Numerical Analysis

Final task

Submission date: 15/2/2023 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 (14pt):

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 (10pt):</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 (4pt).

<u>הרעיון הכללי –</u> מימוש שיטת ניוטון לאינטרפולציה כאשר נתונה לי הפונקציה המקורית וטווח.

<u>מימוש –</u>

1. פונקציית ה interpolate – תחילה מרווחת n נקודות על פי שיטת ה – Chebyshev בטווח הנתון.
2. לאחר קבלת סט הנקודות חישוב ערכי הפונקציה בכל אחד מן הנקודות.
3. שליחת הנקודות אל פונקצית ה newton_polynomial .
4. פונקצית ה newton_polynomial משתמשת בפונקצית ה coefficient אשר מחשבת את מקדמי הפולינום ומחזירה אותם אל הפונקציה.
5. לאחר מכן הפונקציה בונה את הפולינום על פי שיטת Hermite לבניית פולינומים בזמן ליניארי ומחזירה אותו.
6. השלב האחרון הוא בניית פונקציית lambda שמקבלת x ומחזירה את הפולינום המתקבל מפונקציית הnewton_polynomial .
7. החזרה של פונקציית הlambda כתוצאה של האינטרפולציה

Assignment 2 (14pt):

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 (10pt):</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 (4pt).

<u>הרעיון הכללי –</u> קבלת 2 פונקציות וחישוב כל נקודות החיתוך שלהם בטווח מסויים עם שגיאה אשר אנו מקבלים.

<u>מימוש –</u>

1. פונקציית ה intersections – תחילה מחסרת בין הפונקציות הנתונות בכדי ליצור פונקציה חדשה.
2. לאחר מכן נשלח את הטווח , הפונקציה והשגיאה אל פונקציה נפרדת הנקראת area_finder אשר מחזירה לי שטחים שבהם בוודאות יש חיתוך בין הפונקציות, נמצא שטח זה בעזרת חוק ערך הביניים.
3. ערך הביניים – אומר שכאשר יש 2 ערכים שהמכפלה שלהם ביחד קטנה מאפס , בוודאות בין 2 ערכים אלו יש חיתוך עם ציר ה- x.
4. חיתוך בין הפונקציות מתקיים אשר לפונקציית חיסור בניהם יש נקודת חיתוך עם ציר ה-x או בטווח של השגיאה הרצויה.
5. לאחר שנמצא טווח אשר בו בוודאות יש חיתוך לפונקציות , נשלח את הטווח עם הפונקציה לפונקציית ה Regula_Falsi .
6. פונקציית ה Regula_Falsi ממשת את שיטה זאת למציאת שורש של פונקציה, באמצעות הקטנת הטווח על ידי משוואה עד הגעה לטווח של השגיאה הרצויה.
7. לאחר קבלת ערכים מן הפונקציה נכניס אותם לרשימה ונחזיר אותה

Assignment 3 (36pt):

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.

Grading policy (10pt): 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(…). (4pt)

הרעיון הכללי – קבלת פונקציה וטווח אשר נצטרך לחשב את שטח הפונקציה (האינטגרל) בטווח זה, בנוסף אנו מקבלים מגבלה אשר אנו מקבלים מספר n ולנו מותר לדגום את הפונקציה רק n פעמים.

מימוש –

1. תחילה נבדוק האם הn שקיבלנו הוא זוגי או לא.
2. במידה והוא זוגי נשלח לפונקציה הנקראת Simpson את הפונקציה טווח ו n - 2 נקודות שמותר לו לדגום.
3. במידה והוא לא זוגי נשלח לפונקציה n – 1 נקודות אשר מותר לו לדגום בגלל ההגבלה שקיבלנו.
4. בפונקציית הSimpson נחשב שטח של הפונקציה בטווח הנתון בעזרת שיטת הSimpson לאינטגרציה נומרית.
5. שיטת הSimpson – שיטה לחישוב אינטגרלים נומריים שבה מחלקים את הפונקציה למספר זוגי של חלקים פרבוליים,ומחשבים את השטחים האלו.
6. לאחר שנסיים לחשב את שטח הפונקציה בטווח הנתון נמיר אותו לסוג np.float32 כנדרש ונחזיר.

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”.

Grading policy (10pt): 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 (…) (4pt).

הרעיון הכללי – קבלת 2 פונקציות אשר צריך למצוא בניהם שטח (איטנגרל) בטווח בין 1 ל100.

מימוש –

1. תחילה נייבא את משימה 2 אשר ביצענו בפרוייקט.

2. לאחר מכן נמצא את נקודות החיתוך של 2 הפונקציות יחדיו בטווח בין 1 ל100, עם שגיאה של 0.001 (ערך ברירת המחדל של הפונקציה).
3. לאחר שמצאנו את כלל נקודות החיתוך של הפונקציות נבדוק האם ישנם נקודות , במידה ואין נקודות נחזיר את המספר 0 מכיוון שאין בין הפוונקציות שטח מוגדר בטווח.
4. לאחר מכן נבדוק האם יש פחות נקודות חיתוך מ – 2 כי זה המינימום הדרוש לחישוב שטח, ואם אין נוסיף את אחד הקצוות.
5. במידה ויש מעל 2 נקודות חיתוך (כולל 2) , נרוץ בלולאת for על כל החיתוכים שיצאו ונבדוק בטווח שבין 2 החיתוכים איזה פונקציה נמצאת למעלה בטווח זה.
6. נבדוק בתנאי איזה פונקציה יותר גבוהה ונשלח לפונקציה שבנינו לאינטגרציה בסעיף א' פונקציה חדשה שהיא חיסור בין הפונקציות אשר הפונקציה הגבוהה יותר היא הפונקציה שמחסרים ממנה
7. נשלח לפונקציית אינטגרציה מסעיף א' את הפונקציה החדשה את נקודות החיתוך שכרגע אנו רוצים עליהם ביחד עם 500 נקודות בכדי לקבל את הדיוק המרבי.
8. נסכום את כלל השטחים שאנו מקבלים בערך מוחלט ונחזיר אותם במשתנה חדש מסוג np.float32 כנדרש.

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? (4pt)

<u>הסבר –</u> הפונקציה קשה לאינטגרציה נומרית עם נקודות מרווחות באופן שווה מכיוון שיש לה נקודה סינגולרית ב x = 0.
כלל ש x מתקרב ל 0, הפונקציה מתנדנדת מהר יותר ויותר, המשרעת גדלה ללא גבול. דבר זה מאתגר להעריך במדויק את התנהגות הפונקציה באמצעות מספר סופי של נקודות דגימה בריווח שווה.
בכלל שיטות האינטגרציה הנומריות דורשות בדרך כלל שהפונקציה תתנהג בצורה טובה וחלקה בתחום האינטגרציה , אך פונקציה זאת מפרה את ההנחות האלו ליד x = 0.
כתוצאה מכך, הדיוק של כל האינטגרציה הנומרית יפגע והשגיאה עשויה להיות גדולה באופן חריג.

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. (4pt)

<u>הסבר –</u> מבין שיטות האינטגרציה Simpson rule מבצע קירוב לפולינום ממעלה שנייה ולכן עבורות נקבל את התוצאה הטובה ביותר.
השגיאה המקסימלית תתקבל כאשר n = 2 , מכיוון שנבצע שיטה זאת רק בין 2 נקודות בלבד, כלומר מתחילת המקטע ועד סופו.
כפי שלמדנו בכיתה השגיאה המתקבלת על ידי סימפסון מוגדרת לפי הנוסחא - $-\frac{1}{90}\left(\frac{b-a}{2}\right)f^4(\xi)$
כפי שהבנו מהסעיף הקודם הפונקציה בעלת תנודות רבות בקירוב לראשית הצירים כך שבכל תנודה הפונקציה שואפת לאינסוף.
על מנת למקסם את השגיאה נבחר במספר הכי קרוב לראשית הצירים בטווח שלנו שהוא 0.1 ולכן נציב $\xi = 1$.
הנגזרת הרביעית של הפונקציה בנקודה זאת - $f^4(0.1) = -4.0e^{42}$.
נציב את הכל בנוסחא ונקבל שהשגיאה שווה ל – $1.3208 * 10^{44}$.

Assignment 4 (14pt)

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.

<u>Grading policy (10pt):</u> 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. (4pt)

<u>הרעיון הכללי –</u> קבלת פונקציה רועשת טווח ודרגה והתאמת פולינום מקורב לפונקציה הרועשת, בנוסף נקבל maxtime אשר נותן לנו זמן מקסימלי לביצוע המשימה.

<u>מימוש –</u>

1. תחילה נדגום את הפונקציה בשביל לקבל זמן נתון לדגימה.
2. לאחר מכן נחשב בעזרת הדגימה של הזמן את כמות הנקודות אותן נדגום מן הפונקציה.
3. לאחר קבלת כמות הנקודות נחלק את הטווח שקיבלנו במרווחים שונים לפי כמות הנקודות שקיבלנו מהדגימה.
4. נדגום את הפונקציה שלנו לפי נקודות ה -x שקיבלנו.
5. ונשלח את כלל הנקודות לפונקציה הנקראת Create_matrix אשר תבנה לי את המטריצה לפי שיטת הleast squares.
6. לאחר בניית הפונקציה נבנה את וקטור הפתרונות בעזרת הפונקציה answer_matrix .
7. ניצור 2 מטריצות – אחת עליונה ואחת תחתונה מן המטריצה הראשית שבנינו, נפרק מטריצות אלה בעזרת שיטת ה lu decomposition והפונקציה המיועדת שביניתי לכך.
8. לאח מכן בעזרת שיטת גאוס נפתור את מערכת המשוואות הנוצרת מן המטריצה העליונה ווקטור הפתרונות ונקבל ווקטור חדש.
9. בעזרת הווקטור החדש והמטריצה התחתונה נפתור גם כן את מערכת המשוואות ונקבל מערך של מקדמים של הפולינום החדש שאנו רוצים לבנות.
10. ניקח את וקטור המקדמים ונשתמש בפונקצית np.poly1d(np.flip(coefficient)) אשר יוצרת לנו פולינום מן המקדמים.
11. נחזיר את הפונקציה שקיבלנו מן המקדמים וזאת תהיה פונקציית הקירוב של הפונקציה הרועשת המקורית.

Assignment 5 (27pt).

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 (9pt):</u> the grade is affected by your running time.

Question 4B.1: Explain the key points in your implementation. (4pt)

<u>הרעיון הכללי –</u> חישוב שטח של צורה סגורה בעזרת קבלת contour של הצורה ושגיאה מקסימלית של השטח.

<u>מימוש –</u>

1. תחילה ניצור משתנה שיחזיק לנו את כמות הנקודות שנרצה לדגום מן הצורה, נקרא למשתנה n ותחילה נגדיר אותו 1000.
2. נדגום את הפונקציה n פעמים.
3. לאחר מכן בעזרת הנקודות שלנו נשתמש באלגוריתם Shoelace בכדי לחשב את שטח הצורה.
4. אלגוריתם Shoelace הוא אלגוריתם המשמש לחישוב שטח מצולע, עובד על ידי לקיחת נקודות מן המצולע וחישוב השטח על ידי מכפלת קרובה של מספר זוגות , וסכום המכפלות נמוך מתוך מכפלות אחרות של זוגות הנקודות, המכפלה עצמה מתחשבת בכיוון הסיבוב של הנקודות.
5. לאחר חישוב השטח הראשוני נרוץ בלולאת while תוך כדי שאנו מגדילים את מספר הנקודות של הדגימה.
6. בכל ריצה של הלולאה אנו נגדיל את מספר הנקודות ונחשב את השטח מחדש.
7. הלולאה תעצור כאשר דגמנו מעל 10000 נקודות או כאשר מצאנו שטח שההפרש בינו לבין השטח הראשוני קטן או שווה לשטח שמצאנו.
8. נחזיר את השטח החדש בתוך משתנה של np.float32 כנדרש.

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 (10pt): 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. (4pt)

הרעיון הכללי – קבלת גנרטור שנקרא sample אשר דוגם נקודות מן צורה ומחזיר את הנקודות בצורת גנרטור, וקבלת זמן מקסימלי לביצוע המשימה , המשימה מחזירה משתנה מסוג myshape שמותאם לנקודות אשר קיבלנו שיודע לחשב שטח של צורה.

מימוש –

1. תחילה ניצור משתנה שיחזיק לנו את כמות הנקודות אשר אנו דוגמים מן הגנרטור, נקרא למשתנה n ונגדיר אותו להיות 100000.
2. לאחר מכן ניצור משתנה שיזחזיק לנו את כלל הנקודות אשר אנו דוגמים.
3. נסדר את הנקודות לפי שיטת הclockwise.
4. שיטת הclockwise היא שיטה אשר מוצאת את הנקודה האמצעית של הצורה בעזרת ממוצע של כל הנקודות וסידור הנקודות בצורת "שעון" סביב נקודת האמצע.
5. לאחר קבלת הנקודות באופן מסודר נשתמש בחבילת scipy.splprep אשר לוקחת את הנקודות ומעבירה דרכם עקומה.
6. ניקח סט חדש של נקודות המרוחקות באופן שווה מן משתנה שקיבלנו מהעקומה.
7. ולאחר מכן נשתמש בsplev שגם הור חלק מחבילת scipy אשר הופכת לי את העקומה לקו ישר אשר מחבר את כל הנקודות ויוצר צורה נקייה מן הנקודות.
8. ניצור משתנה חדש מסוג MyShape ונשלח לו את הפרטמטרים של הנקודות החדשות.
9. בתוך הclass של MyShape ניצור לו תכונות אשר מכילות את ערכי נקודות הx של הצורה וערכי נקודות הy.
10. בנוסף ניצור לclass פונקציה חדשה אשר נקראת area אשר תחשב את שטח הצורה מן הנקודות שלה בעזרת שיטת הShoelace .
11. ונחזיר את המשתנה החדש שיצרנו.