## Numerical Analysis

# Final task

Submission date: 18/1/2023 23:59

This task is individual. No collaboration is allowed. Plagiarism will be checked and will not be tolerated.

The programming language for this task is Python 3.7. You can use standard libraries coming with Anaconda distribution. In particular limited use of numpy and pytorch is allowed and highly encouraged.

Comments within the Python templates of the assignment code are an integral part of the assignment instructions.

**You should not use those parts of the libraries that implement numerical methods taught in this course.** This includes, for example, finding roots and intersections of functions, interpolation, integration, matrix decomposition, eigenvectors, solving linear systems, etc.

The use of the following methods in the submitted code must be clearly announced in the beginning of the explanation of each assignment where it is used and will result in reduction of points:

numpy.linalg.solve (15% of the assignment score)

(not studied in class) numpy.linalg.cholesky, torch.cholesky, linalg.qr, torch.qr (1% of the assignment score)

numpy.*.polyfit, numpy.*.*fit (40% of the assignment score)

numpy.*.interpolate, torch.*.interpolate (60% of the assignment score)

numpy.*.roots (30% of the assignment 2 score and 15% of the assignment 3 score)

All numeric differentiation functions are allowed (including gradients, and the gradient descent algorithm).

Additional functions and penalties may be allowed according to the task forum.

You must not use reflection (self-modifying code).

Attached are mockups of for 4 assignments where you need to add your code implementing the relevant functions. You can add classes and auxiliary methods as needed. Unittests found within the assignment files must pass before submission. You can add any number of additional unittests to ensure correctness of your implementation.

In addition, attached are two supplementary python modules. You can use them but you cannot change them.

Upon the completion of the final task, you should submit the four assignment files and this document with answers to the theoretical questions archived together in a file named <your ID>.zip

All assignments will be graded according to **accuracy** of the numerical solutions and **running time**.

Expect that the assignment will be tested on various combinations of the arguments including function, ranges, target errors, and target time. We advise to use the functions listed below as test cases and benchmarks. At least half of the test functions will be polynomials. Functions 3,8,10,11 will account for at most 4% of the test cases. All test functions are continuous in the given range. If no range is given the function is continuous in $[-\infty, +\infty]$.

1. $f_1(x) = 5$
2. $f_2(x) = x^2 - 3x + 5$
3. $f_3(x) = \sin(x^2)$
4. $f_4(x) = e^{-2x^2}$
5. $f_5(x) = \arctan(x)$
6. $f_6(x) = \frac{\sin(x)}{x}$
7. $f_7(x) = \frac{1}{\ln(x)}$
8. $f_8(x) = e^{e^x}$
9. $f_9(x) = \ln(\ln(x))$
10. $f_{10}(x) = \sin(\ln(x))$
11. $f_{11}(x) = 2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$
12. For Assignment 4 see sampleFunction.*

**Assignment 1 (30pt):**

Check comments in Assignment1.py.

Implement the function **Assignment1.interpolate(..)**.

The function will receive a function f, a range, and a number of points to use.

The function will return another "interpolated" function g. During testing, g will be called with various floats x to test for the interpolation errors.

<u>Grading policy:</u>

Running time complexity > O(n^2): 0-20%

Running time complexity = O(n^2): 20-80%

Running time complexity = O(n): 50-100%

The grade within the above ranges is a function of the average absolute error of the interpolation function at random test points. Correctly implemented linear splines will give you 50% of the assignment value.

Solutions will be tested with $n \in \{1,10,20,50,100\}$ on variety of functions at least half of which are polynomials of various degrees with coefficients ranging in $[-1,1]$.

**Question 1.1:** Explain the key points in your implementation.

**נקודות המפתח:**

**1.** תחילה השתמשתי בsample functions ע"מ להשתמש בפונקציית bezier וכן מימשתי את Thomas Algorithm.

**2.** אם מספר הנקודות המקסימלי שניתן להשתמש בו הוא 1 זהו מקרה קיצון עבורו לקחתי את הדגימה של הפונקציה להיות $\frac{a+b}{2}$

**3.** בכל מצב אחר, דגמתי n נקודות וחילקתי את הטווח [a,b] למרחקים שווים בהתאם לn הדגימות.

**4.** יצרתי points_lst שהיא למעשה רשימה של הנקודות שדגמתי, שכוללת את ערך האיקס שלהן וערך הy שלהן.

**5.** תוך העזרות במצגת 6 בהרצאה, יצרתי את המטריצה שאראה בהמשך, ע"י שימוש בפונקציית filling_diagonal_lines למילוי האלכסונים המתאימים של המטריצה:

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 4 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 4 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & 4 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 2 & 7 \end{bmatrix}$$

**6.** יצרתי את וקטור K ע"י שימוש בpoints_lst:

$$\begin{bmatrix} K_0 + 2K_1 \\ 4K_1 + 2K_2 \\ 4K_2 + 2K_3 \\ \vdots \\ 4K_{n-3} + 2K_{n-2} \\ 4K_{n-2} + 2K_{n-1} \\ 8K_{n-1} + K_n \end{bmatrix}$$

**7.** יצרתי את וקטור a באמצעות Thomas Algorithm, ע"מ למצוא את האיקסים והy-ים של הוקטור שלחתי בכל פעם לאלגוריתם תומס את האיקסים של וקטור K והy-ים של וקטור K:

$$\begin{cases} 2a_0 + a_1 + \cdots + 0 = K_0 + 2K_1 \\ 0 + \cdots + a_{i-1} + 4a_i + a_{i+1} + \cdots + 0 = 4K_i + 2K_{i+1} \\ 0 + \cdots + 2a_{n-2} + 7a_{n-1} = 8K_{n-1} + K_n \end{cases}$$

**8.** יצרתי את וקטור b ע"פ הנוסחא $b_i = 2K_{i+1} - a_{i+1}$ והנקודה האחרונה התווספה ע"פ הנוסחה: $a_{n-1} - 2b_{n-1} + K_n = 0$

**9.** יצרתי מילון לשמירת התוצאות שחוזרות מפונקציית Bezier (שלחתי לפונקציה זו בכל פעם 2 נקודות מpoints_lst, את וקטור a ואת וקטור b).

**10.** לבסוף, יצרתי פונקציה בשם approx_func שמקבלת ארגומנט בודד. פונקציה זו מוצאת המיקום של הx שקיבלנו בין ה-x-ים שדגמנו, אם נפלנו על x שדגמנו נגדיר ש t=0 אחרת נמצא את המיקום היחסי של הx שקיבלנו על העקומה המתאימה ונחזיר את ערך ה-y המתאים.

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) שימוש ב Cubic Bezier Curve, מאפשר לכך ששגיאת האינטרפולציה מצטמצמת באופן משמעותי בהשוואה לשיטות אינטרפולציה לינארית או פולינומית וכן שיטה זו מדויקת יותר בהשוואה לאינטרפולציה לינארית או פולינומית.

2) השימוש בThomas Algorithm פותר ביעילות מירבית מטריצות תלת אלכסוניות, דבר שמסייע בהקטנת זמני הריצה של תהליך האינטרפולציה

3) גמישות - ניתן להשתמש באלגוריתם זה למגוון פונקציות וטווחים, מה שהופך אותו לפתרון גמיש לבעיות אינטרפולציה. בנוסף, האלגוריתם הנ"ל יכול להתמודד עם פונקציות שאינן ליניאריות ביחס לאלגוריתמים פשוטים יותר.

4) הפונקציה שחוזרת הינה בעלת מעברים חלקים. בניגוד לאינטרפולציה ליניארית, לאינטרפולציה של spline אין מעברים חדים בין מקטע למקטע.

**Assignment 2 (15pt):**

Check comments in Assignment2.py.

Implement the function **Assignment2.intersections(..)**.

The function will receive 2 functions- $f_1$, $f_2$, and a float maxerr.

The function will return an iterable of approximate intersection Xs, such that:

$$\forall x \in X, |f_1(x) - f_2(x)| < maxerr$$

<u>Grading policy:</u> The grade will be affected by the number of correct/incorrect intersection points found and the running time of **Assignment2.intersections(..)**.

**Question 2.1:** Explain the key points in your implementation.

**נקודות המפתח:**

**1.** תחילה יצרתי פונקציה אשר בנויה מהפרש הפונקציות, פונקציה זו תשמש אותי למציאת השורשים שלה שהם למעשה נקודות החיתוך בין 2 הפונקציות. אנו נדרשים במשימה להחזיר את ערך האיקס של נקודות החיתוך הללו ולכן נתייחס למשימה שלנו כמציאת שורשים בפונקציה החדשה שנוצרה.

**2.** יצרתי מערך בו אשמור את ערכי האיקס של נקודות החיתוך בין הפונקציות- final.

**3.** חילקתי את הטווח [a,b] למרחקים שווים, עם 1100 נקודות באמצעות פונקציית linspace של numpy.

**4.** לכל טווח כזה בדקתי אם סימני הגבולות הם שונים מאחר ולפי משפט ערך הביניים פונקציה f(x) היא רציפה בקטע [segment[i], segment[i+1]] אם מתקיים f(segment[i])*f(segment[i+1]) < 0 אז קיימת נקודה $x_i \in (segment[i], segment[i+1])$ כך ש: $f(x_i) = 0$.

**5.** אם אכן סימני הגבולות שונים, נכנס לפונקציה בשם *finding_roots* שלמעשה משתמשת בשיטת ה*bisection*.

**6.** אחרת, נתקדם לטווח הבא.

**הסבר על פונקציית *finding_roots*:**

פונקציה זו מקבלת את פונקציית *func_for_finding_roots* שהיא הפונקציה אשר בנויה מהפרש הפונקציות, בנוסף פונקציה זו מקבלת טווח וכן את השגיאה המקסימלית.
ביטאתי את הטווח השמאלי ב*l_line*, ואת הטווח הימני ב*r_line*. יצרתי נקודה התחלתית *x* שתהווה את אמצעו של מקטע כלשהו. וביצעתי 3 בדיקות. בבדיקה הראשונה בדקתי האם ערך ה*y* בערך מוחלט, של נקודה הנמצאת על הטווח השמאלי, קטנה או שווה מ*maxerr*. אם כן, זה אומר שמצאנו שורש, לכן נוכל להחזיר את ה-*x* של הנקודה שנשלחה לבדיקה. בבדיקה השנייה בדקתי האם ערך ה*y* בערך מוחלט, של נקודה הנמצאת על הטווח הימני, קטנה או שווה מ*maxerr*. אם כן, זה אומר שמצאנו שורש, לכן נוכל להחזיר את ה-*x* של הנקודה שנשלחה לבדיקה. אם שתי הבדיקות הללו לא מתקיימות יצרתי לולאת *while*, שרצה כל עוד הערך המוחלט של הפונקציה בניחוש הנוכחי של השורש קטן מהשגיאה המקסימלית. בכל איטרציה *x* מתעדכן להיות אמצע של המקטע החדש שניצור בריצת הלולאה. אם ערך ה*y* של הניחוש *x* כפול ערך ה*y* של ה*l_line* מניב תוצאה שלילית, זה אומר שעלינו לצמצם את המקטע מימין ולכן נעדכן את המקטע הימני להיות שווה לניחוש הנוכחי. אחרת, זה אומר שעלינו לצמצם את המקטע משמאל ולכן נעדכן את המקטע השמאלי להיות שווה לניחוש הנוכחי. בסיום לולאת ה*while*, נקבל כי הערך המוחלט של הפונקציה בניחוש הנוכחי של השורש קטן מהשגיאה המקסימלית, כלומר מצאנו שורש ונחזיר אותו.

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) אלגוריתם זה יכול להתמודד עם פונקציות מורכבות בעלות שורשים מרובים אשר קשה לפתור אותן בצורה אנליטית. בנוסף, אלגוריתם זה יכול להתמודד עם פונקציות שיש להן נגזרות שקרובות לאפס, מה שיגרום לשיטות מציאת שורשים אחרות כמו Newton Raphson לעבוד ביעילות נמוכה יותר.

2) שיטת ה*bisection* היא אלגוריתם פשוט שקל להבין וליישם, שיטה זו דורשת רק פעולות חשבון בסיסיות והשוואות.

3) שיטת ה*bisection* פחות רגישה לניחוש הראשוני משיטות אחרות, כמו שיטת *Newton Raphson*, שיכולה להתכנס לשורש הלא נכון או להתפצל אם הניחוש הראשוני אינו קרוב מספיק לשורש בפועל.

**Assignment 3 (25pt):**

Check comments in Assignment3.py.

Implement a function **Assignment3.integrate(…)** and **Assignment3.areabetween(..)** and answer two theoretical questions.

**Assignment3.integrate(…)** receives a function f, a range, and several points to use.

It must return approximation to the integral of the function f in the given range.

You may call f at most n times.

<u>Grading policy:</u> The grade is affected by the integration error only, provided reasonable running time e.g., no more than 5 minutes for n=100.

**Question 3.1:** Explain the key points in your implementation of Assignment3.integrate(…).

**נקודות המפתח:**

בחלק זה, השתמשתי בSimpson's Rule. אם n זוגי, הקטנתי את n ב1 על מנת להפוך אותו לאי זוגי לאחר מכן, מצאתי את height שהוא הרוחב של כל תת מרווח של האינטגרציה ע"י הנוסחה $h = \frac{b-a}{n-1}$.

יצרתי משתנה x_points שהוא מערך של נקודות מרווחות באופן שווה בין a ל-b עם n נקודות, באמצעות פונקציית linspace של numpy. וכן, יצרתי משתנה y_points שהוא מערך של ערך הפונקציה בכל x_point ע"י קריאה לפונקציה f עם ה-x_points.

לבסוף, ביצעתי את קירוב השטח מתחת לעקומה על ידי התאמת מקטעים לפונקציה על פני כל זוג של תת-מרווחים סמוכים תוך שימוש בנוסחה:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = \frac{h}{3} * \left(f(x_{i-1}) + 4 * f(x_i) + f(x_{i+1})\right)$$

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) ל*Simpson's Rule* יש דרגת דיוק גבוהה יותר בהשוואה לטכניקות אינטגרציה מספריות אחרות, כמו *Trapezoidal Rule* או *Mid point Rule*.

2) זהו אלגוריתם פשוט וקל ליישום בהשוואה לשיטות אינטגרציה מספריות אחרות. הוא מבוסס על נוסחה פשוטה שניתן לתכנת בקלות.

3) *Simpson's Rule* מדויק עבור פונקציות מעוקבות, פונקציות ריבועיות ופונקציות לינאריות, כלומר ניתן להשתמש בו על טווח רחב יחסית של פונקציות.

**Assignment3.areabetween(..)** receives two functions $f_1, f_2$ .

It must return the area between $f_1, f_2$ .

In order to correctly solve this assignment you will have to find all intersection points between the two functions. You may ignore all intersection points outside the range $x \in [1,100]$.

Note: there is no such thing as negative "area".

<u>Grading policy:</u> The assignment will be graded according to the integration error and running time.

**Question 3.2:** Explain the key points in your implementation of Assignment3. areabetween (…).

**נקודות המפתח:**

**1.** תחילה יצרתי פונקציה חדשה, שהיא חיסור של f1 וf2 תוך שימוש בפונקציית lambda, מאחר ואם אמצא את השטח של פונקציית החיסור, זה למעשה יהיה שקול למציאת השטח הכלוא בין 2 הפונקציות הנתונות בטווח [1,100].

**2.** יצרתי מערך של כל נקודות החיתוך (ערכי הx) של פונקציות f1, f2 ע"י שימוש במטלה 2. שמרתי את התוצאה במשתנה intersection_points.

**3.** הגדרתי תנאי עבורו, אם אורך המערך של Intersection_points קטן מ2, החזרתי np.Nan

**4.** לאחר מכן, הגדרתי counter לסכימת תוצאות השטחים. הסכימה תתבצע ע"י לולאת for שבכל איטרציה יתבצע counter = counter + התוצאה החוזרת מפונקציית integrate בערך מוחלט (כשבכל פעם אנו שולחים כטווח נקודה ואת הנקודה העוקבת מתוך intersection_points). כאשר לבסוף התוצאה שתחזור מהcounter תהווה את השטח המבוקש.

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) האלגוריתם אינו דורש חישובים מורכבים או מספר רב של הערכות פונקציות, כך שהוא יכול להיות מהיר בהרבה משיטות אינטגרציה מסובכות אחרות.

2) ניתן להשתמש באלגוריתם כדי למצוא את השטח בין כל שתי פונקציות מה שהופך אותו לכלי גמיש מאוד עבור מגוון רחב של פונקציות

3) האלגוריתם הינו פשוט ואינטואיטיבי שמהווה רעיון בסיסי ביותר של מציאת השטח בין שתי עקומות. הוא פשוט וקל להבנה ובעיקר קל למימוש.

**Question 3.3:** Explain why is the function $2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$ is difficult for numeric integration with equally spaced points?

לאחר שהצבתי את הפונקציה הנתונה באתר ליצירת פונקציות על מנת להבין את נראות הפונקציה ולהסיק ממנה מסקנות. נוכל לראות כי פונקציה זו קשה לאינטגרציה מספרית עם נקודות מרווחות באופן שווה כיוון שבאזור בו x=0 הפונקציה הופכת לגדולה יותר עד כדי מצב של אינסוף או בלתי מוגדרות. כלומר, הפונקציה לא מוגדרת בנקודה הזאת וגם ככל שנתקרב לx=0 נוכל לראות שהערך של הפונקציה הופך להיות גדול מאוד וכן ישנם מקטעים בהם הערך קטן. זה גורם לשיטות אינטגרציה מספריות המשתמשות בנקודות מרווחות באופן שווה, להיות מאוד לא מדויקות באזור בו x=0 מכיוון שהפונקציה משתנה מאוד מהר באזור זה

**Question 3.4:** What is the maximal integration error of the $2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$ in the range [0.1, 10]? Explain.

כדי לחשב את השגיאה המקסימלית, נשתמש בנוסחה הבאה:

$$|E| \leq \frac{(b-a)^5}{180 * n^4} * (\max\left|f^{(4)}(x)\right|)$$

לאחר שהצבתי את הנגזרת הרביעית של הפונקציה הנתונה באתר ליצירת פונקציות על מנת להבין את נראות הפונקציה ולהסיק ממנה מסקנות, נוכל לראות כי על מנת שנרצה להגיע לשגיאה מקסימלית, נרצה לקבל כמות של n מקטעים קטנה ככל האפשר. בנוסף, הנגזרת הרביעית של הפונקציה וכן המקסימום של נגזרת זו שואפת לאינסוף. לכן עבור a = 0.1, b = 10, n = 2 נקבל:

$$|E| \leq \frac{(10-0.1)^5}{180 * 2^4} * \left(max\left|f^{(4)}(x)\right|\right) = \frac{(9.9)^5}{180 * 16} * \left(max\left|f^{(4)}(x)\right|\right) \rightarrow \infty$$

מכאן ששגיאת האינטגרציה המקסימלית שואפת לאינסוף.

c**Assignment 4 (20pt)**

Check comments in Assignment4.py.

Implement the function **Assignment4.fit(…)**

The function will receive an input function that returns noisy results. The noise is normally distributed.

Assignment4A.fit should return a function $g$ fitting the data sampled from the noisy function. Use least squares fitting such that $g$ will exactly match the clean (not noisy) version of the given function.

To aid in the fitting process the arguments $a$ and $b$ signify the range of the sampling. The argument $d$ is the expected degree of a polynomial that would match the clean (not noisy) version of the given function.

You have no constrains on the number of invocation of the noisy function but the maximal running time is limited. Additional parameter to **Assignment4.fit** is maxtime representing the maximum allowed runtime of the function, if the function will execute more than the given amount of time, the grade will be significantly reduced.

Grading policy: the grade is affected by the error between $g$ (that you return) and the clean (not noisy) version of the given function, much like in Assignment1. 65% of the test cases for grading will be polynomials with degree up to 3, with the correct degree specified by $d$. 30% will be polynomials of degrees 4-12, with the correct degree specified by $d$. 5% will be non-polynomials

**Question 4.1:** Explain the key points in your implementation.

**נקודות המפתח:**

בחלק זה של העבודה השתמשתי בLeast squares Bezier fit. תוך העזרות במצגת 6 בהרצאות - "splines and bezier".

**1.** תחילה הגדרתי את זמן ההתחלה שזהו הstart_samp. לאחר מכן, הוספתי לsamp את הערך של הפונקציה בנקודה a – ע"מ על מנת לחשב את הזמן שלוקח לבצע דגימה מתוך הפונקציה. ולבסוף הגדרתי את זמן הסיום שזהו הend_samp. החסרתי מend_samp את start_samp. אם הזמן הזה קטן או שווה מ0.0001 שניות, אדגום 10,000 נקודות. אחרת, אדגום:

int(abs(maxtime – (maxtime / 5)) / time_to_samp) – 1)) נקודות - מספר הדגימות מחושב על סמך הזמן שנדרש לדגימת הפונקציה והזמן המרבי המותר, זאת על מנת לא לחרוג מזמני הריצה.

**2.** יצרתי את רשימת ה-x-ים של הדגימות וכן את רשימת ה-y-ים של הדגימות

**3.** יצרתי מערך ריק ע"י שימוש בnp.empty בשם upper להחזקת החזקות שמציינות את הדרגה של הפונקציה והכנסתי לתוכו את הערכים בטווח [0, 3], כלומר מערך החזקות הינו-[0, 1, 2, 3].

**4.** יצרתי מערך דו מימדי בשם matrix שמחזיק את הערכים הנ"ל:

$$\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

**5.** מצאתי את המטריצה ההופכית של matrix תוך שימוש בnp.linalg.inv שתשמש אותי בהמשך.

**6.** יצרתי מערך דו מימדי A ריק, בגודל 4 × samples. רצתי בלולאה על כל הדגימות, כאשר עבור כל נקודה ברשימת ה-x-ים, ביצעתי עבורה נרמול כלומר המרתי את מיקום הנקודה במרווח [a, b] למיקום במרווח [0,1]. זאת מאחר ואז נוכל לבטא את פונקציות בסיס הspline, מה שמקל על חישוב המקדמים. לאחר מכן, העליתי בחזקה את נקודת הx, עבור כל חזקה במערך הupper, זאת על מנת ליצור את המטריצה הבאה:

$$\begin{bmatrix} z_1^3 & z_1^2 & z_1 & 1 \\ z_2^3 & z_2^2 & z_2 & 1 \\ & \dots & & \\ z_n^3 & z_n^2 & z_n & 1 \end{bmatrix}$$

**7.** על מנת לבטא את מטריצה B (היא למעשה מבטאת את הcontrol points) שמורכבת מהכפלת המטריצות הבאה:

$$B = M^{-1} * (A^T * A)^{-1} * T^T * y\_points$$

מצאתי את הtransposed matrix של A, תוך שימוש ב-A.T. וכן מצאתי את מכפלת המטריצה $A$ ו$A^T$ ע"י שימוש ב*np.dot* וע"מ למצוא את ההיפוך השתמשתי ב*np.linalg.inv*. וכן הכפלתי בין המטריצות שמצאתי ע"י *np.dot* ע"מ למצוא את מטריצת *B*.

**8.** בשלב האחרון השתמשתי בפונקציה, approx_func, אשר מקבלת ארגומנט בודד. הפונקציה ממירה תחילה את הערך num לערך z, שהוא מיקום מנורמל בין [0,1] המתאים למקום שבו num נופל בטווח [a, b]. לבסוף, החזרתי את $A \times M \times B$ שמבטא את ערך ה-*y* המקורב בהינתן *x* כלשהו. למעשה פונקציית *approx_func*, משתמשת במקדמים המחושבים מנקודות הנתונים שנדגמו כדי לבצע אינטרפולציה של פונקציה פולינומית המקרבת את הפונקציה המקורית.

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) לאלגוריתם זה יש עלות חישובית נמוכה. הוא נמנע משימוש בטכניקות אופטימיזציה יקרות, מה שהופך אותו למהיר ויעיל יותר. בנוסף, הוא מאפשר לחשב את המקדמים של הפולינום ישירות ללא צורך באופטימיזציה איטרטיבית מסובכת.

2) האלגוריתם יכול לעבוד עם נתונים שנדגמים מפונקציות שרירותיות. באמצעות מודל פולינום, האלגוריתם יכול להתאים את עצמו לצורת הפונקציה הבסיסית, מה שהופך אותו לאלגוריתם שאינו מוגבל לסוג ספציפי של פונקציה וניתן להשתמש בו עבור מגוון רחב של פונקציות.

3) האלגוריתם משתמש באלגברה לינארית, מה שיכול לגרום להתכנסות מהירה בהשוואה לאלגוריתמי אופטימיזציה למשל מבוססי גרדיאנט

**Assignment 5 (10pt + bonus 20pt).**

Check comments in Assignment5.py.

Implement the function **Assignment5.area(…)**

The function will receive a shape contour and should return the approximate area of the shape. Contour can be sampled by calling with the desired number of points on the contour as an argument. The points are roughly equally spaced.

Naturally, the more points you request from the contour the more accurately you can compute the area. Your error will converge to zero for large $n$. You can assume that 10,000 points are sufficient to precisely compute the shape area. Your challenge is stopping earlier than according to the desired error in order to save running time.

<u>Grading policy:</u> the grade is affected by your running time.

**Question 4B.1:** Explain the key points in your implementation.

בפונקציה זו אנו נדרשים למצוא את שטח הצורה ע"י קו מתאר נתון. תחילה הגדרתי flag שמאותחל לערך true, 35 נקודות ומילון ריק. יצרתי לולאה שרצה כל עוד לא חוזרת תשובה מתוך הלולאה. בלולאה שלחתי את 35 הנקודות לcontour ע"מ לקבל דגימה של נקודות מתוך קו המתאר. הכנסתי את ערכי קואורדינאטת הx למשתנה וכן את ערכי קואורדינאטת הy למשתנה. לאחר מכן, השתמשתי בshoelace method שטוענת שניתן לחשב את שטחו של מצולע על ידי לקיחת סכום המכפלה של קואורדינטות ה-x וקואורדינטות ה-y שלו עבור קודקודים עוקבים, ולאחר מכן לחלק את התוצאה ב2. S1 ו-S2 הן תוצאות ביניים שמשמשות לחישוב השטח של הקטע. S1 מחושב כמכפלה (ע"י שימוש בnp.dot) של רשימת קואורדינטות ה-x והרשימה המוסטת של קואורדינטות y (הרשימה המוסטת מחושבת ע"י שימוש בnp.roll), ו-S2 מחושב כמכפלה של רשימת קואורדינטות ה-y והרשימה המוסטת של קואורדינטות x (הרשימה המוסטת מחושבת ע"י שימוש בnp.roll). לבסוף, הכנסתי למילון במפתח הi את תוצאת המתודה. הגדלתי את הi ב1 על מנת להכניס את התוצאה הבאה היותר מדויקת למילון זאת ע"י הגדלת כמות הנקודות על מנת להיכנס ללולאה בשנית ולקבל תוצאות יותר מדויקות. בנוסף, בתחילת הכניסה ללולאה, ביצעתי בדיקה האם הi שלי גדול מ1, כלומר האם מספר הערכים שלי במילון יותר גדול מ1 כדי לבצע בדיקה האם התשובה הנוכחית שאליה הגעתי למול התשובה הקודמת אליה הגעתי נותנת לי שגיאה שהינה יותר קטנה מmaxerr (כלומר מדובר כאן בחישוב הrelative error ווידוא שהשגיאה הזו קטנה יותר מmaxerr), אם כן, זה אומר שתוצאת השטח שהגעתי אליה עד כה היא הכי מדויקת וארצה להחזיר אותה.

למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:

1) האלגוריתם מגדיל באופן דינמי את מספר הנקודות המשמשות לקירוב הצורה, מה שמאפשר דיוק רב יותר בחישוב השטח. האלגוריתם ממשיך להגדיל את מספר הנקודות עד שהשגיאה בחישוב השטח קטנה מ-maxerr. המשמעות היא שהאלגוריתם יכול להתכנס במהירות לתוצאה, ולצמצם את זמן החישוב הכולל הדרוש

2) ניתן להשתמש בshoelace method על מגוון רחב של צורות, מצורות מצולע פשוטות ועד לצורות מורכבות יותר כמו עקומות וצורות לא סדירות, מה שהופך את השיטה הזו לרבת שימוש על מגוון של צורות.

Implement the function **Assignment4.fit_shape(…)** and the class **MyShape**

The function will receive a generator (a function that when called), will return a point (tuple) (x,y), a that is close to the shape contour.

Assume the sampling method might be noisy- meaning there might be errors in the sampling.

The function will return an object which extends **AbstractShape**
When calling the function **AbstractShape.contour(n)**, the return value should be array of n equally spaced points (tuples of x,y).

Additional parameter to **Assignment4.fit_shape** is maxtime representing the maximum allowed runtime of the function, if the function will execute more than the given amount of time, the grade will be significantly reduced.

In this assignment only, you may use any numeric optimization libraries and tools. Reflection is not allowed.

Grading policy: the grade is affected by the error of the area function of the shape returned by Assignment4.fit_shape.

**Question 4B.2:** Explain the key points in your implementation.

במימוש הפונקציה הזו השתמשתי בk means. אתחלתי 25 k means (חילוק 25 נקודות ל25 אשכולות) ודגימה של 2500 נקודות מתוך הcontour. לאחר מכן, השתמש בפונקציית fit על מנת לאמן את המודל שלי ושלפתי את מרכזי האשכולות. יצרתי 2 מערכים, מערך אחד

שמחזיק את ערכי קואורדינאטת הx של המרכזים שמצאתי ומערך שני שמחזיק את ערכי קואורדינאטת הy של המרכזים שמצאתי. לאחר מכן, מיינתי את נקודות המרכז תוך שימוש בפונקציית angle_distance שמטרתה להחזיר את הזווית השלילית בין שתי נקודות (השליליות מטרתה למיין את הנקודות בכיוון השעון מה שמבטיח שקו המתאר של הצורה יתואר בסדר הנכון) וכן את המרחק בין שתי הנקודות. למעשה מתבצע מיון של נקודות המרכז בהתבסס על מרחק הזווית של כל נקודה מהערך המינימלי של קואורדינאטת x ומהערך הממוצע של קואורדינאטת y. ולבסוף, שלחתי את הנקודות הממוינות לפונקציית My Shape.

<u>למה לדעתי האלגוריתם שלי יעיל וייחודי יותר?:</u>

1) השימוש במספר k-means מוגבל, ומציאת המרכז שלהן, מביא להפחתה משמעותית בכמות הנתונים שיש לעבד. בנוסף, הדבר מפחית את מספר האיטרציות הנדרשות כדי להתאים את המודל לנתונים – מה שמביא לחישוב מהיר ויעיל יותר

2) הפונקציה angle_distance אשר מחשבת את הזווית בין שתי נקודות והמרחק שלהן והיעזרות בה על מנת למיין את מרכזי האשכול לפי סדר הזוויות שלהם עוזר ליצור קו מתאר חלק יותר של הצורה באמצעות נקודות הנתונים הרועשות שנדגמו

3) השימוש בk – means עוזר לאסוף ביעילות את נקודות הנתונים הרועשות, ומפחית את המורכבות של מציאת הפתרון