MCS 471 Project Two : Bairstow's Method

The goal of this project is to investigate a numerical algorithm to find all roots of a polynomial equation: Bairstow's method. Since this method involves synthetic division readily available in MATLAB's **deconv** command, we will use MATLAB in our numerical investigations.

0. MATLAB and Polynomials

Polynomials in MATLAB are represented by their coefficient vectors. For example, the cubic polynomial $3.3x^3 - 2.23x^2 - 5.1x + 9.8$ is entered in MATLAB typing $c = [3.3 \ \ -2.23 \ \ -5.1 \ \ 9.8]$ at the prompt. With $y = \textbf{polyval}(c, x)$, we evaluate the polynomial at some point x. After typing $r = \textbf{roots}(c)$, the vector r will contain approximations for the roots of the polynomial.

We can group commands into .m files, defining functions. The name of the function should match the file name. For example, place the definition of the function "newton" in the file "newton.m". If the path is set correctly (do **help path**), then you can call this function just as a regular MATLAB command. See `http://www.math.uic.edu/~jan/MCS471.html`, especially the .m file "newton.m" in Lecture 8 is relevant for this project.

1. Study and implementation of Bairstow's method

Read section 1.8 in the text book. A good starter is to go through example 1.3 in an interactive MATLAB session, using the command **deconv** and some extra manipulations. The book suggests Cramer's rule to solve Δr and Δs, but you may also use the \ command. For example, to compute the solution x to the linear system $\mathbf{a}x = \mathbf{b}$, we type $x = \mathbf{a}\backslash\mathbf{b}$.

In a next step, we could write one intermediate routine for splitting off one quadratic factor. The prototype for this routine could be

```
function [q,f] = bairstow1(p,eps,N)
%
% Computes one quadratic factor of the polynomial p.
% On entry :
%   p        coefficient vector of polynomial, size(p,2) > 2;
%   eps      accuracy requirement on the coefficients of result;
%   N        maximal number of iterations.
% On return :
%   q        coefficient vector of the quotient: p = q*f
%   f        coefficient vector of a quadratic factor.
```

We use `bairstow1` repeatedly until the quotient has degree two or less. On this last quotient and on all factors we apply the quadratic formula to find the roots. We could use the following prototype

```
function r = bairstow(p,eps,N)
%
% Computes all roots of the polynomial p with Bairstow's method.
% On entry :
%   p        coefficient vector of a polynomial;
%   eps      accuracy requirement on the quadratic factors;
%   N        maximal number of iterations to compute one factor.
% On return :
%   r        approximations for all the roots of p.
```

2. Numerical experiments on random and special polynomials

In principle, we cannot start with the second part of the project without a correct implementation of Bairstow's method. However, if you fail to complete the implementation and/or feel you cannot trust your code, then you may resort to Program 1.2 in the text book.

In the following assignments we study the finding of the roots for three classes of polynomials.

1. **Random Polynomials.** We generate a coefficient vector c of length 6 typing $c = \mathbf{randn}(1,6)$, defining a polynomial of degree 5. Compare the output of your implementation with the output of the **roots** command. Experiment with different levels of the accuracy requirement `eps`, for example 1.E-4, 1.E-8, 1.E-12, and 1.E-16. Make a table to show the relation between `eps` and the errors on the roots. Do you notice the effect of the propagation of errors caused by the deflation? In particular, is the first root computed more accurately than the last root?

2. **Multiple Roots.** Type $r = \mathbf{ones}(1,5)$; followed by $c = \mathbf{poly}(r)$ to generate the coefficient vector c of the polynomial $(x-1)^5$. Apply the implementation of Newton's method used in Lecture 8 to answer the following questions:

   1. Investigate how to show the geometric convergence rate to the root 1 of multiplicity 5, starting sufficiently close to 1 and using sufficiently many iterations.
   2. Extend the implementation of Newton's method to take advantage of the multiplicity to restore the quadratic convergence.

   In your answer to these questions, show the numerical experiments, the code for the extended Newton's method you used and the numerical output. State your conclusions briefly.

3. **A famous polynomial.** Consider the polynomial $(x-1)(x-2)\cdots(x-20)$. In MATLAB, we can generate its coefficient vector c by $w = \mathbf{diag}(\mathbf{wilkinson}(41))$; $r = w(1:20,:)'$ (observe the prime) and $c = \mathbf{poly}(r)$. Compare the output of the **roots** command and the output of your implementation of Bairstow's method with the exact roots. Explain why it is difficult to find the roots of this polynomial with a numerical method.

3. Deadline is Wednesday 13 February, at 1PM

Bring your project solution to class. It should contain the following:

1. The listing of the MATLAB code to implement Bairstow's method. If you resorted to the C code in the text book and extended the code, then you may get some partial credit for completing the code.
2. Numerical output, with numbers formatted in scientific format.
3. Answers to the questions in the assignments. Please write complete grammatically correct sentences and avoid spelling mistakes.

In addition, you could also hand in the log files created via the **diary** command, eventually after some proper editing. Typing **diary** followed by the name of a file in a MATLAB session creates a new file with the given name which will contain everything you see on the screen during the session. While this file created by **diary** is by no means a substitute for the other items in the project solution, it could help you to gain some (partial) credit when some answers are (partially) wrong.

See http://www.math.uic.edu/~jan/MCS471 for the hypertext version of this project.

If you have questions, comments, or difficulties, feel free to come to my office for help.