Exercise 3: Numerical methods

Šimun Šopar

The goal of the third exercise was to create an algorithm for calculating first and second derivative of exp(x), compare it to analytical values and plot relative error.

Source code EX3-double.cpp containts algorithm in C++ for calculating needed derivations.

The source code contains three main functions, one for calculating first derivative using three-point formula, one for second derivative using three-point formula and one for using first derivative five-point formula. Derivatives where calculated for the values of $x = \{0,1,2,3,4,5,6,7,8,9,10\}$, with values of derivation step $h = \{0.1, 0.01,\ 0.001,\ \dots, 10^{-8}\}$. The program than prints out those values. I didn't bother with putting all the results in a table because that table would be very unclear and too big, so reading direct from program ouput seemed like a better solution here. We could expect that as *h* becomes smaller, the results should become better. That is true from a purely mathematical stand-point. From stand-point of computer precision, we know that will not be the case. As we see in example Table 1. for value of *x* = 3, at first the results are getting more precise, but with even more decreasing *h*, results tend to become less precise. So, there is a „*sweet spot*“, where *h* is neither too small nor too big and the results are optimal. These are the reasons of numerical errors.

h	**Numerical derivative**	**Exact result**
0.1	20.119029559992878	20.085536923187668
0.01	20.085871683809838	
1E-3	20.085540270775578	
1E-4	20.085536956706555	
1E-5	20.085536923630791	
1E-6	20.085536926117690	
1E-7	20.085536895919621	
1E-8	20.085536789338210	

Table 1. Three-point formula results for *x* = 3.

(Blue fill marks the optimal numerical result)

Truncation error

First error we get is truncation error. When using three-point formula, $f_3'(x) = \frac{f(x+h)-f(x-h)}{2h}$, we can expand $f(x \pm h)$ in a Taylor series and insert them in three-point formula. The first and the biggest term in the resulting f_3' would be exactly f'. But we would also get higher order Taylor terms multiplied by some positive power of *h*. We neglect those terms in the fomula, so those terms represent truncation error. For larger *h*, these errors are big enough to be spotted. For an extreme example, lets put *x* = 3, *h* = 3. The numerical value for first derivative using three-point formula is 67.071465582122514 which is more than three times larger than analytical value.

If we put *h* = 0, we would get exactly the derivative. However, we cannot to that on the computer since *h* is in the denominator. But we can decrease *h* to value close to 0. As seen on Table 1., result for *h = 0.01* are better than for *h = 0.1*, we decreased truncation error. For even smaller *h*, we get even smaller truncation error. However, even tho truncation error decreases with smaller *h*, other errors occur instead.

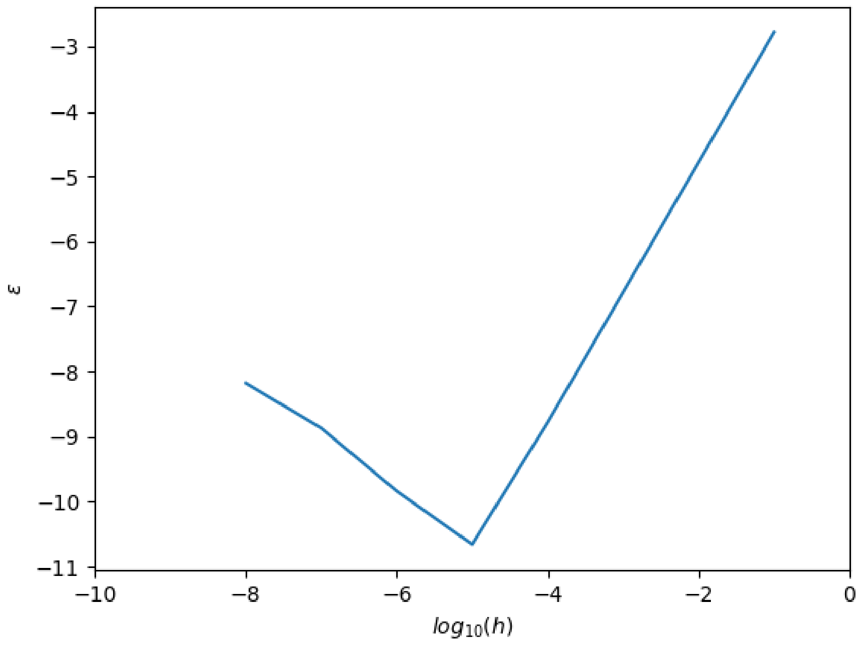
Loss of precision

For smaller and smaller *h*, numbers in numenator become very close. Substracting two close numbers results in loss of precision. After that we divide two small numbers which can result in even more loss of precision. This will not be a problem if we stick to, perhaps, 6 digits, we can see in the Table 1. that results would be nearly the same for all *h*. That is a viable solution when all we need to do is calculate the derivative. However, often times we need to persue further operations with the result. In that case, a small precision error in the beginning can pile up and become quite a significant error at the end result. So for decreasing *h*, we have closer numbers to subtract and smaller numbers to divide, leading to greater loss of precision.

For that reason it is important to know which *h* works best for us.

Plotting error plots

The code EX3-double.cpp calculates derivatives to 15 digits than prints them. It also prints the corresponding value of exp(x) to 15 digits taken from Python, this represents analytical solution and is used for comparison with numerical. These numerical solutions are then taken to Python code EX3pit.py where the relative error ε is calculated compared to Pythons built-in exponent function, and plotted on a logarithm graph. One of such graphs is seen below, on Graph 1., others can be seen in Python code.



Graph 1. Error plt for x = 3

This graph confirmes what was already described, that total error at first decreases, but than it starts increasing with decreasing the value of *h*.

Optimal *h*

Along with three functions for derivatives, the code contains three more functions. These are used to calculate relative error of the numerical solution and than return the value of the derivatives for *h* with lowest relative errors. The results are given in Table 2. for first derivative and in Table 3. for second.

x	**Optimal *h***	**Numerical result**	**Analytical result**	**Error**
0	1E-05	1.000000000012102	1.0	-10.917147335130622
1	1E-05	2.718281828517631	2.718281828459045	-10.666500449909764
2	1E-06	7.389056098983102	7.389056098930650	-11.148832073159653
3	1E-05	20.085536923630791	20.085536923187668	-10.656359279046388
4	1E-05	54.598150033058324	54.598150033144236	-11.803125502651493
5	1E-05	148.413159099902660	148.413159102576600	-10.744320289462523
6	1E-05	403.428793484294890	403.428793492735110	-10.679412891927502
7	1E-05	1096.633158397252800	1096.633158428458500	-10.545827824053962
8	1E-05	2980.957986983411000	2980.957987041728300	-10.708558781540550
9	1E-05	8103.083927426265300	8103.083927575384200	-10.735117551642620
10	1E-05	22026.465794260719000	22026.465794806718000	-10.605753152055946

Table 2. Optimal values of first derivative

x	**Optimal *h***	**Numerical result**	**Analytical result**	**Error**
0	1E-04	0.999999993922528	1.0	-8.216277067221842
1	1E-04	2.718281866265214	2.718281828459045	-7.856731817214680
2	1E-04	7.389056122519828	7.389056098930650	-8.495876161407724
3	1E-04	20.085536789338210	20.085536923187668	-8.176266829627654
4	1E-04	54.598147869455680	54.598150033144236	-7.401983179498572
5	1E-04	148.413158740368000	148.413159102576600	-8.612513657186696
6	1E-04	403.428794015780740	403.428793492735110	-8.887227315007076
7	1E-04	1096.633150154956900	1096.633158428458500	-8.122372016266532
8	1E-04	2980.958015541543800	2980.957987041728300	-8.019513807151563
9	1E-04	8103.083928290284300	8103.083927575384200	-10.054404964202075
10	1E-04	22026.465740054831000	22026.465794806718000	-8.604545730371505

Table 3. Optimal values for second derivative

All of the errors that are present in calculating in first derivative are also present in second derivative. The numerical formula of course is different for second derivative, so the size of truncation error and loss of precision are different and depend differently on *h*. We can see that error for first derivative is smaller (since this error is log scale, more negative value means smaller error) and that first derivative has smaller optimal *h*. So we can expect that as we calculate higher derivatives numerically, third, fourth, and so on, the error would increase.

Three-point vs. five-point

For the end, let's look at the difference in three-point and five-point formula for first derivative. For three-point formula, leading term in truncation error is proportiona to h^2, while in five-point it's proportional to h^4. So we expect five-point formula to be more precise. Table 4. shows optimal *h* for couple of values of *x*, as well as corresponding error, for both formulas.

x	**Optimal h (3-point)**	**Optimal h (5-point)**	**Error (3-point)**	**Error (5-point)**
3	1E-05	1E-03	-10.656359279046388	-12.971286298959720
4	1E-05	1E-03	-11.803125502651493	-13.124411910664392
5	1E-05	1E-03	-10.744320289462523	-12.305025785678870

Table 4. Comparison between three-point and five-point formula

As we can see, error is smaller for five-point formula. Also, optimal *h* is bigger for five-point formula, meaning that truncation error is greatly reduced and formula works better than three-point for bigger *h*, and also that loss of precision is reduced because we don't have to resort to such small *h*.

Perhaps if we would contruct seven-point formula, the result would be even better.

So the main difference between three-point and five-point formula is that we can sue bigger *h* for five-point, meaning that total error is reduced.

To conclude, the main problem of numerical derivation is limitiations of representing real numbers with finite amount of digits and choosing the optimal *h* for our problem as to reduce error to a minimum.