

MCS 471 Project Two : roots of polynomials

The goal of this project is to investigate numerical algorithms to find all roots of a polynomial equation. In our experiments we will use MATLAB or Octave. MATLAB is available in all computer labs on campus, but is commercial and expensive. Octave is similar to MATLAB (for our work, Octave runs just as fine), and is free to download from <http://www.octave.org/>.

0. Polynomials in MATLAB or Octave

Polynomials are represented by their coefficient vectors. For example, the cubic polynomial $3.3x^3 - 2.23x^2 - 5.1x + 9.8$ is entered typing `c = [3.3 - 2.23 - 5.1 9.8]` at the prompt. With `y = polyval(c, x)`, we evaluate the polynomial at some point or vector x . After typing `r = roots(c)`, the vector r will contain approximations for the roots of the polynomial. The complement to the `roots` command is `poly`, e.g. `c = poly(r)` returns the coefficient vector of the monic polynomial which has its roots in r .

We can group commands into `.m` files, defining functions. The name of the function should match the file name. For this project, we need the function “weierstrass”, with its definition in the file “weierstrass.m” (download it from the class web site). If the path is set correctly (do `help path` otherwise to see how to set the search path), then you can call this function just as a regular command.

1. The method of Weierstrass (a.k.a the Durand-Kerner method)

The method of Weierstrass simultaneously approximates all roots of a polynomial p of degree d , assumed to be monic, i.e.: the coefficient with the highest degree term x^d of p equals one. Starting at some initial approximation for all the roots in $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_d^{(0)})$, the method uses the following formula:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{p(x_i^{(k)})}{\prod_{\substack{j=1 \\ j \neq i}}^d (x_i^{(k)} - x_j^{(k)})}, \quad \text{for } i = 1, 2, \dots, d, \quad \text{and } k = 0, 1, \dots$$

This formula is implemented in the function `weierstrass`, which is called as follows:

```
>> p = randn(1,6); % degree 5 polynomial
>> p = p/p(1); % make monic
>> x = randn(1,5) + i*randn(1,5); % 5 random complex numbers
>> [x,res] = weierstrass(p,x,1.0e-10,30)
```

The last command returns in `x` the new vector of approximations for the roots of `p` after no more than 30 iterations. The method stops earlier if the desired accuracy of `1.0e-10` is reached. The `res` is the magnitude of `p` evaluated at `x`. If `res` is smaller than the desired accuracy, the method succeeded, otherwise it failed.

2. Numerical experiments on random and special polynomials

In the following assignments we study the finding of the roots for three classes of polynomials: random polynomials, polynomials with multiple roots, and ill-conditioned polynomials.

2.1 Global and local convergence of the method. We generate random polynomials to investigate experimentally the global and local convergence of the method. For the global convergence, we just generate random coefficient vectors as in the example above. To examine the local convergence, we generate random roots to define the test polynomials. We then start the method at points close to the generated roots.

Assignment One. Generate random coefficient vectors for polynomials of degree $d = 2, 3, \dots, 8$ and call the method, starting at randomly chosen initial approximations for the roots. Take as desired accuracy each time $1.0\text{e-}12$ and the maximum number of steps equal to 100. Do three random experiments for every d . Record in a table (rows indexed by d and columns by the trial: one, two, three) the number of steps it takes to reach the desired accuracy. *What can you say about the global convergence of the method?*

Assignment Two. Generate random complex vectors for the roots of polynomials of degree $d = 2, 3, \dots, 8$ and call the method, starting at initial approximations at a distance from the generated roots equal to $1.0\text{e-}2$, $1.0\text{e-}4$, and $1.0\text{e-}8$. An initial approximation \mathbf{x} at distance \mathbf{eps} from the generated roots \mathbf{r} can be generated as $\mathbf{x} = \mathbf{r} + (\text{randn}(1,d) + i * \text{randn}(1,d)) * \mathbf{eps}$. So for each polynomial we do three experiments, taking as desired accuracy each time $1.0\text{e-}12$ and the maximum number of steps equal to 100. Record in a table (rows indexed by d and columns by $1.0\text{e-}2$, $1.0\text{e-}4$, and $1.0\text{e-}8$) the number of steps it takes to reach the desired accuracy. *What can you say about the local convergence of the method?*

2.2 Multiple roots. The command `poly(ones(1,d))` returns the coefficient vector of a polynomial which has only one root with multiplicity d at $x = 1$, i.e.: $p(x) = (x - 1)^d$.

Assignment Three. Take $p(x) = (x - 1)^d$, for $d = 2, 3, \dots, 8$ and run the method with a desired accuracy equal to zero and maximum allowed steps equal to 1000, each time starting a random complex initial approximations of the roots. Make a triangular table, with rows indexed by d , the distance of each computed approximation to the multiple root one. *What is the relationship between the multiplicity d of the root one and the distance of the computed approximations to one?*

2.3 A perfidious polynomial. An easy looking test polynomial is $p(x) = (x - 1)(x - 2)(x - 3) \dots$. We can generate its coefficient vector as `poly(1:d)`, for any degree d .

Assignment Four. Run the method of Weierstrass on this polynomial for increasing values of d , starting at $d = 2$. Each time record the value for the residual `res`, for a desired accuracy of $1.0\text{e-}12$, and allowing no more than 100 iterations, starting at random initial approximations for the roots. Stop at that degree d for which `res` is larger than 1. *Explain why this polynomial is so ill conditioned.*

3. Deadline is Wednesday 1 October, at 1PM

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

1. Tables with numerical data, numbers formatted in scientific format.
2. Answers to the questions in the assignments. Please write complete grammatically correct sentences and avoid spelling mistakes.
3. Sequences of commands used to generate the data for the experiments. It is good practice to store these commands in little `.m` files, e.g., in `assignment_one.m`, `assignment_two.m`, etc.
4. Output of your sessions with MATLAB or Octave as *an appendix*. Typing `diary` followed by the name of a file in a session creates a new file with the given name which will contain everything you see on the screen during the session. You may edit out mistakes from the output of `diary`.

See <http://www.math.uic.edu/~jan/mcs471/index.html> for the hypertext version of this project and for the function `weierstrass.m`.

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