Exam Project: Exercise 15 - Generalizing an ODE solver to the Complex Plane

Mikkel Elkjaer Pedersen

This short report presents my reflections on the exam exercise 15: *Generalize the ODE solver of your choice to solve ODE with complex-valued functions of complex variable along a straight line between the given start- and end-point.* As this exam project is an extension of the ODE-exercise in the course I have reused large parts of the code from my ODE-solver. Thus, this ODE-solver also utilizes a Runge-Kutta23 stepper as the advancing mechanism, also called the Bogacki-Shampine method.

My first consideration in this project was how to modify the working ODE-solver for real functions to complex functions, that is, what is the most natural way to incorporate the imaginary part into the already established part. A first thought was to exploit that the solver already works for any number of variables. In this sense one might *pair up* variables where each pair would represent the real and complex parts of a single complex variable, respectively. Even though this seems like a natural way of solving the problem I could not convince myself that this would work in general as complex functions of complex variables in general "mixes" the two parts. As an example, squaring a complex number with a non-zero imaginary part intricately affects the real part, which might even be negative, and therefore i quickly abandoned this approach. The much safer approach, although more cumbersome in typing, is to just cast every variable as complex from the very start. To do this i chose to utilize the GSL-library representation. The price of the build-in complex arithmetic functions is of course the famously long function names of GSL-routines. Looking back at the resulting code after implementing these modifications I believe it would have been just as easy to use my own small library for dealing with complex variables from the exercise called "komplex".

With regards to the mathematics in the modifications not many comments are necessary as the Runge-Kutta methods nowhere assume real variables. Thus adhering to the arithmetics of complex numbers should leave a proper algorithm. Most notable is the caution one must show when considering the step, h, as the meaning of a step and the step size coincide in one dimension, that is for real variables, but not for two dimensions. Thus the step sought by the driver for a complex function of a complex variable is a complex number, while the step size naturally remains a real number.

As in part B of the ODE-exercise in the course I have retained the property that the solver saves the steps validated by the driver and the corresponding function values. For this part I have one reflection. This concerns the fact that the user must supply an upper limit to how many steps that is considered feasible for the driver to take. If the upper limit of the integration of the ODE is not reached within this limit an indication is send to the output along with the steps and function values that were reached before the count limit. Of course this allows the user to restart the solver at the reached point untill completion but if this is repeatedly necessary this a hardly flexible driver. Also, establishing an upper limit has the clear disadvantage that the user must somehow be able to ballpark the necessary number of steps before the routine is run. In general, the user should then also overestimate the required number of steps just to be safe. For complecated problem this might lead to serious overestimates and in consequence unnecessarily huge collector matrices to be handled by the driver which could slow the routine quite significantly. For such complicated problems the user relies heavily on restarting the driver and storing the results in between restarts. The stored path might also be used to reveal treacherous or seamless sub-regions for which a more or less demanding accuracy should be supplied to the driver before a total restart.

To demonstrate that my routine works I have solved three very simple problems. The first is in fact a purely real problem meant as an explicit display that real problems may of course also be handled by this complex driver. Thus, as seen in figure 1, the real exponential function is determined by its ODE-representation $\mathrm{dy}/\mathrm{dt} = y$ with initial condition $y(0) = 1$. Secondly, the ODE for the complex exponential function along the imaginary axis is used to produce the purple and blue graphs in figure 2. Do note that the imaginary part is strictly 0 all throughout as it should be. Lastly, the ODE for the complex exponential function along the real axis is integrated up to 2π to show one revolution of the sine and cosine functions which are of course the imaginary and real parts of the complex exponential, respectively. The result is included in figure 2.

As stated in the exercise description the integration of the ODE should be along a straight line in the complex plain. To ensure this I have implemented an enforcer of this restriction into the driver routine. That is, when the user has supplied a start and end point of the straight line for which the driver should solve the ODE the user is also required to supply an initial guess of the first step the driver should take. Regardless of whether this guess is in fact in the correct direction the driver now extracts the approximate size of the intended first step in both the real and imaginary direction, separately. It then calculates an initial guess in the correct direction which is on the scale of the user guess for each direction, separately. Thus all directions in the complex plain should be solvable for the driver if the user provides the correct ODE even though a very misleading initial step guess is provided.

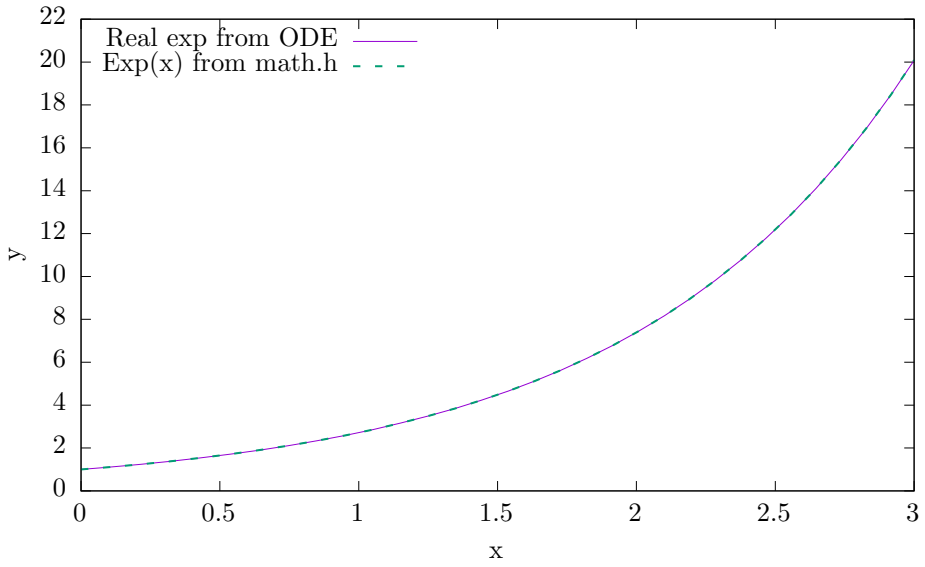

Figure 1: A plot of the ODE representation of the real exponential function and the one from the standard C library math.h in the range $[0, 3]$.

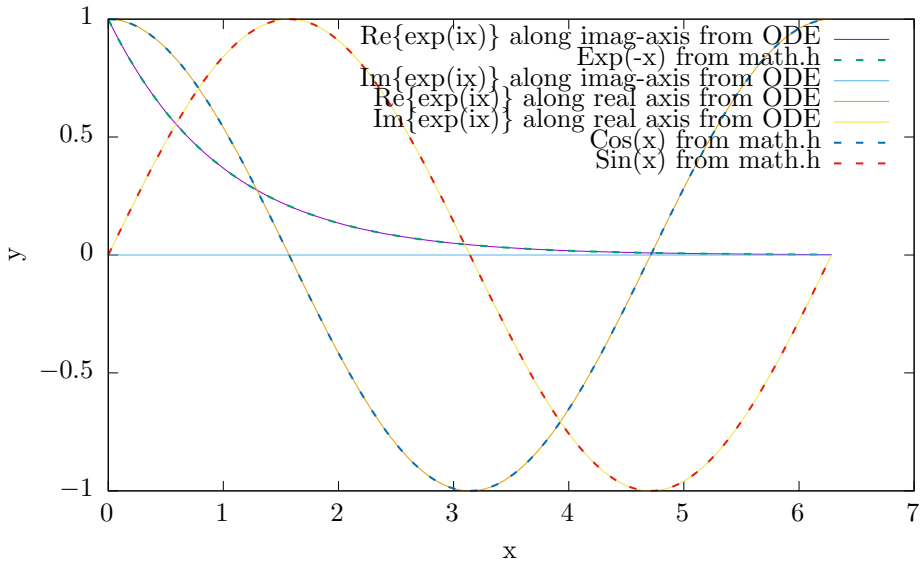

Figure 2: Here the complex exponential gotten from the ODE-routine is plotted for two inputs: a purely imaginary one and a purely real one. The corresponding functions from math.h is included for comparison, showing that the routine works as intended for complex functions of complex variables. In the case of a purely real input to the complex exponential the resulting values are split into their real and imaginary parts for easy comparison.