & 3 & -1\\ 0 & ¿2 & 1\end{array}\right),B=\left[\begin{array}{c}1 ¿¿1¿¿\\ ¿\end{array}\right),C=\frac{1}{2}¿$$

(5 поена)

```
# Matrica je simetricna ako je jednaka svojoj transponovanoj matrici
# Matrica je kososimetricna ako je njen negativ jednak njenoj transponovanoj matrici, elementi su simetricni u odnosu na glavnu dijagonalu sa suprotnim znakovima i glavna dijagonala kososimetricne matrice mora biti nula
# Matrica je ortogonalna ako je matrica koja joj je inverzna jednaka transponovanoj matrici, proizvod matrice i njene transponovane matrice mora biti jedinicna matrica

# Inicijalne matrice
A = np.array([[5, 3, -1],
              [np.nan, 0, np.nan],
              [np.nan, 2, 1]])

B = np.array([[np.nan, np.nan, 1],
              [-1, np.nan, np.nan],
              [np.nan, 1, np.nan]])

C = (1/2) * np.array([[1, 1],
                      [1, np.nan]])

# Dopunjavanje simetrične matrice A
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        if np.isnan(A[i, j]):
            A[i, j] = A[j, i]

# Dopunjavanje koso-simetrične matrice B
for i in range(B.shape[0]):
    for j in range(B.shape[1]):
        if np.isnan(B[i, j]):
            if i == j:
                B[i, j] = 0
```

```
            else:
                B[i, j] = -B[j, i]

# Dopunjavanje ortogonalne matrice C
C[1, 1] = (1 - C[0, 0] * C[1, 0]) / C[0, 1]

# Ispis matrica A, B i C
print("A:")
print(A)
print("\nB:")
print(B)
print("\nC:")
print(C)
```

```
A:
[[ 5.  3. -1.]
 [ 3.  0.  2.]
 [-1.  2.  1.]]

B:
[[ 0.  1.  1.]
 [-1.  0. -1.]
 [-1.  1.  0.]]

C:
[[0.5 0.5]
 [0.5 1.5]]
```

Задатак 3. Написати програмски код којим се реализује Штрасенов алгоритам множења две матрице. Нека је $n=2^p$ и $A,B \in M_{n\times n}$. Уколико је $n_{min}=2^d, d \le p$, онда алгоритам израчунава производ $C=AB$ по следећем правилу:

```
    def C=strass(A,B,n,n_min):
        if n<=n_min
            C=AB na klasičan način
        else
            m=n/2
            M1=strass(A[:m,:m]+A[m:,m:],B[:m,:m]+B[m:,m:])
            M2=strass(A[m:,:m]+A[m:,m:],B[:m,:m])
            M3=strass(A[:m,:m],B[:m,m:]-B[m:,m:])
            M4=strass(A[m:,m:],B[m:,:m]-B[:m,:m])
            M5=strass(A[:m,:m]+A[:m,m:],B[m:,m:])
            M6=strass(A[m:,:m]-A[:m,:m],B[:m,:m]+B[:m,m:])
            M7=strass(A[:m,m:]-A[m:,m:],B[m:,:m]+B[m:,m:])
            C[:m,:m]=M1+M4-M5+M7
            C[:m,m:]=M3+M5
```

```
            C[m:,:m]=M2+M4
            C[m:,m:]=M1-M2+M3+M6
```

(10 поена)

```
# Štrassenov algoritam ima za cilj da poboljša vreme izvršavanja
množenja matrica, posebno za veće matrice.
# Algoritam se zasniva na principu "divide and conquer" (podeli i
vladaj), gde se matrice dele na manje delove koji se rešavaju
rekurzivno.
# Dimenzije matrica trebaju biti stepena reda 2
# U teoriji bi ovaj algoritam trebao da bude brzi od klasicnog za
matrice veceg reda.
# Strassenov algoritam za množenje matrica zahteva da matrice imaju
dimenzije koje su stepeni broja 2 (npr. 2, 4, 8, 16, ...) zbog načina
na koji radi.
# Algoritam deli matrice na manje kvadrante i množi ih koristeći
rekurziju. Za rad algoritma, matrice moraju biti kvadratne i imati
dimenzije koje su stepeni broja 2.
import time
def strass(A, B, n, n_min):
    if n <= n_min:
        return A @ B  # Klasično množenje matrica
    else:
        m = n // 2

        # Podela matrica A i B na četvrtine
        A11, A12, A21, A22 = A[:m, :m], A[:m, m:], A[m:, :m], A[m:,
m:]
        B11, B12, B21, B22 = B[:m, :m], B[:m, m:], B[m:, :m], B[m:,
m:]

        M1 = strass(A11 + A22, B11 + B22, m, n_min)
        M2 = strass(A21 + A22, B11, m, n_min)
        M3 = strass(A11, B12 - B22, m, n_min)
        M4 = strass(A22, B21 - B11, m, n_min)
        M5 = strass(A11 + A12, B22, m, n_min)
        M6 = strass(A21 - A11, B11 + B12, m, n_min)
        M7 = strass(A12 - A22, B21 + B22, m, n_min)

        C = np.empty((n, n), dtype=A.dtype)
        C[:m, :m] = M1 + M4 - M5 + M7
        C[:m, m:] = M3 + M5
        C[m:, :m] = M2 + M4
        C[m:, m:] = M1 - M2 + M3 + M6

        return C

"""07Domaci.ipynbdef classic_matrix_mult(A, B):
    n = A.shape[0]
```

```
    C = np.zeros((n, n), dtype=A.dtype)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]
    return C"""
# Klasično množenje matrica
A = np.random.rand(2048, 2048)
B = np.random.rand(2048, 2048)

# Klasično množenje matrica
"""start_time = time.time()
C_classic = classic_matrix_mult(A, B)
classic_time = time.time() - start_time
print(f"Vreme izvršavanja za klasično množenje matrica:
{classic_time:.6f} sekundi\n")"""

# Štrassenov algoritam
n = A.shape[0]
n_min = 64  # Podešavanje granice za prelazak na klasično množenje
matrica

start_time = time.time()
C_strass = strass(A, B, n, n_min)
strassen_time = time.time() - start_time
print(f"Vreme izvršavanja za Štrassenov algoritam: {strassen_time:.6f}
sekundi\n")

# NumPy @ množenje
start_time = time.time()
C_numpy = A @ B
numpy_time = time.time() - start_time
print(f"Vreme izvršavanja za NumPy množenje (@ operator):
{numpy_time:.6f} sekundi\n")

# Poređenje vremena izvršavanja
"""print(f"Razlika u vremenu izvršavanja (Štrassenov algoritam -
klasično množenje): {strassen_time - classic_time:.6f} sekundi")"""
print(f"Razlika u vremenu izvršavanja (Štrassenov algoritam - NumPy
množenje): {strassen_time - numpy_time:.6f} sekundi")
#
```

```
Vreme izvršavanja za Štrassenov algoritam: 5.265299 sekundi

Vreme izvršavanja za NumPy množenje (@ operator): 0.506067 sekundi

Razlika u vremenu izvršavanja (Štrassenov algoritam - NumPy množenje):
4.759233 sekundi
```

Задатак 4. Прилагодити претходни алгоритам тако да може да се користи за множење две квадратне матрице произвољног реда. То значи, уколико су матрице непарне димензије допунити их нула врстом и колоном до парне димензије.

(10 поена)

```
# Potrebno je dopuniti matricu do najblizeg broja stepena dvojke
import numpy as np
import time

def dopuni_matricu(A):
    n = A.shape[0]
    dopunjena_velicina = 2 ** int(np.ceil(np.log2(n)))
    if n == dopunjena_velicina:
        return A
    else:
        dopunjena_A = np.zeros((dopunjena_velicina,
dopunjena_velicina), dtype=A.dtype)
        dopunjena_A[:n, :n] = A
        return dopunjena_A

def strassen(A, B, n, n_min):
    if n <= n_min:
        return A @ B
    else:
        m = n // 2

        A11, A12, A21, A22 = A[:m, :m], A[:m, m:], A[m:, :m], A[m:,
m:]
        B11, B12, B21, B22 = B[:m, :m], B[:m, m:], B[m:, :m], B[m:,
m:]

        M1 = strassen(A11 + A22, B11 + B22, m, n_min)
        M2 = strassen(A21 + A22, B11, m, n_min)
        M3 = strassen(A11, B12 - B22, m, n_min)
        M4 = strassen(A22, B21 - B11, m, n_min)
        M5 = strassen(A11 + A12, B22, m, n_min)
        M6 = strassen(A21 - A11, B11 + B12, m, n_min)
        M7 = strassen(A12 - A22, B21 + B22, m, n_min)

        C = np.empty((n, n), dtype=A.dtype)
        C[:m, :m] = M1 + M4 - M5 + M7
        C[:m, m:] = M3 + M5
        C[m:, :m] = M2 + M4
        C[m:, m:] = M1 - M2 + M3 + M6

        return C

# Generiši slučajne matrice
```

```python
A = np.random.rand(2049, 2049)
B = np.random.rand(2049, 2049)

# Dopuni matrice nulama
dopunjena_A = dopuni_matricu(A)
dopunjena_B = dopuni_matricu(B)

# Strassenov algoritam
n = dopunjena_A.shape[0]
n_min = 64

start_time = time.time()
dopunjena_C_strass = strassen(dopunjena_A, dopunjena_B, n, n_min)
strassen_vreme = time.time() - start_time
print(f"Vreme izvršavanja za Strassenov algoritam:
{strassen_vreme:.6f} sekundi\n")

# Ukloni dopunu
C_strass = dopunjena_C_strass[:A.shape[0], :A.shape[1]]

# NumPy @ množenje
start_time = time.time()
C_numpy = A @ B
numpy_vreme = time.time() - start_time
print(f"Vreme izvršavanja za NumPy množenje (@ operator):
{numpy_vreme:.6f} sekundi\n")

# Uporedi vremena izvršavanja
print(f"Razlika u vremenu izvršavanja (Štrassenov algoritam - NumPy
množenje): {strassen_vreme - numpy_vreme:.6f} sekundi")
```

```
Vreme izvršavanja za Strassenov algoritam: 29.892966 sekundi

Vreme izvršavanja za NumPy množenje (@ operator): 0.901012 sekundi

Razlika u vremenu izvršavanja (Štrassenov algoritam - NumPy množenje):
4.759233 sekundi
```

Задатак 5. За примере 2.7.3 и 2.7.4 из књиге, на конкретним примерима матрица и вектора малих димезија проверити формуле о сечењима одговарајућих тензорских производа.

(10 поена)