Numerical Analysis

Final task

Submission date: 22/2/22 23:59 (strict).

This task is individual. No collaboration is allowed. Plagiarism 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.

You should not use those parts of the libraries that implement numerical methods taught in this course (unless explicitly stated otherwise in the instructions of the particular assignment)**.** This restriction 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 deduction of points. Failure to announce the use of any restricted functions will result in disqualification of the assignment.

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

(not studied in class) numpy.linalg.cholesky, torch.cholesky, linalg.qr, torch.qr (10% 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)

~~TBD – restriction of~~ numeric differentiation functions are allowed!

numpy.linalg.inv, scipy.linalg.inv, torch.inverse,

and all other external libraries for matrix inversion (20% of the assignment score)

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

You must not use reflection (self-modifying or self-inspecting 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. BUT! existing unit tests are provided for demonstration and to encourage you to write additional tests as you go. You can add any number of additional unittests to ensure correctness of your implementation. Passing only the existing unittests does not ensure that your code will not fail in all cases. It is your responsibility to test your code and ensure that it is stable. You should add 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.

Formatted: Font: (Default) +Headings (Calibri Light), 9 pt, Complex Script Font: +Headings (Calibri Light), 9 pt

Formatted: Font: (Default) +Headings (Calibri Light), 9 pt, Complex Script Font: +Headings (Calibri Light), 9 pt

Formatted: Font: Bold, Complex Script Font: Bold

Upon the completion of the final task, you should submit the five assignment files and this document with answers to the theoretical questions. The archive should not contain folders, but only the submission files!

Assignments will be graded according to **error** of the numerical solutions and **running time**. Some assignments have required specific error bounds – they will be graded according to running time. Some assignments limit the running time – they will be graded according to error. For all executions there is 2 minutes running time cap after which the execution will be halted.

Every assignment will be AUTOMATICALLY tested on a number of different functions and different parameters. It may be executed multiple times on the same function with the same parameters. Every execution will start with a clean memory. Any exception throwed during an execution will render the execution invalid and nullify its contribution to the grade. **Test your code!!!**

Any disqualification of an assignment (e.g. due to unannounced use of restricted functions) or an execution (e.g. due to exception) will not contribute to the grade regardless the effort put in the development.

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 – add additional unittests with implementations of these functions. At least half of the test functions will be polynomials. Functions 3,8,10,11 will account for at most 5% 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):

(10pt) Implement the function **Assignment1.interpolate(..)** following the pydoc instructions.

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.

Grading policy:

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

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

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

Running time complexity will be measured empirically as a function of n.

The grade within the above ranges is a function of the average relative 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 $n \in \{1,10,20,50,100,200,500,1000\}$ on variety of functions at least half of which are polynomials of various degrees with coefficients ranging in $[-1,1]$.

Restricted functions I used:

(~~5pt~~4pt) **Question 1.1:** Explain the key points in your implementation.

I used Bezier spline interpolation. (For n>1 else I return y=x)

- Firstly, I sample the area n times (using linspace), and for each point sample f.
- Then for every pair of points I Fit a Bezier curve by calculating the control (only y values) points using Thomas algorithm.
- Afterwards I made a list that contains all the Y values (without X values) and returns function g.
- Function g gets an input of x value, searches for the cell that will contain the relevant curve (only need the x from the linspace because Bi(0) = x[i] and Bi(1) = x[i+1])
- After finding the right cell it estimates a value for t (I assume it should be about the same as the ratio of the x value).
- At last I return the value of Bi(estimated_x)

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

Assignment 2 (14pt):

(10pt) Implement the function **Assignment2.intersections(..)** following the pydoc instructions.

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:

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

$$\forall x_i x_j \in X, |x_i - x_j| > maxerr$$

Grading policy: The grade will be affected by the number of correct and incorrect intersection points found, the running time of **itr = Assignment2.intersections(..)** followed by **list(itr).**

Restricted functions I used:

(~~5pt~~4pt) Question 2.1: Explain the key points in your implementation in particular explain how did you address the problem of finding multiple roots.

I used normal bisection, I estimated the intersection to be spaced and not close to each other. And divided the range to 256, estimated it will be enough to find most of the intersections.

Assignment 3 (31pt):

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

(~~5pt~~) Assignment3.integrate(…) receives a function f, a range, and a number of points n.

Formatted: Strikethrough

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: The grade is affected by the integration error only, provided reasonable running time e.g., no more than 2 minutes for n=100.~~

Restricted functions I used:

(~~5pt~~4pt) **Question 3.1:** Explain the key points in your implementation of Assignment3.integrate(…).

I used Simpson's rule for integration in case of more than 2 points if there is an even number, I reduced the number of points by 1. If I have less than 3 points to sample than I either draw a trapeze (two points) or makes a rectangle.

(~~10pt~~15pt) **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: The assignment will be graded according to the integration error and running time.

Restricted functions I used:

(4pt) **Question 3.2:** Explain the key points in your implementation of Assignment3. areabetween (…).

I use the intersection finding from assignment 2, find all intersection points (by bisection). Afterwards for each pair of intersection points I calculate the integral of f1-f2 and adds the absolute value to the result.
In case of less than two intersection points in the range [1:100] I return NAN value as requested.

(**4~~5~~pt) 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?

The function above oscillates with varied frequency and approaches inf and -inf as x approach 0. So if you were to choose a point around 0 the result will probably be a very big number (in means of absolute value), moreover because of the varied frequency you can't chose a range that will represent a cycle (in cycles I mean the repeat of going up and down and not actual cycle) i.e. you will miss cycle as x approach 0 and because of that you wont be able to determine the integral.

(~~5pt~~4pt) **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.

Assuming use of Simpson's rule the max error is :

~~$\max err$~~ $= M * \dfrac{(b-a)^5}{180 * n^4}$

As -M is the maximal absolute value of the 4th derivative of the function $f(x) = 2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$.
a,b are the range of the integration (0.1, 10).
n is the number of subintervals.

The value of the 4th derivative of $f^4(0.1) \approx -4.03147x10^{42} = C$ and this is the absolute max
So: $\max err = |C| * \frac{(10-0.1)^5}{180*n^4}$

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Formatted: Font: Italic

Formatted: Superscript

Formatted: Superscript

Formatted: Complex Script Font: Bold

Formatted: Font: Not Bold, Complex Script Font: Bold

Formatted: Font: Not Bold, Complex Script Font: Bold

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Formatted: Font: Not Bold, Complex Script Font: Not Bold

Assignment 4 (14pt)

(10pt) Implement the function **Assignment4.fit(…)** following the pydoc instructions.

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

Assignment4.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 invocations of the noisy function but the maximal running time is limited. Invocation of f may take some time but will never take longer than 0.5 sec.

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 execution will not contribute to the grade causing significant deduction. You should consider the risk of failure vs gains in accuracy when you get close to the time limit.

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. 60% 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. 10% will be non-polynomials with random $d \in [1..12]$.

Restricted functions I used:

Matrix inverse

(~~5pt~~4pt) Question 4.1: Explain the key points in your implementation.

- After a few trials I realized that 5000*max time /d is a gets me a good precision and not to long running time so I decided to keep it although it doesn't use the whole running time.
- Firstly I made a coefficient (Vandermonde) matrix (A) as shown in class
- I used matrix inverse$(A * At)^{-1}$ in order to solve the equation and get the coefficients of the polynomial.
- At last I return a np.poly1d with the coefficients that I got.

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

Assignment 5 (27pt).

(9pt) Implement the function **Assignment5.area(…)** following the pydoc instructions.

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 that according to the desired error in order to save running time.

Grading policy: the grade is affected by your running time.

Restricted functions I used:

(~~5pt~~4pt) Question 5.1: Explain the key points in your implementation.

First I took 512 samples with contour than I calculated the area of the by adding and subtracting areas of trapezes according to the value of the samples. (x1<x2 add, x2<x1 sub).

(10pt) Implement the function **Assignment5.fit_shape(…)** and the class **MyShape** following the pydoc instructions.

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 should 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). When calling the function **AbstractShape.area()**, the return value should be the area of the shape. You may use your solution to **Assignment5.area** to implement the area function.

Additional parameter to **Assignment5.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 execution will be halted.

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.

There are no restricted functions. The use of any library is allowed.

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

- First, I calculated a reference point by the mean of all points (in most cases should be the middle).
- afterwards I sorted the points by the angle that each line that cross the point and ref makes with the x axis (using arctg and distinguishing between quarters)-.
- I divided the sorted to 500 groups (by the degree) and for each group calculated the mean point.
- Then I calculated the area using the points as points like those from contour.

Formatted: Font: (Default) +Body (Calibri), 11 pt, Font color: Auto, Complex Script Font: 11 pt, Pattern: Clear

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm