EP 501 Homework 2: Nonlinear Equations and Root-finding

October 7, 2019

Instructions:

- Complete all problems.
- Submit all source code and publish Matlab results via Canvas.
- Discussing the assignment with others is fine, but you must not copy anyone's code.
- I must be able to run your code and produce all results by executing a single top-level Matlab script, e.g. assignment1.m or similar.
- You may use any of the example codes from our course repository: `https://github.com/mattzett/EP501/`.
- Do not copy verbatim any other codes (i.e. any source codes other than from our course repository). You may use other examples as a reference but you must write you own programs (except for those I give you).

  ```
  load testproblem.mat
  ```

  or double click on the .mat file in the Matlab file browser.
- For demonstrating that your code is correct when you turn in the assignment, you must use the test problems in the course repository found in...
- All solutions for this homework must be accurate to at least six decimal places.

Purpose of this assignment:

- Learn principles behind numerical solutions to nonlinear algebraic equations.
- Develop good coding and documentation practices, such that your programs are easily understood by others.
- Hone skills of developing, debugging, and testing your own software
- Learn how to build programs on top of existing codes

1. Finding roots of functions lacking a closed form:

   (a) Alter the Newton method function from the repository (`newton_exact.m`) so that it implements the approximate Newton method, i.e. so that the derivative is computed numerically as in Equation 3.77 in the course textbook.

   (b) Write a block of code that uses your `newton_approx.m` function to find the first root (i.e. the smallest one) of the Bessel function of order zero: $J_0(x)$ for the region $0 \leq x \leq \infty$. Plot the Bessel function first and use the plot to select a starting point in the vicinity of the first root. The Matlab basics scripts from the course repository (`https://github.com/mattzett/EP501/blob/master/matlab_basics/matlab_basics.m`) shows you how to use Matlab's built in Bessel functions.

   (c) Product a version of this script that finds the first six roots of the Bessel function of order zero (these roots are needed for solutions of various types of boundary value problems in physics and engineering). These roots are commonly listed in ODE and PDE textbooks; look them up and verify your solutions (cite your sources).

2. Numerical solution for multiple polynomial roots:

   (a) Suppose you have a polynomial of known or given order and need to find all of its roots. Write a block of code or a script that uses the exact Newton method (e.g. the function in the course repository) to find all of the real-valued roots of a polynomial. Assume that you do not need to identify repeated roots (if any).

   (b) Produce an altered version of your code to deal with the fact that there are potentially complex roots to your polynomial. Use this code to find all polynomial roots, including complex-valued solutions.

3. Polynomial deflation and its use in root finding for high-order polynomials:

   (a) Write a function that *analytically* solves a quadratic equation of the form:

   $$ax^2 + bx + c = 0 \tag{1}$$

   given a column vector of coefficients $[a, b, c]^\mathsf{T}$. Test your code on a polynomial with known roots and make sure your code works for complex conjugate roots.

   (b) Write a polynomial division algorithm (cf. pgs. 194-195 of the course textbook) capable of dividing a given polynomial $P_n(x)$ (of order n and defined by a set of coefficients) by a given divisor $(x - N)$. Viz. find $Q_{n-1}(x)$ and R such that:

   $$P_n(x) = (x - N)Q_{n-1}(x) + R \tag{2}$$

   Test your code with on a polynomial with a known factorization to make sure that it works.

   (c) Use Newton's method to numerically find *one* root of a given high-order polynomial, and then use your synthetic division algorithm to factor that root out of the polynomial to produce a lower-order polynomial (i.e. $Q_{n-1}(x)$).

   (d) By iterating this process repeatedly find all roots of the polynomial; e.g. by then taking Q_{n-1} and using Newton's method to find one of its roots and then factoring that root to produce Q_{n-2} all the way down to Q_2. Once you have Q_2 use your quadratic formula to find the final two roots.

   (e) Note that the polynomial division approach has the drawback of introducing errors at each stage of factorization and compounded these across later factorization steps. The best way to deal with this is to take the estimated root from the deflation process of part d of this problem and use them as starting points for separate Newton iterations, which will then cause these inputs to converge to the true roots to a desired level of precision. Use this method on your root estimates to produce polished root calculations that are *all* accurate to six decimal places.

4. Multivariate root finding:

   (a) Use the multi-dimensional Newton method to find all four roots of the system:

$$x^2 + y^2 = 2x + y \tag{3}$$
$$\frac{x^2}{4} + y^2 = 1 \tag{4}$$

   (b) Produce an altered multi-dimensional Newton method to find a root for the three equations system:

$$x^2 + y^2 + z^2 = 6 \tag{5}$$
$$x^2 - y^2 + 2z^2 = 2 \tag{6}$$
$$2x^2 + y^2 - z^2 = 3 \tag{7}$$