Numerical Analysis

Course Project

This project is individual. No collaboration is allowed. Plagiarism will be checked and will not be tolerated (including reuse of the assignment solutions from past submissions).

The programming language for this task is Python 3. 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 must follow the instructions in the assignment comments!**

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)

numpy.linalg.pinv, numpy.linalg.inv (10% of assignment 1 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)

*. Lstsq (30% in assignments 1,3, 15% in assignment 2, 60% in assignment 4

numpy.*.interpolate, torch.*.interpolate, scipy.interpolate.splprep, scipy.interpolate.splev (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 or self-inspecting code).

Attached are mockups of 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

Do not use any folders inside the ZIP file! If the zip will contain folders the assignment will not be checked.

All assignments will be graded according to **accuracy** of the numerical solutions, **running time**, and special constraints specified in the grading policies.

You are encouraged to reuse your own code in different assignments.

Example test functions

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 like 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 specified 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. $f_8(x) = e^{ex}$
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 (10pt):

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.

Grading policy (10pt):

Running time complexity = o(n^2): 0-8

Running time complexity = O(n): 5-10

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 at least 5pt.

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 $[-1,1]$.

Assignment 2 (10pt):

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:

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

Grading policy (10pt): The grade will be affected by the number of correct/incorrect intersection points found while the running time of **Assignment2.intersections(..)** is capped.

Assignment 3 (10pt):

Check comments in Assignment3.py.

3.1 **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 (5pt): The grade is affected by the integration error only, provided reasonable running time e.g., no more than 5 minutes for n=100.

3.2 **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 (5pt): The assignment will be graded according to the integration error and running time.

Assignment 4 (10pt)

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 (10pt): 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

Assignment 5 (10pt).

Check comments in Assignment5.py.

5.1 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.

Grading policy (5pt): the grade is affected by your running time.

5.2 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 (5pt): the grade is affected by the error of the area function of the shape returned by Assignment4.fit_shape.