

סדנת פייתון למעבדה 2 – פרק א'

עיבוד נתונים בסיסי

בסדנה זו תלמדו לעבד את נתוני המעבדה בעזרת שפת Python. אנחנו נשתמש בתוכנה Spyder לכתיבה והרצה של הקוד. המטרה היא ליצור מיומנות בסיסית של ניתוח נתונים והצגתם בגרפים כדי שתוכלו לעבד את הנתונים של המעבדה בקלות. בנוסף, הקוד שתכתבו בסדנה יישמר אצלכם ותוכלו לחזור ולהיעזר בו בעתיד.

היקף הסדנה הוא **שעתיים**, בסופה יהיו ברשותכם קבצי קוד שיכילו את כל הפקודות שלמדתם ותוכלו לעשות בהם שימוש במהלך המעבדה.

תוכן הסדנה

- 2..... חלק א' – התקנת ANACONDA ו-SPYDER
- 2..... חלק ב' – סביבת עבודה ב-SPYDER
- 2..... חלק ג' – עיבוד נתוני המעבדה (מתחילים לעבוד)
- 1. כתיבת וקטור נתונים (כל כללי כתיבת וקטורים ב-18 סעיפי תרגול)..... 4
- 2. ביצוע חישוב על וקטורי נתונים, כתיבת פונקציה (10 סעיפים וגם אתם כותבים פונקציות)..... 5
- 3. חישוב עקום תיאורטי (8 סעיפים ואתם שם)..... 6
- 4. טעינת נתונים מקובץ (11 סעיפי תרגול)..... 7
- 5. הצגת נתונים בגרף (4 סעיפים ליצירת גרף)..... 8
- 6. ביצוע רגרסיה לינארית (19 סעיפים לכלי שימושי)..... 9
- 7. ביצוע אינטגרציה לווקטור נתונים (3 סעיפי ביצוע אחרונים)..... 10
- 11..... סיכום

חלק א' – התקנת Anaconda ו-Spyder

בצעו התקנה של Anaconda:

[/https://docs.anaconda.com/anaconda/install/windows](https://docs.anaconda.com/anaconda/install/windows)

התקנה זו מתקינה גם Spyder.

חלק ב' – סביבת עבודה ב-Spyder

הערכת זמן: כ- 15 דק'

1. צרו תיקיית קבצים ריקה שתוקדש לסדנה.

2. פתחו את Spyder.

3. הסתכלו על הממשק של Spyder וזהו את החלקים הבאים:

a. סרגל פקודות ותפריטים, תיקיית ההרצה.

b. חלון הקובץ – אזור כתיבת הקוד.

c. ה-Console, שימו לב שישנם שני Tabs לחלון זה.

d. חלון עזרה, שימו לב שישנם 4 tabs: Help, Variable Explorer, Plots, Files

4. שנו את תיקיית ההרצה לתיקיית הסדנה – כדי שהקוד יוכל למצוא את הקבצים שנמצאים בה.



5. עיברו עם העכבר מעל האייקונים של פקודות ההרצה

וקראו את קיצורי המקשים של הרצת קוד ב-Spyder.

הסעיף האחרון (5) היה חשוב במיוחד כי אתם תצטרכו להריץ את כל שורות הקוד שתכתבו (גם אם לא נדרשתם במפורש) – על מנת לוודא שהן עובדות כמו שציפיתם. שוב, בדקו עם עצמכם/המדריך/השכן שהבנתם איך להריץ את הקוד שתכתבו.

חלק ג' – עיבוד נתוני המעבדה (מתחילים לעבוד)

באופן כללי, הפעולות הנדרשות לעיבוד הנתונים במעבדה הן:

1. כתיבת וקטור נתונים
2. ביצוע חישוב על וקטורי נתונים (כתיבת פונקציה)
3. חישוב עקום תיאורטי
4. טעינת נתונים מקובץ
5. הצגת נתונים בגרף
6. ביצוע רגרסיה לינארית
7. ביצוע אינטגרציה לוווקטור נתונים

ואלו הן בדיוק הפעולות שתלמדו בפרק זה. כמו שאמרנו, זהו שיעור ב"עיבוד נתונים בסיסי".

כדי לעבד את הנתונים, נשתמש בmodules (ספריות פיתון המכילות פונקציות ואובייקטים סטנדרטיים) הבאים:

```
import numpy as np # math functions

import scipy # scientific functions

import matplotlib.pyplot as plt # for plotting figures and setting the figures' properties

import pandas as pd # handling data structures (loaded from files)

from scipy.stats import linregress # for linear regression (fitting a line to data points)
```

1. פתחו קובץ קוד חדש בשם `data_analysis.py` ושמרו אותו בתיקיית העבודה של הסדנה.
2. העתיקו את השורות האלה לתחילת הקובץ של הקוד והריצו אותו.

אתם בוודאי תוהים מה הקוד שהרצתם עושה, אז נשים לב למספר כללים שיעזרו לכם:

- הפקודות בפיתון חייבות להיכתב בתחילת השורה! בלי רווחים. רווח כזה יגרום להודעת שגיאה unexpected indent (מוזמנים לנסות).
- שפת פיתון היא case sensitive, כלומר My_temp ו-my_temp אלו שני משתנים שונים.
- הערות בקוד מתחילות בסימן "#". כדי להפוך קוד להערה ולהיפך בspyder לוחצים ctrl+1.
- לפיתון יש ספריות שמכילות פונקציות מוכנות. כדי לייבא ספרייה (להפוך את הפונקציות שבה לזמינות לשימוש בקוד) משתמשים בפקודה import ואחריה שם הספרייה.
- אפשר לתת כינוי לספרייה בעזרת המילה as כדי לקצר את שם הספרייה כשתפנו אליה בהמשך.
- לספריות יכולות להיות תתי ספריות, למשל pyplot היא תת ספרייה של matplotlib.
- במקום ספרייה, ניתן לייבא רק פונקציה ספציפית מתוכה, למשל הפונקציה linregress מתוך תת הספרייה stats של הספרייה scipy.

3. קראו את ההערות הרשומות ליד כל פקודת import והשלימו את הטבלה:

הפעולה שנרצה לעשות	הספריות שיכילו את הפונקציות הרלוונטיות (שמות מקוצרים) או הפונקציה עצמה
1. כתיבת וקטור נתונים	
2. ביצוע חישוב על וקטורי נתונים (כתיבת פונקציה)	
3. חישוב עקום תיאורטי	
4. טעינת נתונים מקובץ	
5. הצגת נתונים בגרף	
6. ביצוע רגרסיה לינארית	
7. ביצוע אינטגרציה לזקטור נתונים	

1. כתיבת וקטור נתונים (כל כללי כתיבת וקטורים ב-18 סעיפי תרגול)

1. צרו cell חדש ("%#") ורשמו לידו את הכותרת data vectors
2. בתוך ה-cell, כתבו `vec1=[1,2,3]` והריצו את ה-cell (ctrl+enter)
3. הסתכלו בvariable explorer וענו על השאלו הבאות:
 - a. מהו הסוג של האובייקט `vec1`?
 - b. מה האורך של האובייקט `vec1`?
 - c. הקליקו על `vec1`. מהו סוג האובייקטים שבתוך `vec1`?

האובייקט הבסיסי של פייתון הוא רשימה (list):

- יוצרים אותה בעזרת סוגריים מרובעים.
- ניתן לשים ברשימה כל אובייקט שהוא.
- פנייה לחלק מהאיברים ברשימה נקראת list slicing. באופן בסיסי הפנייה היא בפורמט `mylist[start index:end index:index step]`. שימו לב: ה-`end_index` עצמו לא נכלל.
- כדי להוסיף איבר לסוף הרשימה, משתמשים בפקודה `append()`

בואו נשחק עם זה קצת:

4. הוסיפו ל-`vec1` את האיבר 12 בעזרת `vec1.append(12)`
5. הפכו את האיבר הראשון ל-24 בעזרת `vec1[0]=24`
6. הדפיסו את שלושת האיברים הראשונים בעזרת `print(vec1[0:3])`
7. הדפיסו את האיברים באינדקסים הזוגיים בעזרת `print(vec1[0::2])`
8. הדפיסו את כל האיברים בסדר הפוך בעזרת `print(vec1[::-1])`
9. הדפיסו את האיבר האחרון בעזרת `print(vec1[-1])`
10. הדפיסו את האורך של הרשימה בעזרת הפונקציה `len()`: `print(len(vec1))`
11. הדפיסו את שני האיברים האחרונים, בסדר הפוך. (כי למה לא... בהצלחה...)

למידע נוסף – חפשו python list slicing בגוגל.

12. כתבו `vec2=vec1/2` והריצו את ה-cell.

a. מהי הודעת השגיאה שקיבלתם?

מהי הבעיה? מכיוון list יכול להכיל כל מיני סוגי אובייקטים, לא מוגדר מה הפונקציה של חילוק ("/") צריכה לבצע והתוכנה מודיעה על שגיאה.

אנחנו נרצה לבצע פעולות מתמטיות על וקטורי הנתונים שלנו, לכן עלינו להשתמש בספרייה numpy (או בקיצור – np).

האובייקט הבסיסי של np הוא מערך (array). על `np.array` ניתן לבצע את כל הפעולות המתמטיות שנרצה. ה-slicing הוא זהה לזה של list.

לפיכך: תמיד נקליד וקטורי נתונים כמערכי numpy.

תמיד!

13. מחקו את השורה שגורמת לשגיאה והפכו את `vec1` ל-`np.array` על ידי `vec1 = np.array(vec1)`. הסתכלו על ה-type שלו בvariable explorer.
14. בדקו שכעת ניתן לחלק אותו ל-2, ושחישוב על `np.array` מחזיר משתנה מסוג `np.array`.
15. צרו וקטור נתונים חדש בשם times ורשמו בו את הערכים 0.1, 0.12, 0.23, 0.45, 0.5. (בשורת קוד אחת בבקשה, ושיהיה מה-type הנכון).

16. **ציינו** בהערה בסוף הפקודה שהיחידות הן sec והשגיאה היא ± 0.01 (למשל, **כתבו** `sec, +/-0.01`)
17. **הריצו** את הקוד **וודאו** שקיבלתם את ה `type` הנכון עבור `times`.

הסתכלו על השורה שכתבתם – זה הפורמט בו תקלידו וקטורי נתונים במעבדה. יש להקפיד תמיד לציין יחידות והערכת שגיאה, והווקטור צריך להיות מסוג `numpy.array`.

18. **הפכו** את כל פקודות ההדפסה להערות (`ctrl+1`) **והריצו** את `cell`. **וודאו** שלא מודפס שום דבר.

2. ביצוע חישוב על וקטורי נתונים, כתיבת פונקציה (10 סעיפים וגם אתם כותבים פונקציות)

לצורך התרגיל, נניח שהווקטור `times` מכיל זמנים בהם מדדנו נפילה חופשית של גוף כלשהו, תחת תאוצה הכבידה. כעת נחשב את המיקום (הגובה) של הגוף בזמנים הנתונים שכביכול מדדנו במעבדה.

1. **פתחו** `cell` חדש **ורשמו** לו את הכותרת `vector calculations`
2. **צרו** משתנה `x0=0.8` **וציינו** יחידות של `m`. **צרו** משתנה `v=1.6` **וציינו** יחידות `m/sec`. **צרו** משתנה `a=-9.8` **וציינו** יחידות `m/sec**2` (כן, יחידות זה חשוב. "משיקולי יחידות" וכו')
3. **חשבו** את הווקטור `distances` לפי משוואת התנועה, עבור המיקום ההתחלתי `x0`, המהירות `v`, התאוצה `a` והזמנים `times`. יש לכתוב שורת קוד אחת. **ציינו** את היחידות בסוף השורה.

כפי שראיתם, הפעולות המתמטיות הבסיסיות פועלות על `numpy.array` כצפוי. כדי לבצע פעולות יותר מסובכות, נשתמש מפורשות בספריית `numpy`. באופן כללי, כל פונקציה מוכרת (כזאת שאתם יכולים להקליד למנוע חיפוש של `wolfram alpha` או למחשבון) קיימת ב `numpy`. דוגמאות: `np.abs()`, `np.sqrt()`, `np.arctan()`.

4. **חשבו** את השגיאה הנגררת של וקטור המרחקים `distances_err` לפי הנוסחה:

$$distances_err = \left| \frac{dx}{dt} \right| \cdot \delta t = |v + a \cdot t| \cdot 0.01$$

5. **עגלו** את השגיאה לספרה הדומיננטית. לשם כך **הקלידו** ב `console` `"np.round?"` ולפי התיעוד של הפונקציה **כתבו** את הפקודה המתאימה.

6. **פתחו** `cell` חדש **וקראו** לו `defining functions`.

נדגים כעת כתיבה של פונקציה. בכל שפת תכנות, לפונקציות יש 3 תכונות:

- שם הפונקציה (למשל `functionName`)
- הארגומנטים שהיא מקבלת - `input`
- הערכים שהיא מחזירה - `output`

והשימוש בפונקציה ("הקריאה לפונקציה") הוא תמיד באמצעות הפקודה

`the_corresponding_output = functionName(some_specific_input)`

הסינטקס שבו מגדירים את הפונקציות משתנה משפה לשפה. פונקציה בפיתון כותבים באופן הבא:

```
def functionName(argument1, argument2, argument3):  
    # some calculation on the arguments, for example  
    temp1 = np.max(argument1) + 50 * argument2  
    return np.abs(np.sqrt(argument3)) - temp1
```

נשים לב:

- פונקציה נפתחת במילית השמורה def (קיצור של define)
- הארגומנטים לפונקציה מוגדרים בסוגריים עגולים "()" .
- שמות הארגומנטים מוגדרים בשורת ה-def וקיימים רק בתוך הקונטקסט של הפונקציה. אם אתם משתמשים בשם של משתנה שמוגדר כבר מחוץ לפונקציה, ערך הארגומנט "ידרוס" את הערך הזה בתוך הקונטקסט של הפונקציה.
- השורה def מסתיימת בנקודותיים שמסמנות פתיחה של code block והשורות שבתוך הcode block מוזנחות (indented) tab בודד או 4Xspace במקלדת (ביחס לשורה של ה-":"). שורות שאינן בתוך הפונקציה ייכתבו בהזחה רגילה – כלומר ללא הזחה. (אפשר לכתוב code block שהוא בתוך code block אחר וכן הלאה. ההזחה מסמנת את ההיררכיה.)
- הערך שמוחזר מהפונקציה נכתב אחרי המילית .return.
- אפשר לכתוב פונקציה בתוך פונקציה. או slicing אחרי פונקציה, בתוך פונקציה. או slicing של slicing בתוך פונקציה של פונקציה. בכללי בפיתון, התשובה לשאלה "האם אפשר..." היא בדרך כלל "כן, תנסו!" וכדאי לצאת מנקודת ההנחה הזאת כשכותבים קוד בפיתון (=)
- בניגוד ל-list או פעולות slicing שמשמשות בהן בסוגריים מרובעים "[]", סוגריים עגולים יוצרים אובייקט tuple – כמו רשימה, אבל אי אפשר לשנות את התוכן של tuple אחרי שהוא נוצר. אם רוצים להחזיר מהפונקציה מספר ערכים, מחזירים אותם כ-tuple.

לדוגמה:

```
def func1(arg1, arg2):  
    return (arg1/2, arg2+5)  
(A,B) = func1(2.2,15)
```

- שאלה: אילו משתנים יופיעו ב-variable explorer אחרי הרצה של הקוד הזה ומה יהיו הערכים שלהם?
- נניח שנרצה לדעת מתי הגוף הנופל עובר גובה מסוים. כתבו פונקציה בשם timeOfCrossing שמקבלת וקטור זמן t, וקטור מרחק d וערך חצייה cross_value ומחזירה את הזמן בו המרחק של הגוף היה הכי קרוב לערך החצייה (הזמן שמתאים לאיבר ב-d שהוא הקרוב ביותר ל-cross_value בערך מוחלט). ניתן להשתמש בפונקציה np.argmin().
- אם נריץ את cross_time = timeOfCrossing(times, distances, 0.5), מה הערך שיתקבל ב-cross_time?
- בדקו שאתם אכן מקבלים את הערך הרצוי.

3. חישוב עקום תיאורטי (8 סעיפים ואתם שם)

לעיתים נרצה לחשב וקטורים תיאורטיים, למשל כדי לפתור משוואות בצורה נומרית או כדי לשרטט ערכים תיאורטיים ליד נתונים מדודים. כעת נדגים איך לעשות זאת.

1. **פתחו** cell בשם theoretical_vectors.
 2. בחלון help **כתבו** בחיפוש np.linspace. **קראו** רק את התיאור שמסביר מה הפונקציה עושה ואת הדוגמה הראשונה (הדוגמאות מופיעות למטה).
 3. בעזרת np.linspace(), **צרו** ווקטור בשם theoretical_t שמכיל 30 זמנים בין 0 ל-1 שניות (אתם מוזמנים להעתיק את הדוגמה ולערוך אותה בהתאם לצרכים שלכם). **הריצו** את ה-cell **וודאו** שקיבלתם את הערכים הנכונים.
 4. **הדפיסו** את קבוע הכבידה בעזרת scipy.constants.g. אם אתם מקבלים הודעת שגיאה, **בצעו** ייבוא מפורש (import) לתת הספרייה scipy.constants.
 5. **כתבו** פונקציה בשם theoreticalCrossingTime שמקבלת וקטור זמנים t, מרחק התחלתי x0, מהירות התחלתית v וערך חצייה cross_value. הפונקציה צריכה להשתמש בג כתאוצת הכבידה ובפונקציה timeOfCrossing שכבר כתבתם. היא תחזיר את הזמן התיאורטי בו התנועה תחצה את הערך cross_value.
 6. **השתמשו** בפונקציה שכתבתם כדי לחשב את theoretical_cross_time עבור cross_value=0.5 והמרחק ההתחלתי והמהירות שהיו נתונים בסעיפים הקודמים. האם קיבלתם זמן הגיוני? _____
 7. בכמה משתנה התשובה אם אתם מגדירים שווקטור הזמנים התיאורטי יהיה באורך 1000? _____
 8. בכמה שונה התשובה התיאורטית הכי מדויקת מזו שחישבתם מתוך וקטור ה"נתונים"? _____
- לסיכום, כאשר אתם רוצים לחשב ווקטור תיאורטי, אתם תשתמשו ב-np.linspace() כדי להגדיר ערכים התחלתיים ואחר כך תבצעו עליהם את החישובים המתאימים.
- כמו כן, תוכלו למצוא קבועים פיזיקליים ספרותיים ב-scipy.constants. ראו את תיעוד הספרייה לפירוט נוסף.

4. טעינת נתונים מקובץ (11 סעיפי תרגול)

הספרייה שמאפשרת טעינת נתונים מקובץ היא pandas. הספרייה מכילה גם פקודות רבות לעיבוד וסינון של נתונים. אנחנו נשתמש בה רק כדי לטעון אותם. $\sqrt{\frac{1}{2}}$

האובייקט הבסיסי של ספריית pandas הוא pd.DataFrame, או בקיצור df.

1. **הסתכלו** בחלון files.
 2. **וודאו** שהקובץ Trace_0.csv נמצא בתיקיית העבודה שלכם **והקליקו** עליו.
- הקיצור csv מסמן "comma separated values".
- הקובץ נוצר על ידי אחד ממכשירי המעבדה, האוסצילוסקופ, המשמש למדידת מתחים חשמליים.
3. האם אתם מזהים את הנתונים? ה"ערכים שמופרדים על ידי פסיקים"? **נסו להבין**, כמה עמודות של נתונים יש בקובץ? _____
 4. לפי גוש הטקסט בתחילת הקובץ, באיזו שורה מופיעים השמות של עמודות הנתונים (**חשבו**, מה סביר שיהיו השמות של עמודות הנתונים)? _____
 5. **צרו** cell חדש בשם loading_data_from_file
 6. **כתבו** את הפקודה (data=pd.read_csv('Trace_0.csv',header=1) **והריצו** אותה.

הערות:

- מחרוזת (string) בפייתון מסומנת בגרש או גרשיים (שניהם עובדים).
- טעינת הקובץ מתבצעת באמצעות הפקודה pd.read_csv(), שכשמה כן היא, קוראת קבצי csv (אבל לא רק!)

- הפרמטר header שמועבר לפונקציה אומר לה מהי השורה שבה נמצאות הכותרות של העמודות. במקרה שלנו זו השורה השנייה, שהמספר הסידורי שלה הוא 1.
7. **הסתכלו** ב-variable explorer מהו הטיפוס של data? _____
a. מהו הגודל של data? _____
b. כאשר מקליקים על data, לפי מה נקבעים הצבעים? _____
c. מהן הכותרות של שלוש עמודות הנתונים הראשונות (שימו לב, אתם יכולים להעתיק אותן מקובץ ה-csv)? _____
d. **העתיקו** לקוד שלכם והריצו את השורות הבאות:

```
t = data['Time (s)']  
ch1 = data['1 (VOLT)']  
ch2 = data['2 (VOLT)']
```

9. מה ה-typen של t? _____
10. **תקנו** את השורות שהעקתם והפכו את המשתנים t, ch1, ch2 ל-np.array.
11. **בדקו** שקיבלתם את הערכים הנכונים ושהם מתאימים לנתונים שמופיעים בקובץ csv.

וזוהו, ככה טוענים נתונים מקובץ.

4 שורות קוד.

...

אפשר לשים אותן בפונקציה loadData() שתגדירו, אם 4 שורות זה יותר מידי.

ועכשיו ברצינות: באופן עקרוני, כל עוד הקובץ שאתם מנסים לטעון מכיל נתונים שרשומים בצורה "מסודרת" במובן כלשהו, ניתן להסביר לפונקציה pd.read_csv() איך לקרוא אותו. לכן הפורמט לא חייב להיות csv.

5. הצגת נתונים בגרף (4 סעיפים ליצירת גרף)

נפנה לשם כך לספריית matplotlib.pyplot בפייתון. כל הפונקציות הקשורות לציור ועיצוב גרף נמצאות ב-plt. הצורה הכללית להכין גרף היא:

```
fig1 = plt.figure(dpi=300)  
plt.plot(x1, y1, 'b.', label = 'first plot')  
plt.plot(x2, y2, 'ro', label = 'second plot')  
plt.plot(x3, y3, 'g--', label = 'third plot')  
  
plt.legend()  
plt.grid()  
plt.xlabel('label of x-axis')  
plt.ylabel('label of y-axis')
```

מה זה עושה? טוב ששאלתם!

- Figure הוא החלון בו מצויר הגרף. plt.figure() פותח חלון חדש ומחזיר ערך שנשמר במשתנה fig1. אם רוצים להמשיך לכתוב עקומים נוספים על אותו חלון, משתמשים בפקודה plt.figure(fig1) לפני שמציירים את העקומים הנוספים. הגדרת הפרמטר dpi לערך של 300 מגדירה את רזולוציית הגרף (dpi = "dots per inch"), בהנחה שהוא בגודל מסוים (באינצ'ים, מסתבר).
- plt.plot() מציירת את העקומים. צריך לתת לה וקטור של x ווקטור של y. לאחר מכן אפשר לפרט איך לצייר את הנתונים (למשל, בעיגולים אדומים – 'ro') ולציין את השם של הנתונים בפרמטר label.
- לאחר שפירטתם את השמות של הנתונים בפרמטר label של פקודת ה-plot, הפקודה plt.legend() מוסיפה לגרף מקרא או מעדכנת אותו.
- grid, xlabel, ylabel חייבים להופיע בכל גרף שאתם מציירים במעבדה! חייבים!

1. פתחו cell חדש בשם plotting a graph
2. צרו גרף fig1 וציירו עליו את הנתונים times ו-distances שהזנתם/חישבתם בתחילת התרגיל. ציירו אותם כ-x שחורים ('kx'). כותרת ציר x צריכה להיות time [sec] וכותרת ציר y צריכה להיות distance [m].
3. צרו גרף fig2 וציירו עליו את ch1 ואת ch2 כתלות ב-t (שני עקומים). כותרת ציר x צריכה להיות time [sec] וכותרת ציר y צריכה להיות voltage [V].
4. בשורות קוד מתחת (כדי לתרגל את plt.figure(fig1)), הוסיפו לגרף fig1 עקום תיאורטי (בעזרת theoretical_t שכבר הכנתם). עדכנו את המקרא.

אופציות העיצוב של גרפים הן כמובן אינסופיות – נרחיב על כך בהמשך. כלומר, בעוד 4~ שבועות. לנקודת הזמן הזו, זה סוג הגרפים שאתם צריכים לדעת לייצר.

6. ביצוע רגרסיה לינארית (19 סעיפים לכלי שימושי)

רגרסיה לינארית היא השם של האלגוריתם שמקבל אוסף של נקודות (כלומר, וקטור x ווקטור y) ומחשב את הקו הישר שעובר הכי קרוב אל הנקודות. הרגרסיה מחזירה את השיפוע (slope) ואת האיבר החופשי (intercept) של הקו הישר. כמו כן, היא מחזירה מקדם שנקרא rvalue שמכמת עד כמה הנקודות "רחוקות" מהקו. כש-rvalue=1 הנקודות נמצאות ממש על הקו. אם הן נמצאות לידו, אז הערך של rvalue קטן יותר.

רגרסיה לינארית היא כלי מאוד שימושי ואתם תשתמשו בה לעיתים קרובות במעבדה.

1. פתחו cell חדש בשם linear regression
2. הכינו וקטור בשם xs שמכיל 40 ערכים בין 0 ל- 2π (חשבו, באיזו ספרייה יהיה שמור הערך של π ?).
3. הכינו וקטור בשם ys ששווה $3 * xs - 0.2$.
4. בחלון help, חפשו linregress (זו הפונקציה שייבאתם בתחילת הקוד שלכם)
5. מצאו דוגמה לשימוש בפונקציה והעתיקו אותה אל הקוד שלכם. את זאת שנראית לכם הכי פשוטה.
6. חפשו בתיעוד של מה שהפונקציה מחזירה (Returns) וגרמו לה להחזיר גם את השגיאה ב-slope ואת השגיאה ב-intercept.
7. בצעו רגרסיה לינארית על xs,ys.
8. שמרו לתוך משתנים את הערכים של slope, intercept, rvalue, stderr, intercept_stderr
9. האם הרגרסיה מצאה את המקדמים הנכונים של הקו הישר? מה השגיאה שהיא העריכה לכל אחד מהם? מהו ה-rvalue?
10. צרו וקטור חדש ys2 שווה $2 * xs + 1$
11. הפכו את האיבר ה-13 לערך של 30.

12. **בצעו** רגרסיה לינארית נוספת, על $xs, ys2$, ושמרו את המקדמים במשתנים חדשים. האם הרגרסיה הצליחה למצוא את המקדמים? _____

בואו נראה זאת בגרף:

13. צרו `fig3` – `figure` חדש.

14. **ציירו** את xs, ys בתור "plot 1", כנקודות שחורות.

15. **ציירו** את $xs, ys2$ בתור "plot 2", כנקודות כחולות.

16. **ציירו** את עקום הרגרסיה הראשונה בעזרת:

```
plt.plot(xs, slope*xs + intercept, 'r-', label='regression 1')
```

17. **ציירו** את עקום הרגרסיה השנייה באופן דומה, אבל בצבע תכלת `cyan`

18. **אל תשכחו** להוסיף מקרא, `grid` וכותרות לציירים!

19. **התבוננו** בגרף. מסקנתכם? _____

המסקנה שהייתם אמורים להסיק:

רגרסיה היא כלי נהדר, אבל תמיד צריך לראות שאין טעויות בנתונים כשמשתמשים בה – ולכן צריך לצייר אותם בגרף.

7. ביצוע אינטגרציה לוקטור נתונים (3 סעיפי ביצוע אחרונים)

למה ללמוד דווקא איך לבצע אינטגרציה? כי מצד אחד זה לא טריוויאלי (ראו הרצאות באינפי) ומצד שני אתם תצטרכו לבצע זאת על נתוני המעבדה.

אנחנו נבצע אינטגרציה בעזרת הפקודה `scipy.integrate.cumtrapz(my_y_vector, x=my_x_vector, initial=0)`

כלומר בשיטת `cumtrapz`, קיצור של `cumulative integral using the trapezoidal method`. בשיטה זו מעבירים קו ישר בין הנקודות השכנות ומחשבים את שטח הטרפז הנוצר מתחת לקו. סכימה של הטרפזים נותנת קירוב נומרי של האינטגרל.

`Cumulative` משמעו "מצטבר" – הפונקציה מחזיר וקטור של ערכים ולא ערך בודד. כלומר, היא מחשבת קירוב של האינטגרל הלא-מסוים, הפונקציה הקדומה.

1. **פתחו** `cell` חדש בשם `numerical integration`

2. **כתבו והריצו** את הפקודה `integral = scipy.integrate.cumtrapz(ys, x=xs, initial=0)`

הסבר:

- כדי לבצע אינטגרציה, מספיק להעביר רק ערכי y , ואז האלגוריתם מניח שערכי x הם $1, 2, 3, \dots$ וכו'.
- אם x לא שווה $1, 2, 3, \dots$ אז אפשר להעביר אותו כפרמטר לפונקציה, תחת השם x . במקרה זה העברנו אליו את xs .
- בלי לציין `initial=0`, וקטור האינטגרל יהיה קצר באיבר אחד מווקטור ה- y , שזה הגיוני אם חושבים על שיטת האינטגרציה אבל גם יכול להפריע לצייר את האינטגרל בגרף. לכן `initial=0` מגדיר לאלגוריתם להוסיף את הערך 0 בתור האיבר הראשון של האינטגרל.

3. **ציירו** בגרף חדש (`fig4`) את האינטגרל של ys ושל $ys2$. (לפי הכללים בבקשה. סומכים עליכם)

התבוננו בגרף ותסכימו איתנו שוב כמה חשוב לוודא שאין טעויות בנתונים לפני שמתחילים לבצע עליהם חישובים מסובכים...

סיכום

סיימנו. שמרו את הקוד שכתבתם. מוזמנים לסכם לעצמכם בטבלה איך עושים כל אחד מהדברים שלמדנו (להעתיק מתוך הקוד שכתבתם):

הפעולה שנרצה לעשות	איך עושים
1. כתיבת וקטור נתונים	
2. ביצוע חישוב על וקטורי נתונים (כתיבת פונקציה)	
3. חישוב עקום תיאורטי	
4. טעינת נתונים מקובץ (4 שורות, זכרים?)	
5. הצגת נתונים בגרף	
6. ביצוע רגרסיה לינארית	
7. ביצוע אינטגרציה לזוקטור נתונים	