**תרגיל בית 1 – מודלים לא לינאריים בחקר ביצועים**

**מגישים :**

**אלעד בוכריס – 206202426**

**משה דידי – 311395834**

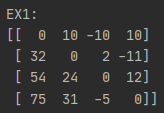
**פונקציית בדיקת הקלט עבור שאלות 1 ו2:**

def is\_valid\_input(A=None, B=None, x=None):  
 valid, error\_message = True, ''  
 n, m = 0, 0  
 if A is not None:  
 try:  
 # Try to create a numpy array\_like from given A lists  
 A = np.array(A)  
 # Validate that all matrix elements are numbers  
 \_type = np.array(list(A)).dtype  
 if \_type not in ['int32', 'float64']:  
 valid = False  
 error\_message += ' Matrix A is not all numbers,'  
 else:  
 # Validate that the matrix is squared and has at least one element  
 n\_temp, m\_temp = A.shape  
 if n\_temp != m\_temp or n\_temp <= 0:  
 valid = False  
 error\_message += ' Matrix A dimensions are not valid,'  
 else:  
 n, m = n\_temp, m\_temp  
 except Exception as err:  
 valid = False  
 error\_message += f' {str(err)},'  
 if B is not None:  
 try:  
 # Try to create a numpy array\_like from given B lists if B is given  
 B = np.array(B)  
 # Validate that all matrix elements are numbers  
 \_type = np.array(list(B)).dtype  
 if \_type not in ['int32', 'float64']:  
 valid = False  
 error\_message += ' Matrix B is not all numbers,'  
 else:  
 # Validate that the matrix is squared and has at least one element and it's dimensions are the same of A  
 n\_temp, m\_temp = B.shape  
 if n\_temp != m\_temp or n\_temp <= 0 or n\_temp != n or m\_temp != m:  
 valid = False  
 error\_message += ' Matrix B dimensions are not valid,'  
 except Exception as err:  
 valid = False  
 error\_message += f' {str(err)},'  
 if x is not None:  
 try:  
 # Try to create a numpy array from given x  
 x = np.array(x)  
 # Validate that all array elements are numbers  
 \_type = np.array(list(x)).dtype  
 if \_type not in ['int32', 'float64']:  
 valid = False  
 error\_message += ' Array x is not all numbers,'  
 else:  
 # Validate that the array dimensions are the appropriate to A  
 n\_temp = x.shape[0]  
 if n\_temp != n or n\_temp <= 0 or len(x.shape) != 1:  
 valid = False  
 error\_message += ' Array x dimensions are not valid,'  
 except Exception as err:  
 valid = False  
 error\_message += f' {str(err)},'  
 if error\_message != '':  
 print(error\_message)  
 return valid

שאלה 1 :

def ex1(A, x):  
 if not is\_valid\_input(A=A, x=x):  
 return  
 n = A.shape[0]  
 X = np.array([x, ] \* n).T  
 i = np.arange(1, n + 1)  
 B = np.add(A, np.multiply(X, i)).T  
 np.fill\_diagonal(B, 0)  
 return B

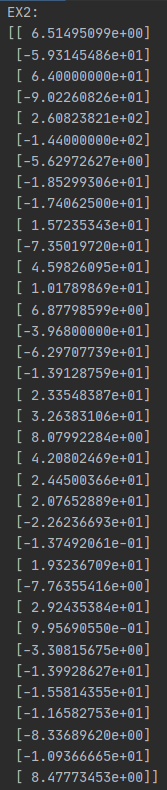
פלט:



שאלה 2 :

def ex2(A, B, n, b):  
 if not is\_valid\_input(A=A, B=B, x=b):  
 return  
 """ Create P """  
 BABT = np.block([[B, A, B.T]])  
 m = A.shape[0]  
 P = np.block([[A, B.T, np.array([np.zeros(m), ] \* (n - 2) \* m).T]])  
 second = np.block([[BABT, np.array([np.zeros(m), ] \* (n - 3) \* m).T]])  
 P = np.append(P, second, axis=0)  
 for i in range(1, n - 3):  
 temp = np.block([[np.array([np.zeros(m), ] \* i \* m).T, BABT, np.array([np.zeros(m), ] \* (n - i - 3) \* m).T]])  
 P = np.append(P, temp, axis=0)  
 before\_last = np.block([[np.array([np.zeros(m), ] \* (n - 3) \* m).T, BABT]])  
 P = np.append(P, before\_last, axis=0)  
 last = np.block([[np.array([np.zeros(m), ] \* (n - 2) \* m).T, B, A]])  
 P = np.append(P, last, axis=0)  
 """ Create y """  
 y = b.T  
 for i in range(2, n + 1):  
 temp = b \* i  
 y = np.append(y, temp, axis=0)  
 """ Create Q """  
 Q = np.kron(A, P)  
 """ Create z """  
 z = np.block([[y] \* m])  
 return np.linalg.solve(Q, z.T)

פלט:



שאלה 3 :

1. טענה 1:

הוכחה :

נסמן

*מש"ל*

*טענה 2 :*

*נסמן :*

*מש"ל*

*טענה 3 :*

*נסמן :*

*מש"ל*

*טענה 4 :*

*הוכחה :*

*נסמן*

*ולכן :*

*מש"ל*

*טענה 5 :*

*בהינתן אזי :*

*נסמן :*

*לכן, ניתן לראות כי לכל הפחות הערך המקסימלי תחת קבוצה D יתקבל תחת ה-x הנותן את הערך המקסימלי תחת C (מכיוון שאנו ממקסמים).*

*בנוסף, ייתכן ש-*

*ולכן אי השוויון מתקיים.*

*באופן דומה עבור המינימום.*

1. *תהי*

*יש להוכיח כי אם*

*כך ש-*

*אזי :*

*לפי הנתון מתקיים כי*

*מש"ל.*

*הטענה אינה נכונה ללא ההנחה.*

*דוגמה נגדית :*

*ניתן לראות כי ההנחה לא מתקיימת מכיוון ש-*

*בנוסף, מתקיים כי :*

*לכן, הטענה אינה נכונה.*

*שאלה 4 :*

*יש לפתור את הבעיה הבאה:*

*עבור מטריצה סימטרית.*

*ננסה למצוא חסם עליון :*

*מכיוון שהמטריצה סימטרית ממשית אזי קיימת מטריצה כך ש-*

*כאשר ע"ע ממשי של A.*

*ולכן :*

*נפתח המכפלה :*

*נשים לב כי :*

*וכי:*

*ולכן מתקיים כי*

*כעת, נמצא ווקטור כך שהחסם העליון מתקבל.*

*נבחר את x להיות הו"ע העצמי המתאים לערך העצמי המקסימלי.*

*נסמנו ב- אזי :*

*ניתן לראות כי החסם העליון מתקבל וזהו הערך האופטימלי לבעיה.*

1. *המטריצה U היא אורתונורמלית, לכן, .*
2. *מאותה סיבה, היא אורתונורמלית ולכן המכפלה הסקלרית של כל שתי עמודות שונות ב-U היא אפס.*

*שאלה 5 :*

*נתונות הניתנות לקירוב על ידי :*

1. *נניח כי על מנת להתגבר על היתירות. ולכן :*

*יש למצוא מטריצה A ווקטור b כך שבעיית הריבועים הפחותים שלנו תהיה :*

*ניתן לראות כי*

*ולכן :*

*נשים לב כי :*

*אנו רוצים למזער ביטוי זה לכל i, לכן נמזער את הסכום :*

*נשים לב כי :*

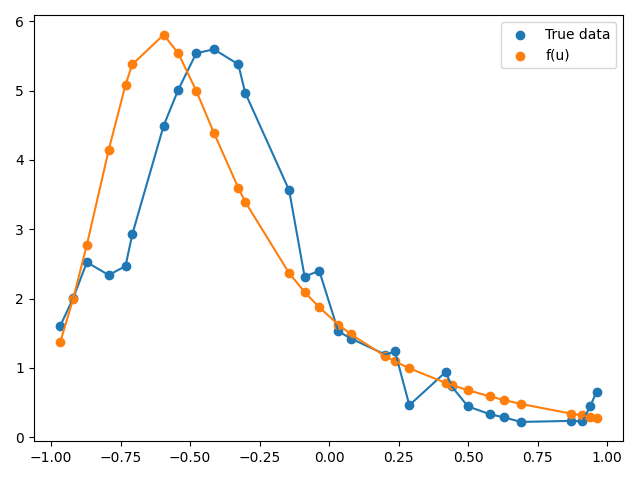
*לכן: הבעיה היא :*

# ex\_5\_b  
def fit\_rational(X):  
 m = X.shape[0]  
 A, y = create\_A\_and\_y(X, m)  
 x = np.linalg.lstsq(A, y, rcond=None)  
 return x

def create\_A\_and\_y(X, m):  
 y = X[:, 1]  
 A = np.zeros((m, 5))  
 A[:, 0] = np.ones(m)  
 A[:, 1] = X[:, 0]  
 A[:, 2] = np.square(X[:, 0])  
 A[:, 3] = -np.multiply(X[:, 0], y)  
 A[:, 4] = -np.multiply(np.square(X[:, 0]), y)  
 return A, y

*פלט:*

**

**

1. *כעת, נניח כי*

*ניתן לייצג את הבעיה כעת בצורה הבאה :*

*נכתוב באופן דומה למה שקיבלנו בסעיף הקודם*

*נשים לב כי*

*כלומר, אנו רוצים לפתור את הבעיה הבאה:*

*נשים לב כי תמיד מטריצה סימטרית ולכן מקבלים בעיה דומה לשאלה 4.*

*בשאלה 4 ראינו כי הערך המקסימלי של הבעיה הוא הע"ע המקסימלי, באופן דומה, הערך המינימלי של הבעיה הוא הע"ע המינימלי.*

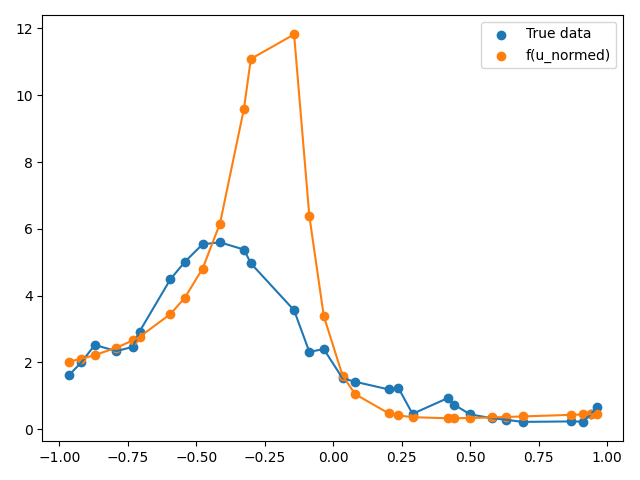
*ולכן u צריך להיות הו"ע המתאים לערך העצמי המינימלי.*

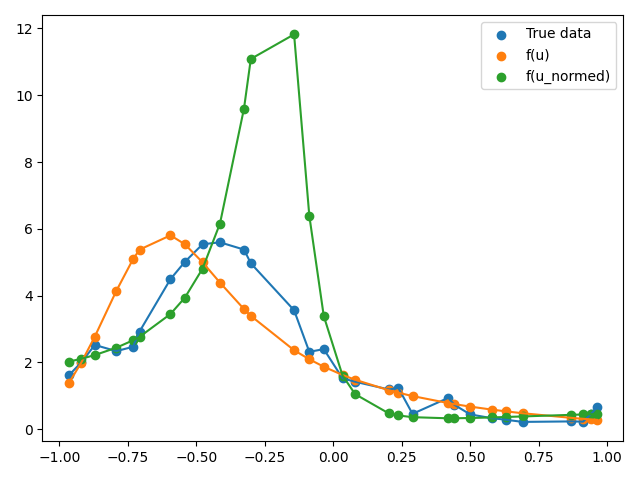
# ex\_5\_d  
def fit\_rational\_normed(X):  
 m = X.shape[0]  
 ATA = create\_A\_TA(X, m)  
 i = np.argmin(np.linalg.eig(ATA)[0])  
 x = np.linalg.eig(ATA)[1][:, i]  
 return x

def create\_A\_TA(X, m):  
 y = X[:, 1]  
 A = np.zeros((m, 6))  
 A[:, 0] = np.ones(m)  
 A[:, 1] = X[:, 0]  
 A[:, 2] = np.square(X[:, 0])  
 A[:, 3] = -y  
 A[:, 4] = -np.multiply(X[:, 0], y)  
 A[:, 5] = -np.multiply(np.square(X[:, 0]), y)  
 ATA = np.dot(A.T, A)  
 return ATA

*פלט:*

**

**

*בגרף הבא מוצגים הנתונים, הקירוב לפי u והקירוב לפי u\_norm:*

*בנוסף נחשב את נורמת המרחק של הנתונים והעקומות הקירוב:*

**

*לכן ניתן לראות כי המרחק בין נתונים והקירוב לפי סעיף א קטן מהמרחק לפי סעיף ג.*

*גם עפ"י הגרף ניתן לראות כי המגמה תואמת יותר לנתונים לפי הקירוב מסעיף א לכן נסיק כי הגישה לפי סעיף א' נותנת קירוב טוב יותר.*