LAB1

שם : אביחי מסלאווי ת.ז : 308365485

הקדמה: במעבדה זו החלטתי להשתמש בשפת פייתון בכדי לממש את המתבקש ממני בכל סעיפי המעבדה.

כמו כן השתמשתי במודולים הבאים השאר מוצגים בתמונה בעיגול :

```
import numpy as np
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
```

אם ברצונך לבצע הרצות בסביבת הפייתון שלך יש לדאוג להתקין את המודולים האלה לפני השימוש בסקריפטים שיוספו בהגשה כמוכן בכול סעיף מתאים אני יראה את מימוש הקוד המתאים עם הסבר.

סעיף 1 :

בסעיף זה התבקשנו ליצור 11 סטי אימון ל11 אלפות שונות ל80 ערכי X שונים שמתפלגים אחידה בין 4- ל-4

תחילה נממש את הפונקציה המבוקשת בסעיף ללא רעש :

```
def f_of_x_withOut_nocice(alpha,x):
    C = 1.5
    return C*np.exp(alpha*x)
```

שנית נממש פונקציה שמחזירה לי את האפסילון שמתפלג נורמלי בסטיית תקן כלשהי מאוחר יותר נגדיר לו ערך 1 ניתן לראות זאת בקוד:

```
def getTheNewEpslone(sigma):
    return np.random.normal(0,sigma)
```

כעת נממש את הפונקציה הכוללת כלומר הפונקציה בלי הרעש עם פונקציה שמחזירה לי את הרעש , זו הפונקציה שנשתמש בה לקבל את ערכי הY בסט האימון ובסט המבחן :

```
def y_with_nocie(alpha,x,sigma):
    return f_of_x_withOut_nocice(alpha,x) + getTheNewEpslone(sigma)
```

למען הנוחות ניצור פונקציה שיוצר שמחזירה תוצאות Y בינתן קובצת X ואלפא כלומר יצירת סט אימון כמוכן ניתן לראות הפונקציה מקבלת פרמטרים אחרים אבל זאת רק לנוחות טכנית במימוש הבעיה בקוד:

```
def makeOneYs(x_set,y_func_with_nocice,alpha,sigma):
    y = []
    for x in x_set:
        y.append(y_func_with_nocice(alpha,x,sigma))

    return y
```

לאחר זאת נכין מבנה נתונים שיכיל את כל תוצאות הY לכל אלפא ונמלא אותו .

ואז נקבל את קבוצת הX שהיא אותו דבר כל פעם מבנה שמכיל 11 קבוצות לY לכל אלפא מתאים כך היה סט האימון שלנו כמו כן נבצע אותו דבר לסט המבחן :

הנה מימוש בקוד :

יצירה של קובצות X ואלפות לפי הדרישה :

```
Xs_training = sorted(np.random.uniform(-4, 4, N).reshape(-1, 1))
Xs_test = sorted(np.random.uniform(-4,4,N).reshape(-1,1))

alphas = np.linspace(0.1,2.1,11)
```

מילוי ויצירת המבנה נתונים הכללי :

```
for a in alphas:
    all_Alphas_Ys_Training_to_Xs_trainingSet.append(makeOneYs(Xs_training, y_func_with_nocice=y_with_nocie, alpha=a, sigma=task_sigma))
    all_Alphas_Ys_test_to_Xs_testSet.append(makeOneYs(Xs_test, y_func_with_nocice=y_with_nocie, alpha=a, sigma=task_sigma))
```

הקבועים שהגדרתי לטובת הנוחות במימוש הקוד :

```
task_sigma = 1
N = 80
C = 1.5
```

להדגמה הנה חלק מפלט קוד להתרשמות שהכול לרגע זה עובד השתמשתי להדגמה זו במודול בשם pandas שגם אותו תצטרך להדפסה יפה של הפלט בסקריפט שהגיש כתבתי את המידע הכולל לתוך 11 קבצי טקסט בהתאמה לכל אלפא גם שוב לטובת ההתרשמות שהכול עובד בסעיף זה :

```
a =  0.30000000000000004
                       X                       Y
0   [-3.9795629379381525]   [-0.5050126417909082]
1   [-3.8149457685657238]    [2.0417746982564733]
2    [-3.718563696580432]    [0.8970267852413243]
3    [-3.545039704578821]     [0.515507606772627]
4   [-3.4792466776391553]  [-0.17483671096774533]
..                    ...                     ...
75   [3.7398930154258236]     [4.717793835176142]
76   [3.7564522924484507]    [1.8385722163472726]
77   [3.8064701105570986]    [4.2232229381791235]
78   [3.8065021186649073]      [4.90105794411434]
79    [3.908221648390308]     [4.502743754805308]

[80 rows x 2 columns]
```

```
a =  0.1
                       X                       Y
0   [-3.9795629379381525]   [2.0741409940458677]
1   [-3.8149457685657238]   [0.6521709446825354]
2    [-3.718563696580432]   [2.0034271205593943]
3    [-3.545039704578821]   [1.8525124910668231]
4   [-3.4792466776391553]   [0.4528003676443706]
..                    ...                     ...
75   [3.7398930154258236]   [3.7524118684898164]
76   [3.7564522924484507]      [3.1804811279674]
77   [3.8064701105570986]   [3.3188886444267576]
78   [3.8065021186649073]    [2.779897189084083]
79    [3.908221648390308]   [0.8365189882135609]

[80 rows x 2 columns]
```

והנה חלק הקוד שמייצר את הפלט הזה ומודול שתצטרך להתקין בנוסף לכל מה שהתקנת לטובת הרצת הקוד בסקריפט LAB1_S1:

```
for i in range(len(alphas)):
    print("a = ",alphas[i])
    print(pd.DataFrame({"X": Xs_training , "Y" : all_Alphas_Ys_Training_to_Xs_trainingSet[i]}))
```

```
import pandas as pd
```

סעיף 2:

בסעיף זה ממשתי את 2 תת הסעיפים א ו ב בסקריפט 1 שנקרא task1

בסעיף זה השתמשתי בכמה פונקציות עזר :

מימוש של סכימה ה-LOST FUNCTION כלומר פונקציה שמקבל שני קבוצות Y ומחשבת את סכום הפרש הריבועים בין שני הקבוצות .

השתמש בפונקציה זו לחישוב ה TSE וTE

הפונקציה הנ"ל :

```
def calcLostFunc(y1,y2):
    res = []
    if len(y1) == len(y2):
        for i in range(len(y1)):
            res.append((y1[i]-y2[i])*(y1[i]-y2[i]))

    return sum(res)
```

כעת הקוד שעושה את רוב העבודה בסעיף 2 עם שורות בצד שהתייחס לכ שורת קוד בהסבר :

```
lr = LinearRegression()
tableOfResults_q2_A_and_B = []
for i in range(len(alphas)):
    writeSetsTofile(Xs_training,all_Alphas_Ys_Training_to_Xs_trainingSet[i],"dataSetfor" + str(i) + ".txt")
    lr.fit(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[i])
    Ys_preds_training = lr.predict(Xs_training)
    all_predicted_Ys_from_Xs_trainingSet.append(Ys_preds_training)
    Ys_preds_test = lr.predict(Xs_test)
    all_predicted_Ys_from_Xs_testSet.append(Ys_preds_test)
    tse = calcLostFunc(all_Alphas_Ys_Training_to_Xs_trainingSet[i],Ys_preds_training) / N
    te = calcLostFunc(all_Alphas_Ys_test_to_Xs_testSet[i],Ys_preds_test) / N
    tableOfResults_q2_A_and_B.append({"alpha": alphas[i],"teta_0_cova": lr.intercept_[0], "teta_1_cova": list(lr.coef_.tolist())[0], "TSE": tse[0],"TE": te[0]})


file = open("res_A_B.txt", "w")
for row in tableOfResults_q2_A_and_B:
    row_string = row.__str__()+"\n"
    file.write(row_string)

file.close()
print("DONE")
```

הסברים :

בשורה 76 : יצירת אובייקט (lr) שיחשב לי את מודל פונקציית הניבוי לפי שיטת הריבועים הפחותים שנלמדה בכיתה .

בשורה 77 : יצירת רשימה שתתפקד כטבלה לשמירת התוצאות שמתבקשות בתתי סעיפים א ו ב לכל אלפא .

שורה 78 : תחילת הלולאה

שורה 79 : שימוש בפונקציה שכותבת סט אימון לקובץ אלו היו אותם קבצים שהגיש לטובת ההדגמה וההתרשמות .

שורה 80 : שימוש ב lr לחישוב מודל פונקציית הניבוי לסט אימון לאלפא מתאים. שורה זו שוות ערך לשורה הזו בקובץ המטלב שהיה לנו באתר המודל

```
LSQ_estimated_line = fitlm(x_train,y_train); % use of the MATLAB command fitlm, which fits
 %... a linear model by the least squares method. See MATLAB help for
 %further details
```

שורה 81 – 82 : יצירת קבוצת Y שהם תוצאות ניבוי לפי פונקציית הניבוי שחישב אובייקט lr ושמירת הקבוצה במבנה לשימוש מאוחר יותר בקוד ובעבודה הזו . שורה זו שוות ערך לשורה הנ"ל בקובץ מטלב שהיה לנו באתר המודל .

```
ypred = predict(LSQ_estimated_line,x); % this is to compute the predicted values, at any point
```

שורה 83 – 84 : אותו דבר כמו בשורות 81 ו 82 רק לטובת סטי המבחן .

שורה 85 86 87 : חישוב שגיאת האימון הממוצעת חישוב שגיאת המבחן הממוצעת ושמירת במבנה שיצרתי בשורה 77

בהמשך הקוד משורה 90 ומטה אני שומר את הנתונים של המבנה בקובץ

הנה הקובץ של אחת הרצות להתרשמות שהכול עובד עד לשלב זה כמו כן גם קובץ זה יוגש .

```
{'alpha': 0.1, 'teta_0_cova': 1.6121257653194154, 'teta_1_cova': [0.164773968922511 06], 'TSE': 1.0191408497468815, 'TE': 1.1785439768001074}

{'alpha': 0.30000000000000004, 'teta_0_cova': 2.0550660334652155, 'teta_1_cova': [0.5031760595960246], 'TSE': 1.0106682592081637, 'TE': 1.219944963315053}

{'alpha': 0.5, 'teta_0_cova': 2.657717517966695, 'teta_1_cova': [0.8430488745186681], 'TSE': 2.607423346810502, 'TE': 3.5343161106223775}

{'alpha': 0.7000000000000001, 'teta_0_cova': 3.953489702942063, 'teta_1_cova': [1.6908796897242675], 'TSE': 8.356776797994264, 'TE': 15.184788481725477}

{'alpha': 0.9, 'teta_0_cova': 6.674860248781445, 'teta_1_cova': [3.2383116070312 85], 'TSE': 37.763817686547256, 'TE': 89.33104743247051}

{'alpha': 1.1, 'teta_0_cova': 11.747062004002073, 'teta_1_cova': [5.990505379073318], 'TSE': 174.04403714421647, 'TE': 453.3814150321929}

{'alpha': 1.3000000000000003, 'teta_0_cova': 21.03402353986914, 'teta_1_cova': [11.297295127029507], 'TSE': 763.0522507806313, 'TE': 2341.484170909504}

{'alpha': 1.5000000000000002, 'teta_0_cova': 38.83462660551686, 'teta_1_cova': [21.42061522017108], 'TSE': 3331.99204028855, 'TE': 11796.862135322863}

{'alpha': 1.7000000000000002, 'teta_0_cova': 72.91853272191206, 'teta_1_cova': [40.993707069316 75], 'TSE': 14125.348765202789, 'TE': 58085.37288570854}

{'alpha': 1.9000000000000001, 'teta_0_cova': 138.9866277882481, 'teta_1_cova': [79.40880041556451], 'TSE': 60006.217090899685, 'TE': 283515.738418545}

{'alpha': 2.1, 'teta_0_cova': 268.554203380907, 'teta_1_cova': [154.83452490277128], 'TSE': 253908.1513987495, 'TE': 1375901.513193756}
```

תת סעיף ג :

ההבדל הקיים בין האלפות הנמוכות לגבוהות זה השגיאה הממוצעת גם לסטי המבחן וגם לסטי האימון ככל שאלפא גדלה כך גם השגיאה אותו דבר גם בתטאות אבל זאת כדי לנסות להדמות לפונקציה המערכית כמה שיותר .

בנוסף לכך אנו צריכים להבין שאנו מנסים לנבא תוצאה לפונקציה מערכית בעזרת פונקציית ניבוי לינארית זאת אומרת שפונקציית הניבוי שהשיטה פולטת תהיה יחסית שווה כלומר שבאופן יחסי פונקציית הניבוי תהיה רחוקה מהפונקציה המערכית המתאימה בכל אלפא .

בנוסף לכך ככל שהאלפא בפונקציה המערכית גדלה הרעש מאבד יחסית את המשמעות שלו .

להסבר והבנה מעמיקה ותר לתשובה שלי הנה גרפים להמחשת לתת סעיף זה

הסדר בכל תצלום הולך משמאל לימין כל גרף לפי האלפא המתאימה

האלפות נמצאות גם בכותרת כל גרף והאלפא שלו בהתאמה.

בכחול הפונקציה המערכית המקורית ללא רעש

נקודות אדומות סט האימון ובכתום פונקציית כלל הניבוי .

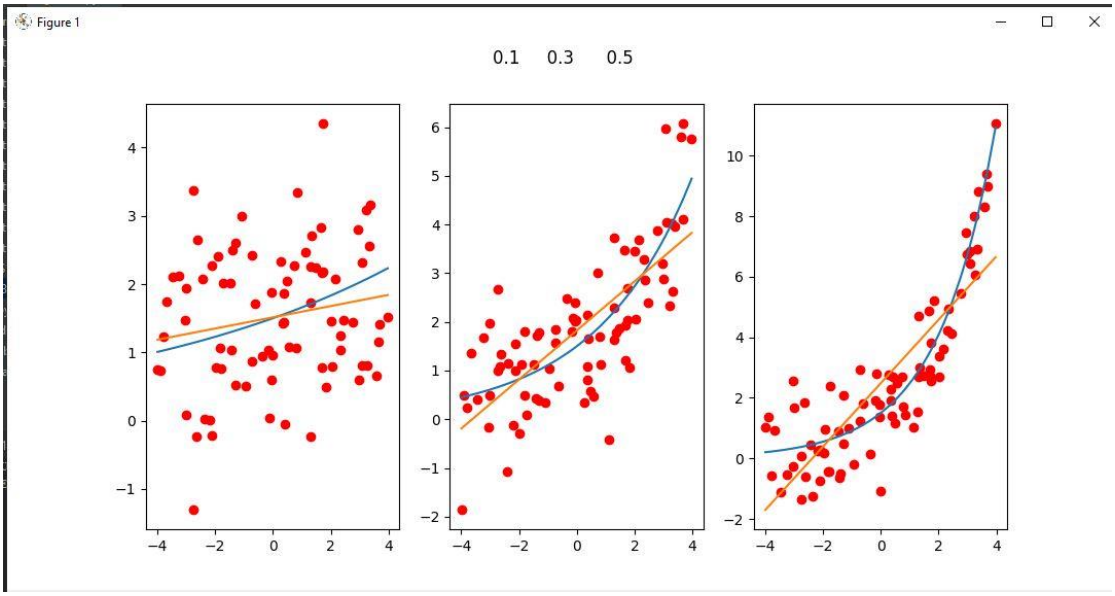


Figure 1
0.1 0.3 0.5

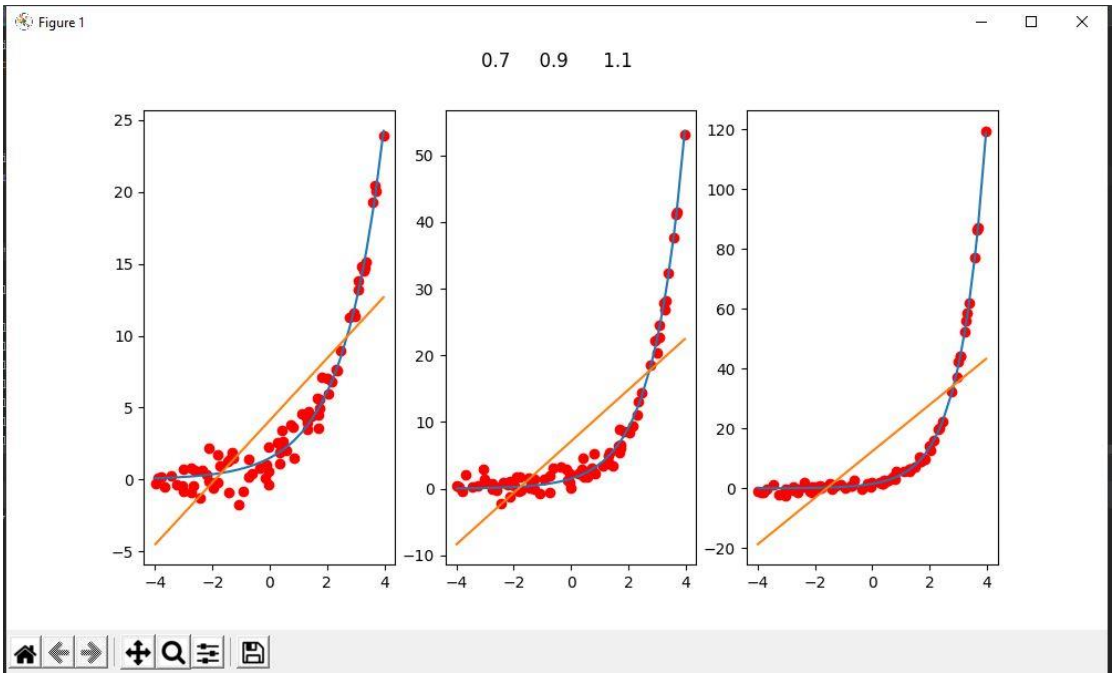
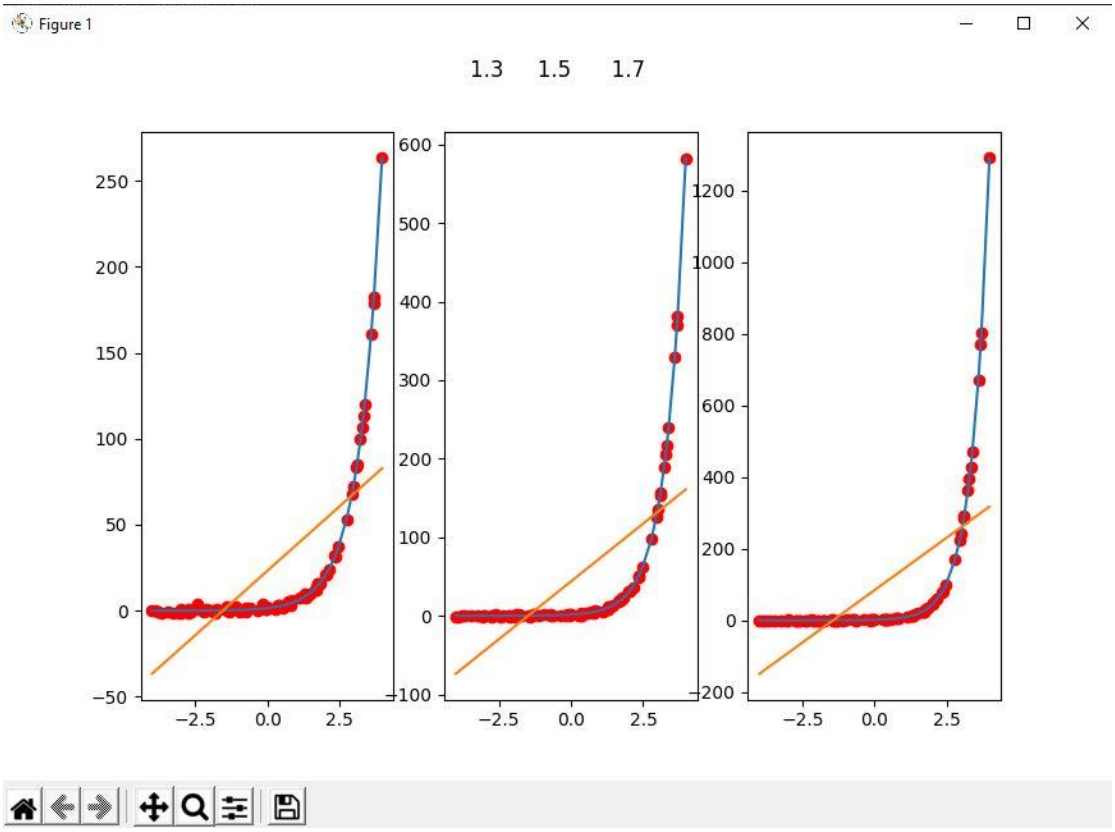
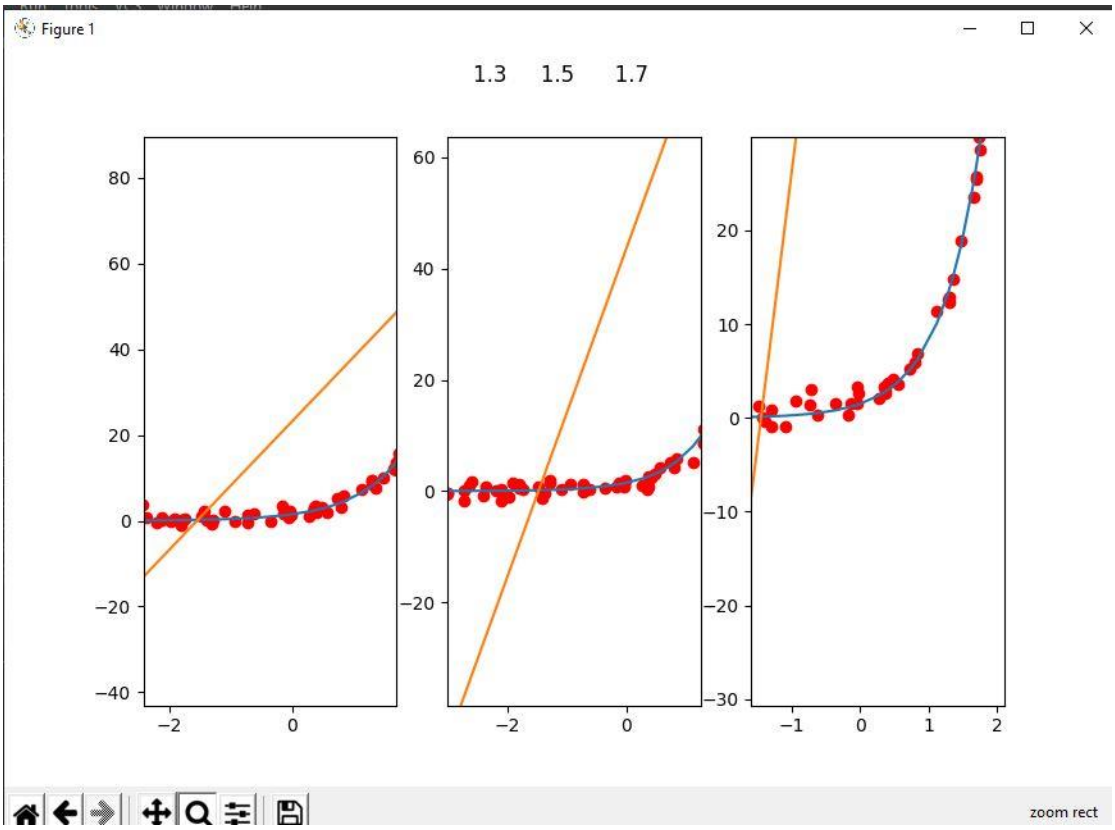
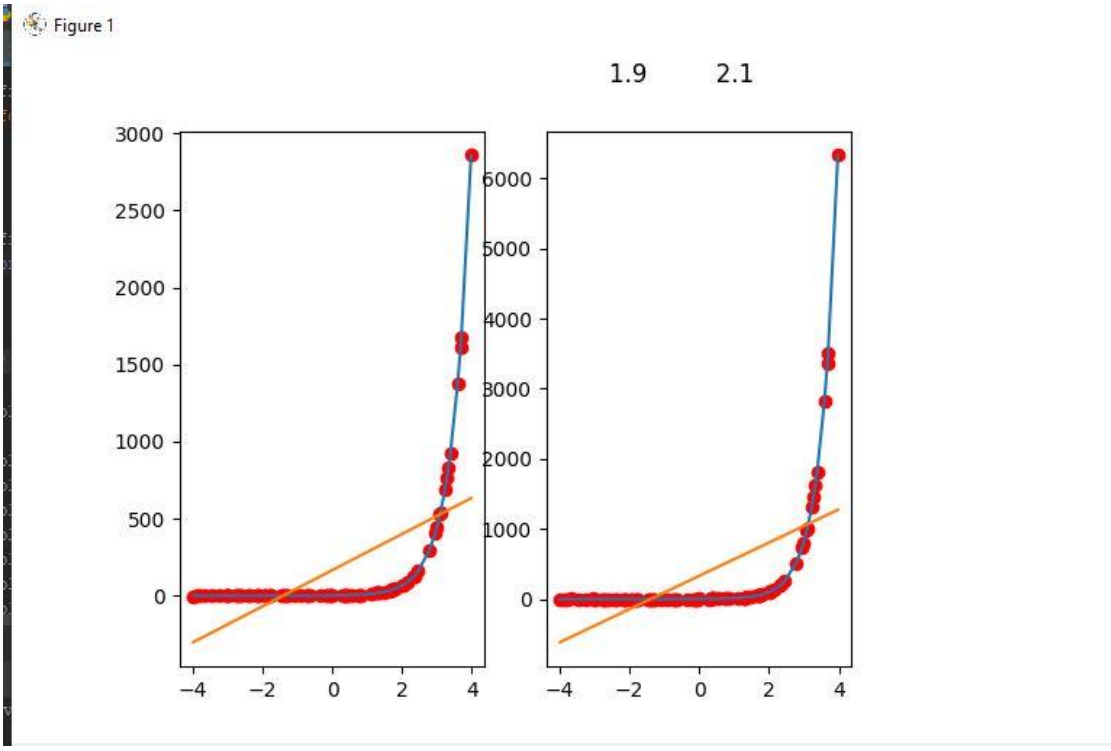


Figure 1
0.7 0.9 1.1

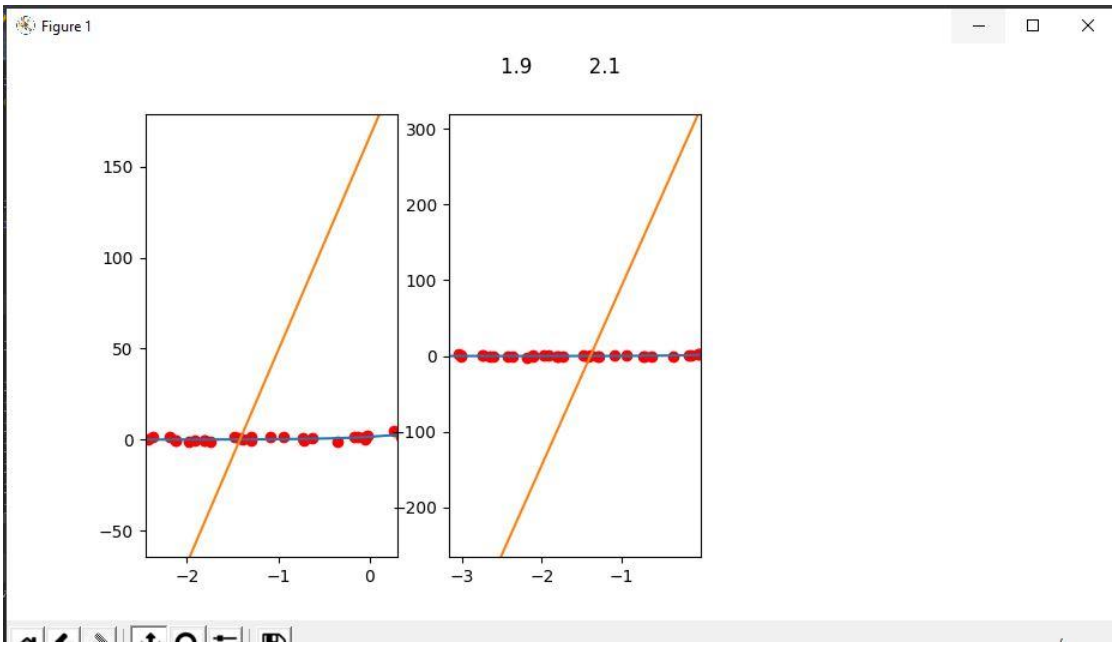


בזום :





בזום :



ניתן לראות שכול עוד האלפא גדל השגיאה בין כלל הניבוי והפונקציה גדלה.

שכול עוד האלפא גדל הרעש אפקט הרעש קטן .

הקוד שייצר את הגרפים :

```
plt.axis([-6, 6, -6, 6])

plt.subplot(131)
plt.suptitle("0.1     0.3      0.5")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[0], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[0])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[0])
plt.subplot(132)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[1], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[1])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[1])
plt.subplot(133)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[2], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[2])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[2])
plt.show()

plt.suptitle("0.7     0.9      1.1")
plt.subplot(131)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[3], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[3])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[3])
plt.subplot(132)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[4], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[4])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[4])
plt.subplot(133)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[5], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[5])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[5])
plt.show()
```

```
plt.suptitle("1.3     1.5      1.7")
plt.subplot(131)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[6], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[6])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[6])
plt.subplot(132)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[7], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[7])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[7])
plt.subplot(133)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[8], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[8])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[8])
plt.show()

plt.suptitle("1.9         2.1")
plt.subplot(131)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[9], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[9])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[9])
plt.subplot(132)
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet[10], "ro")
plt.plot(Xs_training, all_Alphas_Ys_Training_to_Xs_trainingSet_withOut_noice[10])
plt.plot(Xs_training, all_predicted_Ys_from_Xs_trainingSet[10])

plt.show()
```

סעיף 3:

בסעיף זה השתמשתי בנתונים שאני שומר בטבלה של סעיף 2 בכל הרצה .

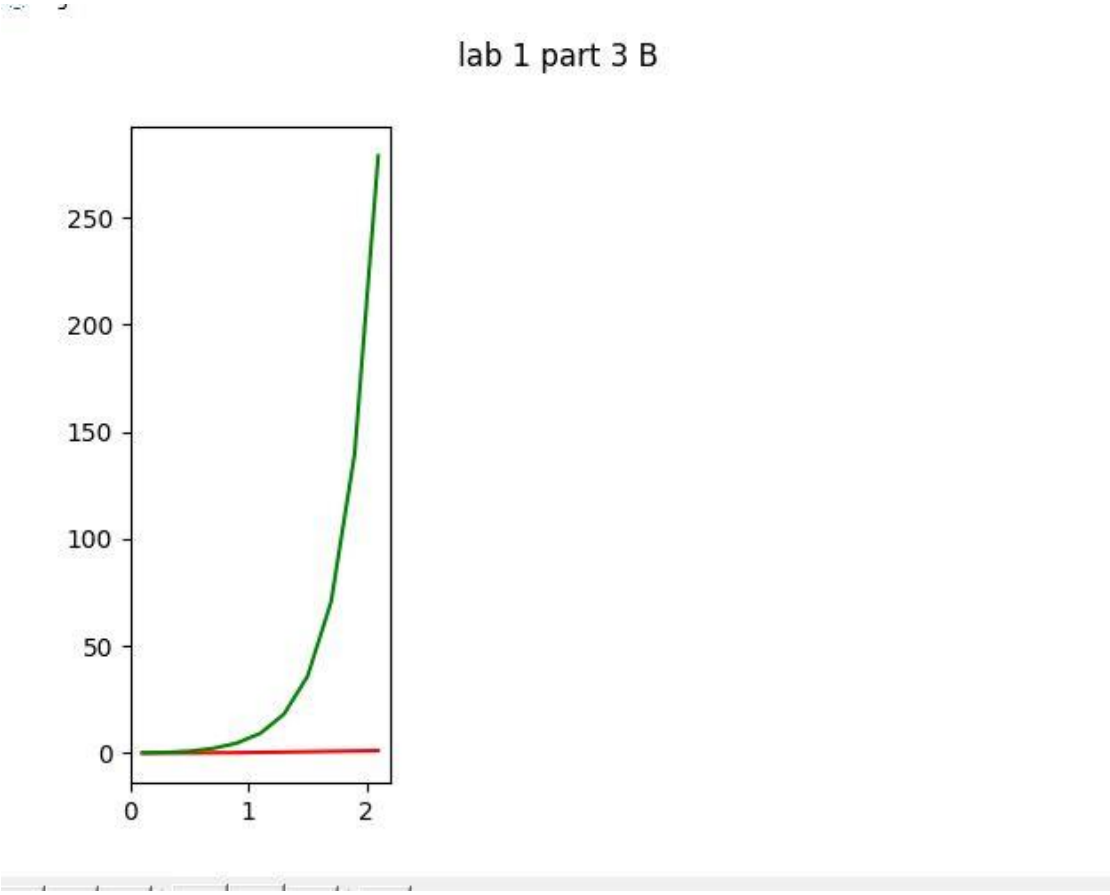
הפלט הוא מריצת הדגמה שונה אחרת לא קשורה לריצת הדגמה בסעיף 2.

קוד למימוש סעיף זה :

```
#self 3

Cs_cova= []
as_cova = []
for row in tableOfResults_q2_A_and_B:
    t1 = row["teta_1_cova"][0]
    t0 = row["teta_0_cova"]
    Cs_cova.append(t0)
    as_cova.append(t1/t0)

Cs_cova_Abs_diff_C = []
as_cova_Abs_diff_alphas = []

for c in Cs_cova:
    res = np.abs(c-C)
    Cs_cova_Abs_diff_C.append(res)

for i in range(len(alphas)):
    res = np.abs(as_cova[i]-alphas[i])
    as_cova_Abs_diff_alphas.append(res)

for i in range(len(alphas)):
    print("alpha : ",alphas[i], "|C^ - C | = ",Cs_cova_Abs_diff_C[i],"|a^ - a| = ", as_cova_Abs_diff_alphas[i],"\n")

plt.subplot(131)
plt.suptitle("lab 1 part 3 B")
plt.plot(alphas, as_cova_Abs_diff_alphas,'r')
plt.plot(alphas, Cs_cova_Abs_diff_C,'g')
plt.show()
```

פלט הקוד שהוא תשובה לסעיף א ו ב בהתאמה :

```
DONE
alpha :  0.1 |C^ - C | =  0.04674077266610244 |a^ - a| =  0.019262467991930363

alpha :  0.30000000000000004 |C^ - C | =  0.2965706086032758 |a^ - a| =  0.01829084498231276

alpha :  0.5 |C^ - C | =  0.8619635420754568 |a^ - a| =  0.05622221829284457

alpha :  0.7000000000000001 |C^ - C | =  2.1942859221400113 |a^ - a| =  0.12843452302531233

alpha :  0.9 |C^ - C | =  4.660481199063817 |a^ - a| =  0.1954355371858909

alpha :  1.1 |C^ - C | =  9.220594003023216 |a^ - a| =  0.32675801619254674

alpha :  1.3000000000000003 |C^ - C | =  18.16024739923717 |a^ - a| =  0.4706709090331551

alpha :  1.5000000000000002 |C^ - C | =  35.791859875076 |a^ - a| =  0.6372345804279023

alpha :  1.7000000000000002 |C^ - C | =  70.51078717632629 |a^ - a| =  0.8072438148787229

alpha :  1.9000000000000001 |C^ - C | =  139.60434700504152 |a^ - a| =  0.9845760627626369

alpha :  2.1 |C^ - C | =  278.75913917188547 |a^ - a| =  1.1653407520238965


Process finished with exit code 0
```



באדום הפרשי האלפות כפונקציה לאלפות שזה נראה גרף מספק

בשביל ערכי אלפא הגבוהים יש הפרש גבוה מידי וזה מה שמאכזב וזה החולשה בשיטה הזו

סעיף 4 :

סעיף זה מומש בסקריפט אחר שנקרא task1_d . בגלל רצון לנוחות עבודה שלי

לטובת סעיף זה היה צריך ליצור סטי אימון ומבחן וסטי אימון ומבחן לוגריתמים.

אז בסעיף זה ממשתי פונקציה שמחשבת בינתן סט מסוים את סט הלוגריתמי המתאים .

להלן הפונקציה :

```
def calculateLogset(xs_set,ys_set,sizeOfSet):
    for i in range(sizeOfSet):
        if ys_set[i][0] <= 0:
            ys_set[i]=None
            xs_set[i]=None

    xs_set = list(filter(lambda x: x != None, xs_set))
    ys_set = list(filter(lambda y: y != None, ys_set))
    ys_set = list(map(lambda y: np.log(y), ys_set))
    return xs_set,ys_set
```

כמו בסעיף הנה הקוד שרק יוצר לי את סטי האימון והמבחן הרלוונטיים לסעיף זה כמובן לכל אלפא :

```
lr = LinearRegression()
alphas = np.linspace(0.1,2.1,11)

All_XS_FOR_Log_Training_SET=[]
All_YS_FOR_Log_Training_SET=[]
All_YS_FOR_Log_Training_SET_PRED=[]

All_XS_FOR_Log_Test_SET=[]
All_YS_FOR_Log_Test_SET=[]
All_YS_FOR_Log_Test_SET_PRED=[]

Cs_cova_from_Log_set = []
as_cova_from_Log_set = []

Xs_Training_set = sorted(np.random.uniform(-4, 4, N).reshape(-1, 1))
Xs_Test_set = sorted(np.random.uniform(-4, 4, N).reshape(-1, 1))

for a in alphas:
    Xs = list(Xs_Training_set)
    Xs_t = list(Xs_Test_set)
    All_XS_FOR_Log_Training_SET.append(Xs)
    All_XS_FOR_Log_Test_SET.append(Xs_t)

for i in range(len(alphas)):
    All_YS_FOR_Log_Training_SET.append(makeOneYs(All_XS_FOR_Log_Training_SET[i],y_func_with_nocice=y_with_nocie,alpha=alphas[i],sigma=task_sigma))
    All_YS_FOR_Log_Test_SET.append(makeOneYs(All_XS_FOR_Log_Test_SET[i],y_func_with_nocice=y_with_nocie,alpha=alphas[i],sigma=task_sigma))

for i in range(len(alphas)):
    All_XS_FOR_Log_Training_SET[i],All_YS_FOR_Log_Training_SET[i] = calculateLogset(All_XS_FOR_Log_Training_SET[i],All_YS_FOR_Log_Training_SET[i],N)
    All_XS_FOR_Log_Test_SET[i],All_YS_FOR_Log_Test_SET[i] = calculateLogset(All_XS_FOR_Log_Test_SET[i],All_YS_FOR_Log_Test_SET[i],N)
```

תת סעיף א :

הפונקציה שנדרשה בקוד פייתון :

```
def get_C_cova_And_a_cova__from_Log_trainingset(x_set,log_y_set):
    lr = LinearRegression()
    lr.fit(x_set, log_y_set)
    beta_0 = lr.intercept_[0]
    beta_1 = lr.coef_.tolist()[0][0]
    a_cova = beta_1
    c_cova = np.exp(beta_0)
    return a_cova, c_cova
```

בעצם עושה את אותם פעולות של חישוב כלל הניבוי לסט מסוים מחשב את אלפא כובע וC כובע כפי שנדרש בסעיף זה ומחזיר אותם.

לפי דרישת התרגיל נצטרך להשתמש בערכים האלה לחשב את ערכי הניבוי עלי הפונקציה ללא הרעש שהצגתי בתחילת המסמך .

להלן פונקציה שמקבל C כובע ו a כובע וסט X ומחזירה סט Y מנובא מתאים .

```
def y_cova_func_for_d (x,c_cova,a_cova):
    return c_cova * np.exp(a_cova*x)

def makePredYs_for_task1D(x_set,c,a):
    y = []
    for x in x_set:
        y.append(y_cova_func_for_d(x,c,a))

    return y
```

תת סעיף ב :

מימוש בקוד לפי דרישות התת הסעיף הזה בעזרת קוד העזר שכתבתי בסקריפט המתאים לסעיף זה קוד יחסית דומה לקוד בסעיף 2 ועובד עלפי אותו רעיון רק שעכשיו ערכי הניבוי מתקבלים בצורה שונה .

```
for i in range(len(alphas)):
    a, c = get_C_cova_And_a_cova__from_Log_trainingset(All_XS_FOR_Log_Training_SET[i],All_YS_FOR_Log_Training_SET[i])
    Cs_cova_from_Log_set.append(c)
    as_cova_from_Log_set.append(a)
    All_YS_FOR_Log_Training_SET_PRED.append(makePredYs_for_task1D(All_XS_FOR_Log_Training_SET[i],c,a))
    All_YS_FOR_Log_Test_SET_PRED.append(makePredYs_for_task1D(All_XS_FOR_Log_Test_SET[i],c,a))

## --after this we have results and we can calc TSE & TE

task1_D_2_res_Table = []

for i in range(len(alphas)):
    size = len(All_YS_FOR_Log_Training_SET[i])
    tse = calcLostFunc(All_YS_FOR_Log_Training_SET[i],All_YS_FOR_Log_Training_SET_PRED[i])/size
    sizete = len(All_YS_FOR_Log_Test_SET[i])
    te = calcLostFunc(All_YS_FOR_Log_Test_SET[i],All_YS_FOR_Log_Test_SET_PRED[i])/sizete
    row = {"alpha : ": alphas[i],"TSE": tse[0],"TE":te[0]}
    task1_D_2_res_Table.append(row)


print(as_cova_from_Log_set)
print(Cs_cova_from_Log_set)

for row in task1_D_2_res_Table:
    print(row, "\n")
```

פלט לפי המתבקש בתת סעיף ב

```
{'alpha : ': 0.1, 'TSE': 1.6106188820060725, 'TE': 1.6363342868648005}

{'alpha : ': 0.30000000000000004, 'TSE': 2.9213964230980167, 'TE': 3.0991155493086517}

{'alpha : ': 0.5, 'TSE': 7.854966814315196, 'TE': 7.565066252876286}

{'alpha : ': 0.7000000000000001, 'TSE': 31.548390755398486, 'TE': 31.9429973558237}

{'alpha : ': 0.9, 'TSE': 91.01585529396127, 'TE': 92.98553125499143}

{'alpha : ': 1.1, 'TSE': 439.8402168386002, 'TE': 423.01241736347225}

{'alpha : ': 1.3000000000000003, 'TSE': 1605.441757604349, 'TE': 1389.2751089149044}

{'alpha : ': 1.5000000000000002, 'TSE': 4922.1256354961, 'TE': 4546.163448892039}

{'alpha : ': 1.7000000000000002, 'TSE': 12563.257217664459, 'TE': 9620.662228887699}

{'alpha : ': 1.9000000000000001, 'TSE': 99876.13002788457, 'TE': 75163.32863067287}

{'alpha : ': 2.1, 'TSE': 214897.89392654126, 'TE': 146515.45736101057}
```

תת סעיף ג :

1. קיים הבדל השגיאות הממוצע גם בסט המבחן וגם בסט האימון גדול בצורה משמעותית מאלפא נמוכה לאלפא גבוה יותר

מספר חולשות :

1. בחישוב סט הלוגריתמי אנו מאבדים תצפיות עקב פעולת הלוג.
2. לפי התוצאות ניתן לראות שהשגיאה יותר גדולה משמעותית מסעיף 2
3. שיטה זו מסובכת ומורכבת מידי

הערות חשובות וטכניות בנוגע למעבדה :

בין סעיפים 1 – 3 בוצעו כמה הרצות שונות (לכאורה סטי X שונים)לטובת ההדגמה עם תרצה ניתן להריץ את הסקריפט task1 ותקבל את כל הפלטים שתרצה לסט X אחד מסוים.

יצרתי סקריפט בנפרד לסעיף 4 לטובת הנוחות שלי

מכיוון התפלגות האחידה של X ברוב הרצות הX היה בקירוב יחסי אותו .

רוב הרצות קיבלתי פלטים עקבים מאוד באותה סביבת המספרים .

אם אתה מריץ את task1 שזו תשובה לסעיפים 1- 3 יקפצו לך חלוניות הגרפים שהראיתי צילומים שלהם בסעיף 2 ו 3 יש לסגור את חלונית הגרף בשביל לעבור לחלונית האחר עד שתוכנית הסקריפט תגמר .

בנוסף לPDF הזה תקבל את הסקריפטים ואת קבצי סטי האימונים של סעיף 2 וקובץ התשובות של סעיף 2 כקובץ טקסט כמו כן אלו היו תוצאות מהרצה האחרונה שאני עשיתי .

ביבליוגרפיה

הנה 2 אתרים שעזרו לי במעבדה :

אתר שמסביר על המודול של sklearn זה מודול שעשה לי את חישובי פונקציית הניבוי לפי שיטת הריבועים הפחותים .
הסבר על המודול באתר הזה יחסית למטה בדף האינטרנט.

https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0

אתר שמסביר על המודול של גרפי המטלב בפייתון .

https://matplotlib.org/3.1.1/tutorials/introductory/pyplot.html