Einführung in die Programmierung für Physiker

WS 2017/2018 – Marc Wagner

Francesca Cuteri: cuteri@th.physik.uni-frankfurt.de
Alessandro Sciarra: sciarra@th.physik.uni-frankfurt.de

Exercise sheet 6

To be corrected in tutorials in the week from 27/11/2017 to 01/12/2017

Exercise 1 [*Numerical derivative*]

Given a function $f(x)$, calculating its derivative is usually a quite straightforward task and we rarely need to do it numerically. Nevertheless, if $f(x)$ is not expressed in term of elementary functions or if only a sample of it is known, a numeric approach is required. The simplest approach to evaluate a derivative numerically is using finite difference formulas, which can be forward backward or symmetric[1]. Symmetric ones, also known as central, are the most common ones and the two simplest read

$$f'(x) \simeq \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon} \equiv g_{\mathrm{I}}(x) \tag{1}$$

$$f'(x) \simeq \frac{f(x-2\varepsilon) - 8f(x-\varepsilon) + 8f(x+\varepsilon) - f(x+2\varepsilon)}{12\varepsilon} \equiv g_{\mathrm{II}}(x)\ . \tag{2}$$

In this exercise we will use them in a particular case, trying to understand how they work and their limitations. Since we want to run our code using both `float` and `double` variables, you are encouraged to use a pre-processor statement like `#define real double` at the beginning of your code and, then, to use `real` everywhere as type of your variables. To avoid to have troubles with the `printf` format specifier, use only `%e` which work both with `float` and with `double` variables.

(i) Consider the function $f(x) = \dfrac{x + \cos(x) - 1}{x}$ and calculate $f'(x)$ analytically.

(ii) Evaluate $\lim\limits_{x\to 0} f(x)$ and $\lim\limits_{x\to 0} f'(x)$.

(iii) Write a program which calculates the derivative $f'(x)$ numerically in the interval [`xMin`, `xMax`], discretising it with a step $\varepsilon = 10^{-1}$, so that you evaluate the derivative in $N = (\texttt{xMax} - \texttt{xMin})/\varepsilon + 1$ points. Fix `xMin=-10` and `xMax=10`. A possible way to structure your code is to,

- (a) set up a pair of `C` functions which, given x, return the values $f(x)$ and $f'(x)$;
- (b) add other two functions which implement Eqs. (1) and (2);
- (c) use a `printf` statement in order to print to the standard output the values of x, $f(x)$, $f'(x)$, $g_{\mathrm{I}}(x)$ and $g_{\mathrm{II}}(x)$.

(iv) Redirect the standard output of your code to a file and use `gnuplot` to plot $f(x)$, $f'(x)$ and $g_{\mathrm{I}}(x)$ on top of each other. Zoom around $x = 0$ and check if what you expect is indeed displayed. If not, go back to your code and try to fix it. Keep in mind what you learnt about decimal numbers comparison and decide whether it is correct to use the `==` operator.

(v) Plot now $|f'(x) - g_{\mathrm{I}}(x)|$ as well as $|f'(x) - g_{\mathrm{II}}(x)|$. Use a logarithmic scale for the y-axis setting its range properly. The command `set format y '10^{%T}'` may be useful. What do you learn from this plot about Eqs. (1) and (2)?

[1] If you are interested in reading more about it, `https://en.wikipedia.org/wiki/Numerical_differentiation` can be a good starting point. If you are interested in higher order derivatives or more accurate formulas also for the formulas for forward and backward derivatives, `https://en.wikipedia.org/wiki/Finite_difference_coefficient` should provide you with some answers. A general calculator of coefficients (for any order and type of derivative) is available on the web at `http://web.media.mit.edu/~crtaylor/calculator.html` where a great explanation about how to obtain the coefficients in general can be found.

(vi) Modify now slightly your code and move some code from your `main` to a

```
real CalculateAverageDerivativeDeviation(real xMin, real xMax,
                                         real epsilon){/*...*/}
```

function, which should now calculate

$$\delta = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\Big[f'(x_i) - g_{\mathrm{I}}(x_i)\Big]^2},$$

always discretising the interval [xMin, xMax] with a step $\varepsilon = x_{i+1} - x_i$.

(vii) Make your code calculate δ for several values of ε, starting with $\varepsilon = 10^{-1}$ and halving it, until it goes below the threshold $\varepsilon_{\min} = 10^{-6}$. Use a `printf` statement to print in an exponential form ε and δ to the standard output and redirect it to a datafile. Repeat this step both with `float` and `double` variables.

(viii) Use `gnuplot` to visualise your data. Here below, you find a possible way to plot them.

```
set logscale xy
set format xy '10^{%T}'
set xrange[1:1.e-7] reverse
plot "data_float.dat"  u 1:2 w lp pt 6,\
     "data_double.dat" u 1:2 w lp pt 6
```

Can you explain the outcome? What do you learn from your plot?

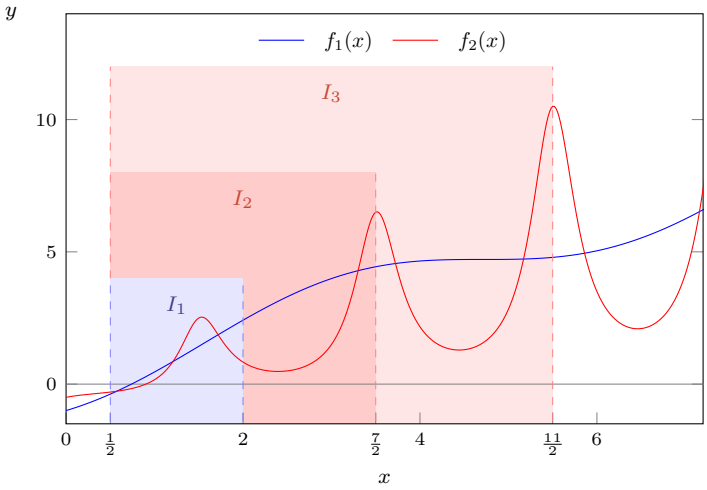
(ix) Repeat item (vii) using g_{II} in the expression of δ and plot again your data. You can add the new points on the old plot produced in item (viii). Is the outcome as you expected?

(x) Is there a way to decide whether to use `float` or `double` at compile time without editing your code? Said differently, what does the option `-D` of `g++` do and how should it be used? Be careful, though! If the user forgot to use it, your code should still compile! Can `#ifndef` and `#endif` help you?

Advanced: A `bool` parameter in your function(s) may be used to decide which finite difference formula should be used. On top, you could use a pre-processor macro calling your function(s) (maybe defined when compiling the code).

Exercise 2 [*Zeroes of functions*]

In the lecture you learnt about the *Bisection method* for the computation of zeroes of one-dimensional real-valued functions. In this exercise you will face the same kind of problem, but try to solve it with different algorithmic strategies.



The functions for which we want to find a real root, together with the given real interval in which we want to look for this root, are

(i) $f_1(x) = -\cos(x) + x$, in the interval $I_1 = [0.5, 2.0]$;

(ii) $f_2(x) = -\frac{1}{2} + x\left[\frac{3}{2} + \sin(\pi x)\right]^{-1}$, in the interval $I_2 = [0.5, 3.5]$;

(iii) $f_2(x)$, but in the interval $I_3 = [0.5, 5.5]$;

and they are plotted in the figure.

There exist, indeed, other iterative methods for *root finding* given a function $f(x)$ and some interval $[x_1, x_2]$. The common feature of all those methods is that, starting from some initial guess of an interval, within which you expect the root to be found, the value of the obtained numerical root is *iteratively refined*.
You are required to write a program to implement two such algorithms and, since you learnt about different ways of achieving the repeated execution of a sequence of statements in your code (`while` loops, `for` loops, `do-while` loops), you are requested to manage the control flow choosing per every algorithm a different kind of loop.

(i) In the case of the **Secant method**, the consecutive refinements of the position of the root are determined as the points where the straight line connecting the function values at the extrema of the search interval crosses the axis. So one needs to just use the equation for the straight line connecting $f(x_1)$ and $f(x_2)$

$$\frac{f(x) - f(x_1)}{f(x_2) - f(x_1)} = \frac{x - x_1}{x_2 - x_1}, \tag{3}$$

then impose $f(x) = 0$ and solve for x. After each iteration, a new search interval $[x_1^i, x_2^i]$ (where the superscript i runs over the iteration steps) is obtained by discarding one of the previous boundary points in favour of the latest best estimate of the root. In the Secant method the most recently determined of the two boundaries is kept for the next iteration (this requires an arbitrary choice on the first iteration).

(ii) In the case of the **Newton-Raphson method**, the consecutive refinements of the position of the root are determined as the points where the tangent line to $f(x)$ in a given point crosses the axis. So this second method requires the evaluation of both the function $f(x)$ and its derivative $f'(x)$ and one just needs to compute the tangent line to the function at some given position, which is, at first, determined as the central value of the original interval, i.e. $x_0 = \frac{x_1+x_2}{2}$. The equation of the tangent line to $f(x)$ at $x = x_0$ is

$$f(x) - f(x_0) = f'(x_0)(x - x_0). \tag{4}$$

The abscissa of the zero crossing of the tangent is taken as new best estimate for the root. The procedure is then iteratively repeated considering the ith estimate x_i of the root to determine the next x_{i+1}.

Some more important remarks are that

- The convergence of each method to its solution is established based on the shift between two subsequent estimates of the root. Iterations should continue until such shift becomes smaller than some tolerance ϵ (which you should set in a meaningful way...).
- You should not forget to set a limit to the number of iteration in case the loop-continuation condition cannot eventually become false (you cannot always ensure the convergence of your root finding algorithm).

Optional. Of course you can try to run your Newton-Raphson algorithm evaluating the numerical derivative of your functions as you learnt to do in the previous exercise. How can this affect the rate of convergence of your algorithm?