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.

Grading policy (10pt):

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).

I chose to use Bezier curves in my code.
My code samples uniformly n points from the given function.
Then I find the control points by using Piecewise Bezier interpolation in matrix form.
To find the a values in linear time I use the Thomas algorithm.
Then I make the function poly(x) which is the output. In this function I go over all the x values to find the approximal location of the given x. Once I find it, I save the index of the first x value that is lower than the given x. Then I build a Quadratic Bezier curve using the x value in the index I found, the following x value, and the two control points with the index I found. Meaning I save time by calculating only one curve.
Then I find the t value that matches the given x by calculating where it should be on the curve.

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:

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

Grading policy (10pt): 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).

I chose to use Newton-Raphson and Bisection methods in my code.
I split the given range into smaller ranges, then used Newton-Raphson on each range.
I made some adjustments to Newton-Raphson so (1) I won't miss roots and that (2) I won't waste time.

(1) My code checks each point if it's a root by checking if its y value is 0. If it is, I add it to the roots list and continued to the next range.
(2) My code checks each two points that makes a range – if the multiplication of their y values is positive it means they don't have a root between them. If it's positive I continue to the next range and by that I save time, if it's negative I try to find the root. There is always a chance to miss a root between them but those chances are decreased as the length between the points is smaller. This can be solved by sampling more points.

I also checked in Newton-Raphson if the point I use is out of range, it could be out of range because the algorithm is changing the point until we can declare it a root and the calculations can cause the point to be out of range. If it happens I use the Bisection method on the original range. The Bisection will avoid this problem because it checks only points inside the range and it can't get out of the given range.
I also stop both of the methods if there are too many iterations so they won't be stuck in a loop. If Newton-Raphson is stuck in a loop it's probably because of a problematic root, then it goes to Bisection. If Bisection is also stuck in a loop, it will stop after a number of iterations.
I picked to use mainly the Newton-Raphson method because it converges quickly to a root of the function meaning it requires only a few iterations
It's also capable of finding roots with high accuracy and it works well with a lot of different types of functions.

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)

In my code I used several methods:
If the maximal number of points I could use was 1, I used mid-point rule.
If n was 2, I used the trapezoidal rule.
If n was higher than 2, I used composite simpson's rule.
The composite simpson's rule can't work with an odd number of points because it uses a weighted average of the function values at evenly spaced points within each subinterval to estimate the integral. This is why before I run this method I first check if n is odd. If n is odd I subtract n by 1, then runs the algorithm.
I picked to mainly use composite simpson's rule because it's a very accurate method and provides better results than other the methods.

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.

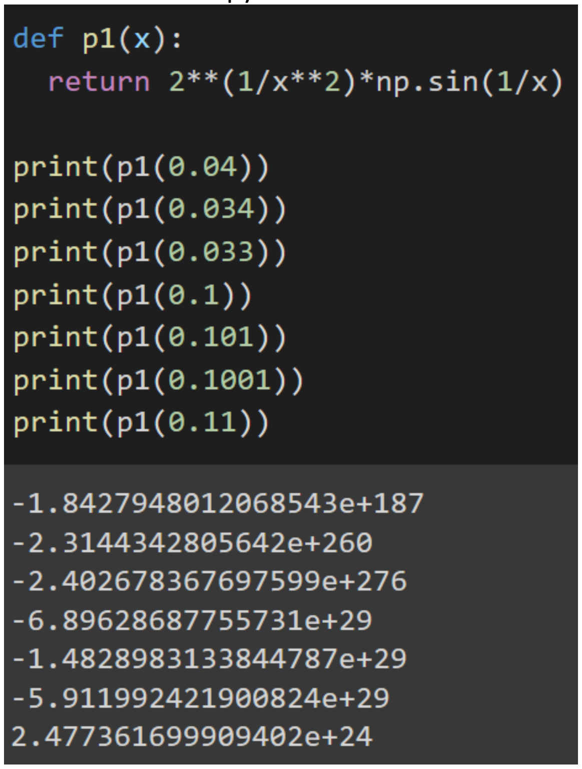
Question 3.2: Explain the key points in your implementation of Assignment3. areabetween (…) (4pt).

I subtracted the two given functions to make a new function that I could use on the integrate function.
Then I used the intersection method from assignment 2 to find the intersections between the two functions. To make sure it would work and that I won't miss any intersection I used a for loop and in each loop I used the intersection function on every part of the range [1,100] meaning in the first loop I picked the range [1,10], in the second [10,20] and so on.
each intersection I got I added to a list of ranges.

If I got less than 2 intersections I returned nan.
I went over the intersections and picked two each time by their order, then used them as a range (a and b) for the integrate function I wrote in the same assignment. Each result I got from this function I saved as absolute value and summed all the values and returned the sum.
In conclusion, I found all the intersections of the functions and used them to find parts of the whole area by using the integrate function.

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)

This function is difficult for numeric integration with equally spaced points because as we get close to the point (0,0) the function changes its value in a rapid pace to a very high or very low values making it difficult to calculate an accurate approximation of its area using equally spaced points.
We can check it in python:



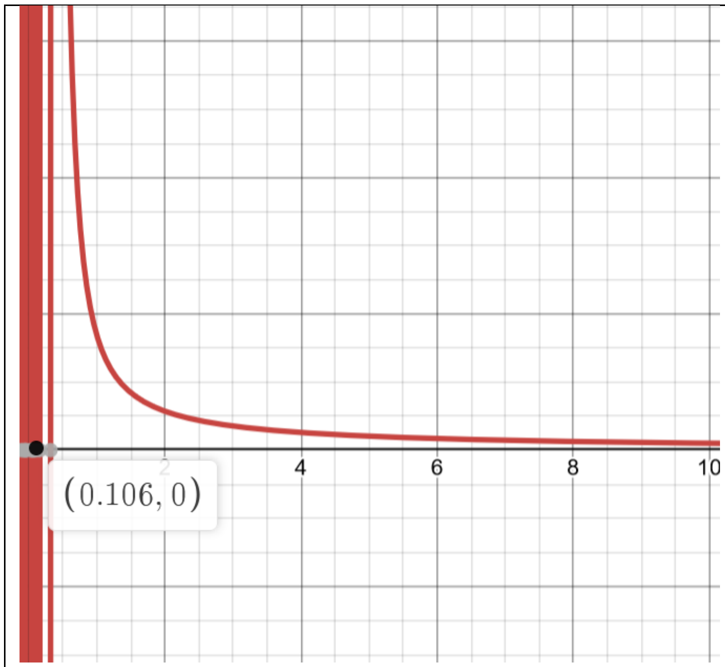
```
def p1(x):
  return 2**(1/x**2)*np.sin(1/x)

print(p1(0.04))
print(p1(0.034))
print(p1(0.033))
print(p1(0.1))
print(p1(0.101))
print(p1(0.1001))
print(p1(0.11))
```

```
-1.8427948012068543e+187
-2.3144342805642e+260
-2.402678367697599e+276
-6.89628687755731e+29
-1.4828983133844787e+29
-5.911992421900824e+29
2.477361699909402e+24
```

Here we see how the value of the function changes very rapidly even with a gap of 0.001 between each x value. This gap also changes as we get further than 0 which is making calculating the integral using a fixed spaced points impossible.
At x = 0.03 couldn't even calculate the y value.

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)

As we saw in the previous question there are ranges where we can't compute the integral in.
We can check in desmos how the function acts within the range:



As we can see near the point where x is 0.1 there are still very high changes in the y values for each x value. This fact Is making it difficult to calculate the error, but as we saw in the previous question we still get values in x = 0.1 and near it but when we integrate we need to find a way to use very small subintervals which is impossible when the y values are changing so drastically in a very short time – meaning we are bound to get errors in our calculations and because the y values are so high, our errors would be high as well. Those errors will be summed into a very high number we can't calculate so we can say the maximal error would be infinite.

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.

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

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

I used least squares in my code.
First, I saved the starting time of the code, then set the time limit to a bit lower than what was asked.
Then I started to create the matrices for the least squares calculations – I called the y value matrix 'b_mat' and the x values 'a_mat' while the x values in this matrix are x power j while 0<= j <=d. The creation is inside a loop, with every loop I add another row in the matrices while checking at the start of each loop if the code passed the time limit I set – if it passed it I stop the loop and start the calculations.
The calculations I made in a function just like we learned in class. Then I called the function while giving it the 2 matrices I made.
Then I return the coefficients I got from least squares as a polynomial. If I haven't got any points at the start meaning I have empty matrices, I return y = 0.

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.

Grading policy (9pt): the grade is affected by your running time.

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

I used the shoelace algorithm to calculate the area of the shape.
The shoelace method works because it calculates each triangle that forms a polygon and then sum up all the areas of the triangles to get the total area of the polygon.

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)

In this part I used k means, clockwise sort and shoelace algorithm.
First I save the start time and the time limit.
Then I sample points in a for loop, at every start of a loop I check if I passed the time limit, if It did the loop breaks.
Then I calculate the number of clusters I should use based on the number of points I got.

Then I use kmeans function to fit the points I got earlier.
Before using the points on the shoelace algorithm to calculate the area I needed to sort the points in a clockwise order for the shoelace to work. To do that I found the value of the center point of the shape, then I sorted the points by sorting them by their angles using the center and arctan of each point.
In the end I returned the new shape with the fitted points.
To do that I set contour in the init function of my shape. I also set area function for my shape based on the shoelace algorithm.