

FFR105: Introductory programming exercise

Mattias Wahde and David Sandberg and Ola Benderius

September 2, 2015

The aim of this exercise is for you to learn how to write well-structured program code. It is very important to be able to do so, partly for your own sake (well-written program code is easier to debug, modify, and extend) and partly for the sake of others (well-written code is usually easy to read). This problem is a complement to the Matlab introduction (during which you implement a genetic algorithm), the main difference being that, here, the focus is on the actual *programming*, rather than the task solved by the program. When writing, keep in mind to write as simple and clear code as possible, using appropriate variable names etc. You *should* follow the coding standard, available on the course web page.

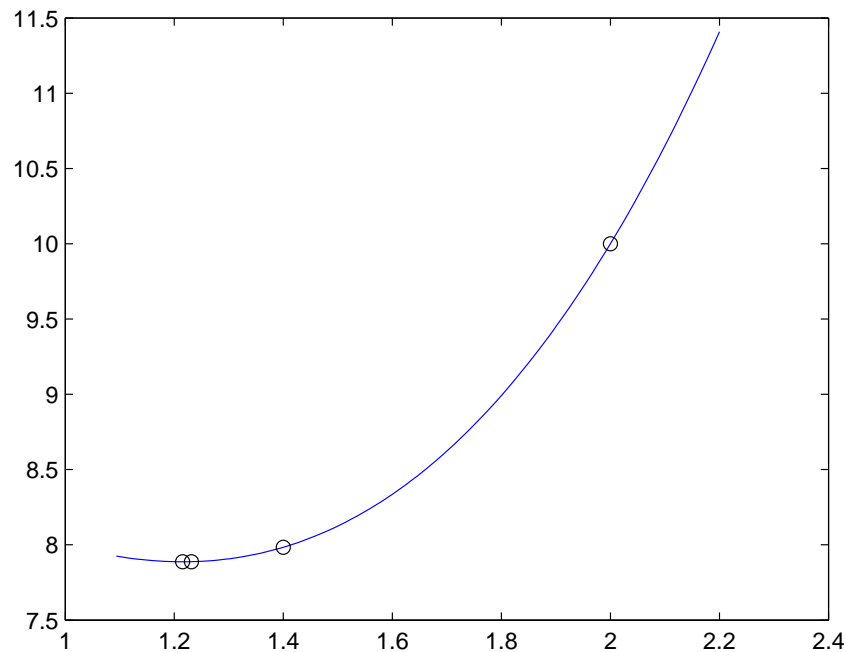
The problem that you will solve is as follows: *Write a Matlab program, following the coding standard, that takes as input (the coefficients of) an n^{th} -degree polynomial and a starting point, and which then uses Newton-Raphson's method to find the nearest stationary point of the polynomial.* When you have completed your program, collect all relevant files in a single .zip (or .rar) file and mail that file to `mattias.wahde@chalmers.se` (Strict deadline: 20150911). The program should consist of precisely the following (not more, not less):

- A function `NewtonRaphson` (in the file `NewtonRaphson.m`) that takes as input (i) a vector containing the coefficients of the polynomial considered, (ii) a starting point, and (iii) a tolerance parameter (for checking convergence), and then executes the main iteration loop while storing the iterates x_j , $j = 0, 1, \dots$. After converging (i.e. when $|x_{j+1} - x_j| < T$, where T is the tolerance parameter) the function should return a vector containing the iterates with the stationary point as the last element in that vector.
- A function `Polynomial` (in the file `Polynomial.m`) which takes x and the $n + 1$ coefficients of an n^{th} -degree polynomial a_0, a_1, \dots, a_n as input, and returns the value $f(x) = a_0 + a_1x + \dots + a_nx^n$.
- A function `PolynomialDifferentiation` (in the file `PolynomialDifferentiation.m`) which takes the $n + 1$ coefficients of an n^{th} -degree polynomial a_0, a_1, \dots, a_n and the order k of the derivative as input and returns the $n + 1 - k$ coefficients of the k^{th} derivative of the polynomial.
Example: if the input polynomial is $1 + 2x + 3x^2$ and $k = 1$, the input coefficients are $(1, 2, 3)$, and the output coefficients should be $(2, 6)$. If, instead, $k = 2$, the output coefficients should instead be (6) (a vector with one element). If $k \geq 3$, an empty vector should be returned.

Note: In this problem, you may *not* make use of any built-in differentiation functions in Matlab!

- A function `NewtonRaphsonStep` (in the file `NewtonRaphsonStep.m`) which carries out a Newton-Raphson iteration step as in Algorithm 2.3 in the course book (p. 22), taking x_j , $f'(x_j)$ and $f''(x_j)$ as input, and returning x_{j+1} .
- A function `PlotIterations`, which should be called after the end of the main loop, and which plots the polynomial (in an appropriate range, both in the horizontal and vertical directions) as well as the iterates (as circles) x_j , $j = 0, 1, \dots$. A plot example is shown below where the function considered in Example 2.4 in the course text book is plotted along with the values obtained by each step of Newton-Raphson, starting at point $x_0 = 2$.
- A main script `Main.m` that defines the polynomial coefficients and starting point, calls the `NewtonRaphson` function with these parameters and finally calls `PlotIterations` as in the following example:

```
polynomialCoefficients = [10 -2 -1 1]; % Defines the polynomial 10 - 2x - x^2 + x^3
startingPoint = 2;
tolerance = 0.0001;
iterationValues = NewtonRaphson(polynomialCoefficients, startingPoint, tolerance);
PlotIterations(polynomialCoefficients, iterationValues);
```



Note that the program should be able to handle *any* non-negative integer values of n and k (including the cases $n = 0$, $k = 0$, $k > n$ etc.). If the iterates cannot be computed (for example, if $n < 2$ or $f''(x_j) = 0$ for some other reason), the program should give a suitable error message (in the form of a text string printed to the screen) and then terminate in a controlled way (without giving any Matlab error output). Note also that all computations of polynomials (including, for example, $f'(x)$) should be carried out using the function `Polynomial`, with appropriate inputs (obtained, in the case of $f'(x)$, by a call to `PolynomialDifferentiation`).