Numerical methods Exercise 4

Šimun Šopar

The goal of Exercise 4 is to examine basic algorithms for numerical integration, implement them in C and try calculating some integrals.

C++ code EX4.cpp contains the code for numerical integration. Functions *trap1, trap2, trap3,* contain the algorithm for calculating integral using trapezoidal rule for given number of sub-intervals N. Functions *simps1, simps2, simps3,* do the same but with Simpson rule for numerical integration. The results are then imported to Python code EX4.py where relative error is plotted.

Also some other integrals where calculated using code EX4-2.cpp and plotted in EX4-2.py.

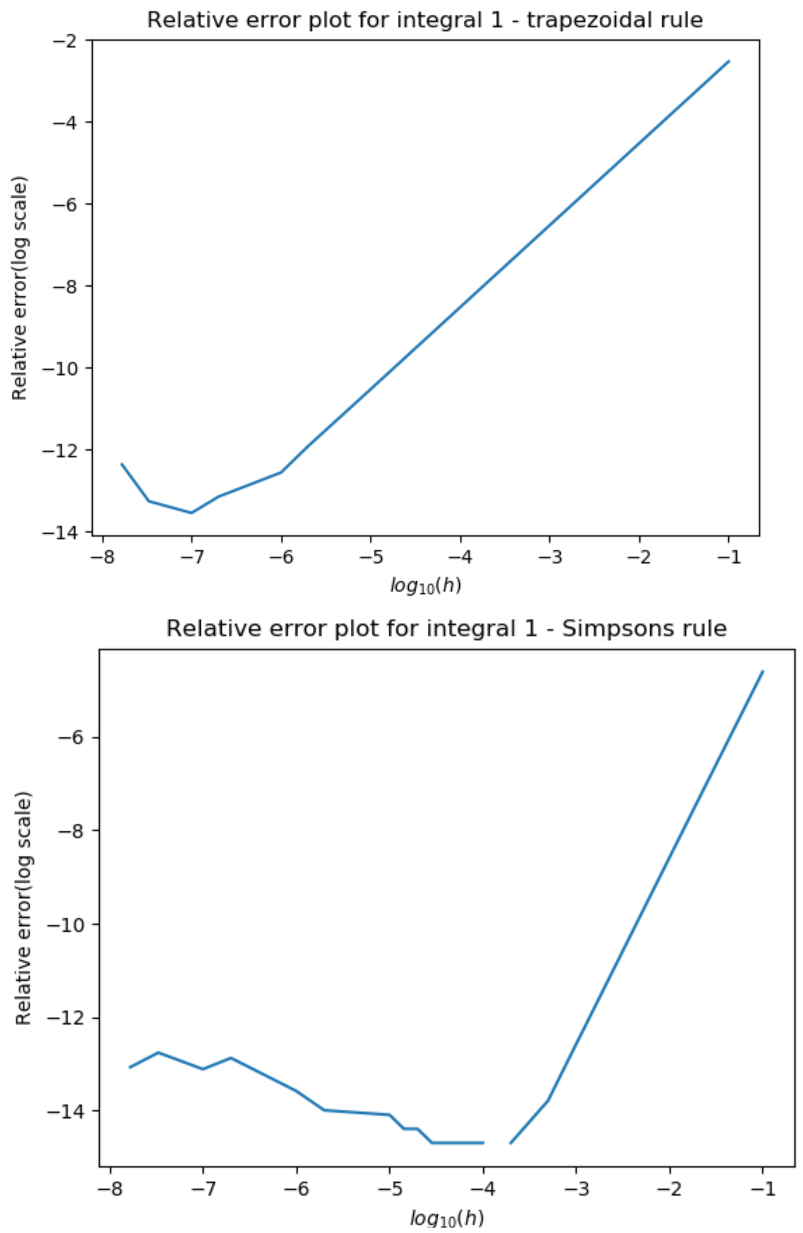
INTEGRAL 1

First integral to calculate was $\int_1^2 \frac{dx}{x^2}$. This can be done easily analytically, $(-1/x)|_1^2 = 1 - 1/2 \ = \frac{1}{2}$. Functions in C++ code for integrating take one argument, number of steps N. The code then calculates step size h and calculates the integral using trapezoidal or Simpson rule. Number of steps vary from 10 to 6E+7. We can expect that for lower number of steps (10, 50, …), the result is somewhat imprecise due to numerical error. We also expect for smaller h loss of precision leading to imprecise results. So, just like in the previous exercise, we expect there to be an optimal h not to big nor to small at which the integral is the most precise.

Let's look at trapezoidal rule first. Here we estimate functions with the linear term in Taylor series. Truncation error is proprtional to h^2. So for big h, i.e. small number of steps, results are quite imprecise, around 0.5015… for N = 10. Direct cause of this is that 10 steps is too small. In case we did numerical calculations ourselves by hand (i.e. if computers didn't exist), this would be an accepting first estimate since calculating for large number of steps would be very time consuming. But since computation of 10 steps and 500 steps is nearly the same on a computer, there is no need to have such small number of steps.

As N increases, results become more precise. However, for too large N, or too small h, the results start becoming incorrect due to loss of precision and roundoff error. This is the same thing we already saw with derivatives. Only this time the effects are smaller because we only have adding and multipyling in the formula, as opposed to subtracting and dividing close numbers in derivative formulas. That is why we don't see this effect so much when integrating. Never the less, computer has finite precision. For too small h, the computer has to calculate the function at two very close points, and with computers finite precision, it is sometimes impossible to calculate the exact value of the function at exactly these points, but rather computer gives some estimates with finite error. Also, when calculating the integral, we sum up more terms since there are more points, leading to a bigger number, which is then multipylied with h, which is now a smaller number. Multiplying small and big number can lead to loss of relevant digits.

If we want a more precise formula, we should use Simpson rule. Simpson rule aproximates functions to the second, quadratic, term of Taylor series. Error of Simpson rule is proportional to h^4. This means we can use smaller number of steps (larger h) with better precision, and can avoid loss of precision. Below are two graphs, relative errors for trapezoidal and Simpson rule.



Graph 1. Relative errors for the first integral

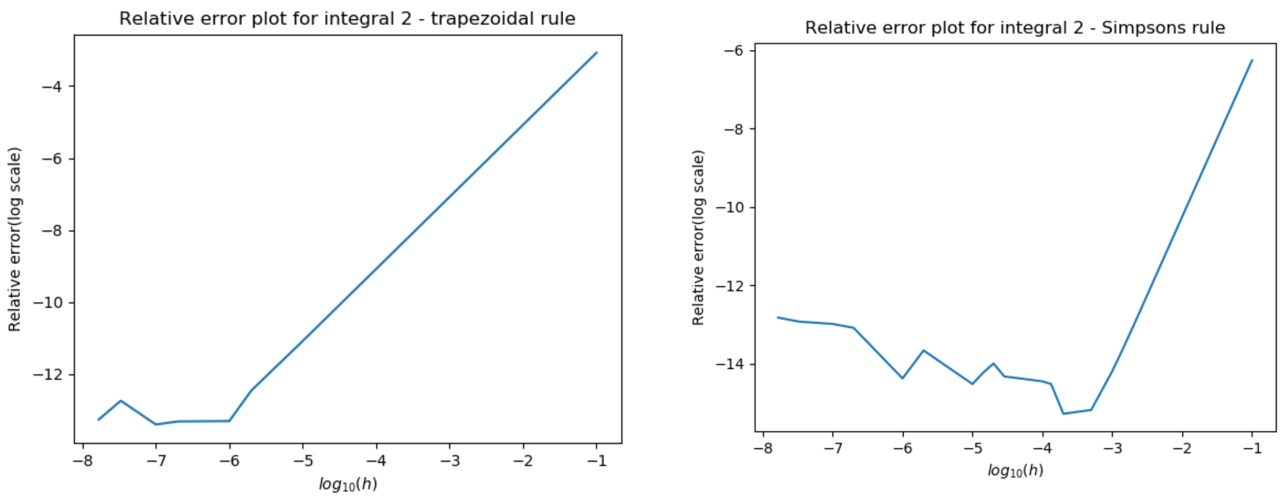
As we can see from the graph, the relative error for higher h is smaller for Simpson rule, as expected. We also see that optimal h for Simpon rule is larger then for trapezoidal, and that the smallest relative error for Simpson is smaller then the smallest relative error of trapezoidal rule. However, we also see that, since optimal h is larger in second case, loss of precision is more noticable earlier on for Simpson rule. That is because truncation error dies off pretty quickly. Also, along side that, is that the sum in Simpson rule will be larger, and that sum will be multipylied by $h/3$, which is smaller, so greater loss of precision accurs here then in corresponding trapezoidal case. Never the less, Simpson rule is overall more precise. The trade-off is that it takes longer to calculate, but not significantly longer. Simpson rule works better for smaller *N*, so if we were to calculate integrals by hand, this would be better option then taking more steps and using trapezoidal rule, both time wise and precision wise. We could go to higher order of Taylor series, but the formula would become significantly more complex and the computation time would increase (and probably loss of precision as well, even tho truncation error would decrease). So it is important to know a good balance between precision and complexity of implementing and calculating something.

Comment: Another point to comment on is the peculiar case of $h \approx$ 1E-04 where we see a disconntinuity in relative error of Simpson rule. The reason for that is that I am working with 15 digit precision. For that particular *h* the result given by C++ will be exactly the same as the analytical solution. So Python encounters 0 in the logarithm and prints out an error. This of course stems from finite precision of C program. If we were to use 20 digits, we would encounter some finite relative error.

Everything described here regarding errors hold the same for other integrals. Of course the errors themselves will be different since the functions are different, but the cause of errors is the same. So I will not be going through explanation of errors again in the rest of the *PDF*, I will just list the results (graphs).

INTEGRAL 2

Second integral was: $\int_0^1 e^x dx = e^x \Big|_0^1 = e - 1 \approx 1.71828$. Below are relative errors for the integral. As in case of first integral, Simpson rule generally gives better results. Here are the graphs.

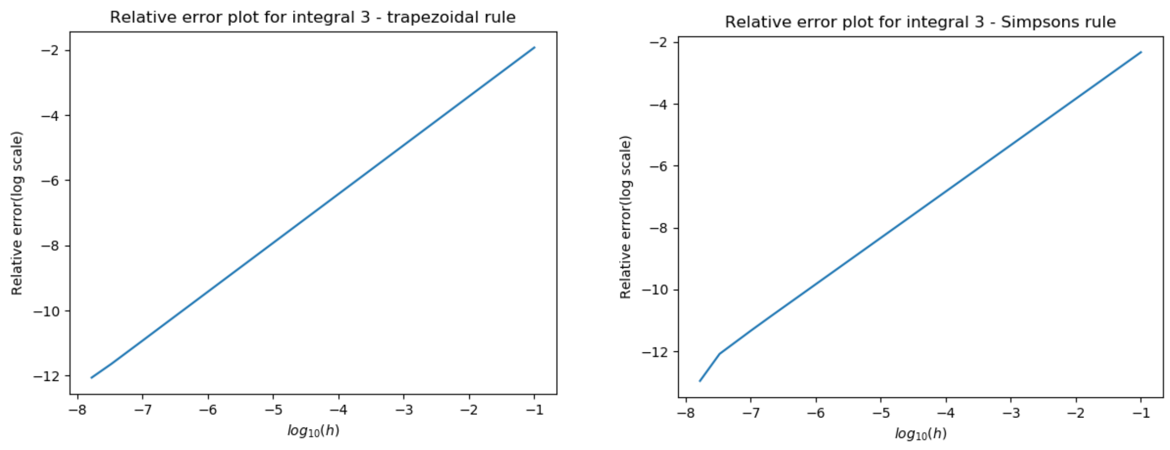


Graph 2. Relative error for the second integral

INTEGRAL 3

Third integral was: $\int_0^1 \sqrt{1 - x^2} = 1/2 \cdot (\sqrt{1-x^2} \cdot x + arcsin(x))|_0^1 = 0.5 \cdot (\frac{\pi}{2} - 0) = \frac{\pi}{4} \approx 0.78540$

Intrestingly, this subintegral funtion from 0 to 1 actually corresponds to a quarter of a circle of radius 1 (this can easily be shown, $y = \sqrt{1-x^2} \rightarrow y^2 + x^2 = 1$). This means the integral calculates quarter of circles area, $r^2\pi$, which for r = 1 gives exactly $\pi/4$. Graphs of error are given below.



Graph 3. Relative error for the third integral

ADDITIONAL INTEGRALS

I also calculates two additional integrals, first was $\int_0^\pi sin\ x\ dx = 2$.

Second was $\int_0^{0.5} xe^x dx = \frac{e^{0.25}}{2} - 0.5 \approx 0.142013$

Both integrals are calculated in file EX4-2.cpp and plots for error of Simpson method are in code EX4-2.py. Here I would just like to comment one interesting thing about the first integral, the one with the sine, and that is its upper limit of integration, which is π. In all the other cases we had rational numbers as limits. Here we have an irrational number, and not just that, but a trascedental number as well. So we have another point where we can get errors. Computers cannot hold infinite number of digits, so they truncate our upper limit. However, I think this contibutes to the error insignificantly, it's more of an interesting comment to make.

So, as we saw, numerical error always exist, but it depends on the function we are integrating, its limits, as well as number of steps and method used. Each integral should thus be analyzed as its on case when determining optimal results. However, from looking at errors of known integrals, we can get a rough estimate which *h* works the best for when we integrate functions without analytical solution, which is, along side checking solutions of analytical(known) integrals, the ultimate goal of numerical integration.