

MCS 471 Project Two: Graeffe's Root-Squaring Method

The goal of the project is to study the method of Graeffe to compute all roots of a polynomial. The method involves repeated squaring and was invented independently by Dandelin and Lobachevsky.

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 $y = \mathbf{polyval}(c, x)$, we evaluate the polynomial at some point or vector x . After typing $r = \mathbf{roots}(c)$, the vector r will contain approximations for the roots of the polynomial. The complement to the **roots** command is **poly**, e.g. $c = \mathbf{poly}(r)$ 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 “`graeffe`” (and two subroutines “`flip`” and “`even`”) with its definition in the file “`graeffe.m`” (download it and “`flip.m`” and “`even.m`” 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. Squaring Separates Roots

We present the idea of the method with a cubic monic polynomial $f(x)$ having roots r_1, r_2 , and r_3 . Consider the product of $f(x)$ with $f(-x)$:

$$f(x) = (x - r_1)(x - r_2)(x - r_3) \quad (1)$$

$$f(-x) = (x + r_1)(x + r_2)(x + r_3)(-1)^3 \quad (2)$$

$$\text{so } f(x)f(-x) = (-1)^3(x^2 - r_1^2)(x^2 - r_2^2)(x^2 - r_3^2). \quad (3)$$

The squaring of the roots will separate them in a way we can derive approximations for the roots directly from the coefficients. To achieve a clear separation, squaring once will often not be enough, we consider a sequence of polynomials and variables:

$$x_{k+1} = x_k^2, x_0 = x \quad \text{and} \quad f_{k+1} = f_k(x_k)f_k(-x_k), f_0 = f. \quad (4)$$

After squaring n times, we obtain

$$f_n(x_n) = (-1)^{3n}(x_n - r_1^{2^n})(x_n - r_2^{2^n})(x_n - r_3^{2^n}). \quad (5)$$

Renaming $g = f$ and $y = x_n$, we have the following general relations between the coefficients of $g(y)$ and its roots. In particular, $g(y) = y^3 + b_1y^2 + b_2y + b_3 = (y - s_1)(y - s_2)(y - s_3)$ leads to the system

$$\begin{cases} s_1 + s_2 + s_3 = -b_1 \\ s_1s_2 + s_1s_3 + s_2s_3 = b_2 \\ s_1s_2s_3 = -b_3 \end{cases} \quad (6)$$

Note that $s_1 = r_1^{2^n}$, $s_2 = r_2^{2^n}$, and $s_3 = r_3^{2^n}$. Due to repeated squaring, the roots have become very well separated. Sorting the roots if necessary, we may assume $|s_1| \gg |s_2| \gg |s_3|$. Thus the above system then becomes

$$\begin{cases} s_1 \approx -b_1 \\ s_1s_2 \approx b_2 \\ s_1s_2s_3 = -b_3 \end{cases} \quad \text{or} \quad \begin{cases} s_1 \approx -b_1 \\ s_2 \approx -b_2/b_1 \\ s_3 \approx -b_3/b_2 \end{cases} \quad \text{thus} \quad \begin{cases} r_1 \approx s_1^{1/2^n} \\ r_2 \approx s_2^{1/2^n} \\ r_3 \approx s_3^{1/2^n} \end{cases} \quad (7)$$

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 Convergence of the Method. We generate random polynomials to investigate experimentally the convergence of the method.

Assignment One. Generate random polynomials with roots in $[0, 1]$, using $\mathbf{p} = \mathbf{poly}(\mathbf{rand}(\mathbf{1}, \mathbf{d}))$, for degrees $\mathbf{d} = 2, 3, \dots, 8$. Take as desired accuracy each time $1.0\mathbf{e-8}$ 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 convergence of the method?

2.2 Multiple roots. Consider a polynomial which has only one root with multiplicity d at $x = 1$, i.e.: $p(x) = (x - 1)^d$.

Assignment Two. Take $p(x) = (x - 1)^d$, using the command $\mathbf{p} = \mathbf{poly}(\mathbf{ones}(\mathbf{1}, \mathbf{d}))$, for $\mathbf{d} = 2, 3, \dots, 10$. Run the method as in assignment one, taking $1.0\mathbf{e-8}$ as desired accuracy and 100 as the number of steps.

Make a triangular table, with rows indexed by d , the distance of each computed approximation (there are d columns in row d , one column for each root) 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?

For the same polynomials $p(x) = (x - 1)^d$, $d = 2, 3, \dots, 10$, run the method with a desired accuracy equal to zero and the maximum allowed steps equal to 100.

Describe what happens when Graeffe is allowed to run long enough on $p(x) = (x - 1)^d$. Can you explain your observations?

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 using $\mathbf{poly}(\mathbf{1}; \mathbf{d})$, for any degree \mathbf{d} .

Assignment Three. Run the method of Graeffe on this polynomial for increasing values of d , starting at $d = 2$. Each time compute and record the maximal distance between the exact root and its corresponding approximation. Use $1.0\mathbf{e-8}$ as desired accuracy. Limit the maximum number of iterations to avoid overflow. Stop at that degree d for which the method is no longer able to compute more than two significant decimal places of each root.

Explain why it is so hard to compute all roots of this polynomial.

3. Deadline is Wednesday 16 February 2005 at 10AM

Bring *your* solution to the project to class. The *your* is emphasized to stress that your solution is the result of an *individual* effort. Collaborations are **not** permitted.

Your solution should contain the following:

1. Answers to the questions in the assignments. Please write complete grammatically correct sentences.
2. Tables summarizing the numerical experiments you have done. Use the **format short e** command when generating data, so your numbers in the tables will also be in scientific notation.
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.

The solution to the project is essentially a report on paper.

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