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. $f_5(x) = arctan\ arctan\ (x)$
6. $f_6(x) = \frac{sinsin\ (x)}{x}$
7. $f_7(x) = \frac{1}{lnln\ (x)}$
8. $f_8(x) = e^{e^x}$
9. $f_9(x) = ln\ ln\ (ln\ ln\ (x)\ )$
10. $f_{10}(x) = sin\ sin\ (ln\ ln\ (x)\ )$
11. $f_{11}(x) = 2^{\frac{1}{x^2}} * sin\ sin\ \left(\frac{1}{x}\right)$
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 n.

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

You can sample fewer than n points, but sampling more than n points will result in an exception.

It can be assumed that n<=100.

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.