Assignments 1 & 2

CSC 2307 — Numerical Software

Assignment 1 due 22 February 2010. Assignment 2 due 15 March 2010.

Assignments 1 and 2 are each worth 30% of your final grade.
(Recall, however, that you can substitute a course project for one of the assignments.)

The following three articles appeared on the *Internet* several years ago. The fourth and fifth ones appeared a little later in the *NA Digest*.

First Article.

I would appreciate constructive suggestions concerning fast stable methods for determining the real zeros of a polynomial with real coefficients. I know of stable but slow methods and of fast methods that occasionally run amok. The polynomial in question is of degree four, if that helps. More general solutions might be useful also. For the purpose of discussion denote the polynomial by

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

where the a_i are all real. What is wanted is one or more real values x such that $P(x) = 0$.

Second Article.

Polynomials of degree 4 or less have algebraic solutions. Any good handbook like CRC will show you how to do it. The solution below came from *"Mathematical Handbook of Formulas and Tables"* by Murray Spiegel, 1968, in the *Schaum's Outline Series*.

Unfortunately, there are some problems you have to watch out for in trying to implement code for this solution. If two (real) roots are very close, or if a complex root is very close to the real line, numerical accuracy problems rear their nasty heads. It is easy for code to decide that there are no real solutions, when there are some, and vice versa. The first time I naively used code for the algorithm below to solve $(x-1)^4 = 0$ on a VAX, it decided that all four roots were complex. A typical root was $1 + 10^{-4} i$.

One possibility is to use the solution below to find points known to be close to the roots, and then use them as starting guesses for an iterative procedure that works in the complex plane. Since the starting guess will be off only by the numerical errors in computing the formulas below, this should be extremely fast for any reasonable iterative algorithm.

Algebraic Solution of Quartic Polynomials.

First normalize $P(x)$ so that the coefficient of x^4 is 1:

$$P(x) = x^4 + ax^3 + bx^2 + cx + d.$$

Let y_1 be a real root of the cubic

$$y^3 - by^2 + (ac - 4d)y + (4bd - c^2 - da^2) = 0. \tag{1}$$

The roots of $P(x)$ are the four roots of the two quadratics

$$z^2 + \frac{a \pm \sqrt{a^2 - 4b + 4y_1}}{2}\, z + \frac{y_1 \mp \sqrt{y_1^2 - 4d}}{2} = 0 \tag{2}$$

where, if you take the first ± to be +, you should take the second ∓ to be −, and vice versa. If all roots of (1) are real, pick y_1 to be the root that gives all real coefficients in (2), if possible.

Algebraic Solution of Cubic Polynomials.

To solve the cubic equation

$$x^3 + ax^2 + bx + c = 0,$$

let

$$Q = \frac{3b - a^2}{9}, \qquad R = \frac{9ab - 27c - 2a^3}{54}, \qquad D = Q^3 + R^2,$$

and

$$S^3 = R + \sqrt{D}, \qquad T^3 = R - \sqrt{D}.$$

Then the roots of the cubic are given by

$$\begin{aligned} x_1 &= S + T - \frac{a}{3}, \\ x_2 &= -\frac{S+T}{2} - \frac{a}{3} + i\frac{(S-T)\sqrt{3}}{2}, \\ x_3 &= -\frac{S+T}{2} - \frac{a}{3} - i\frac{(S-T)\sqrt{3}}{2}. \end{aligned}$$

where $i = \sqrt{-1}$. If a, b, c are real, then

1. one root is real and two are complex conjugates if $D > 0$,

2. all roots are real and at least two are equal if $D = 0$, and

3. all roots are real and unequal if $D < 0$.

Third Article.

Hello, does anyone know of a good, robust, method to find all roots of a quartic equation? I am a computer graphics person and am writing code to intersect a ray with a quartic (as part of a ray tracing program). The problem boils done to finding valid roots of a quartic. I have heard that direct solution is prone to numerical instability, so an iterative technique is called for. I do have a way of getting good initial guesses to start a rather slow iterative method like *regula falsi*, but I'm looking for something that converges as quickly as possible. Perhaps a hybrid method of some kind.

Any code, pseudo-code, or description of algorithm would be much appreciated. I intend to post a synopsis to the comp.graphics group (and this one if someone wishes).

Fourth Article.

From the *NA Digest*, Saturday, August 24, 1991, Volume 91: Issue 34.

From: Herman J. Woltring `<UGDIST@NICI.KUN.NL>`
Date: Mon, 19 Aug 91 07:59 MET
Subject: Cubic & Quartic Equation Solvers

Dear NA-NET readers,

I should be grateful for FORTRAN code to solve (cubic and) quartic equations. It seems that the algorithm in section 3.8.2 in my edition of Abramovitz & Stegun is incorrect, while I am getting good results with the algorithm in section 5.5 of "Numerical Recipes" by Press et al. However, rounding errors in the quartic equation algorithm of section 3.8.3 in Abramovitz and Stegun make it difficult to take reliable branching decisions.

Thanks in advance!
Herman J. Woltring `<na.woltring>`, Eindhoven, The Netherlands.

Fifth Article.

In the next issue of *NA Digest*, Woltring published the following note from Dan Lozier.

Date: Wed, 28 Aug 91 16:26:20 EDT
From: lozier@scuba.cam.nist.gov
Subject: Section 3.8.2
To: ugdist@NICI.KUN.NL

My colleague Marjorie McClain and I have examined Section 3.8.2 and we can explain why results computed by your program are sometimes incorrect. The branch of the cube root that is taken is very important. It should be the principal branch except for a cube root of a negative number, in which case the negative real cube root should be taken. The Fortran expression C**(1.0/3.0) where C is of type COMPLEX always produces the principal branch. Your program works when modified to use arithmetic of type REAL for the computation of cube roots when the discriminant is nonnegative.

In conclusion, we think we can say the formulas in Section 3.8.2 are correct but for algorithmic usage they are slightly ambiguous in that the precise determination of the cube root is not given.

Dan Lozier

A few other *Internet* articles (that you may find interesting but not particularly helpful for this assignment) are stored in ~krj/quadratic/quartic.news on the CDF system.

First Assignment.

Write a Fortran subroutine to find the roots of a general cubic

$$c(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

where the a_i are all real coefficients. Do NOT assume that $a_3 = 1$; in fact, don't even assume that $a_3 \neq 0$. That is, your routine should handle degenerate cubics in a reasonable way. See the instructions concerning IFLAG below.

Although you should attempt to make your subroutine as portable as possible, you may assume that the machine on which it is to run uses floating-point numbers that conform to the IEEE standard. In fact, this assumption is built into some of the requirements below.

A major part of this assignment is determining a *specification* for your subroutine. One should be stated clearly in the comments at the beginning of your code. It is not necessary — or even desirable — to give as stringent a specification as possible, since meeting it will necessitate developing a very large and complicated Fortran subroutine. However, your specification should be "reasonably" stringent and must take into account rounding errors as well as the possibility of floating-point overflows and underflows. Your argument that your subroutine meets the specification is not meant to be a complete proof, but it should be convincing to anyone evaluating your code — in this case, me and two other CSC 2307 students. In addition, run your program on an extensive set of test problems. Include a sample of your numerical results as external documentation that your routine meets its specification.

To ensure uniformity throughout the class for the purpose of testing your subroutines, it must be written **completely in single precision**[1] and it must have the following declaration

SUBROUTINE CUBIC (A, RR, RI, IFLAG)

where

A is a REAL array of dimension 0:3 such that A(I) holds the coefficient a_i of the cubic $c(x)$ above; you should not change the input value of A(I);

RR, RI are both REAL arrays of dimension 1:3 that hold the real and imaginary parts, respectively, of each calculated root — RR(I) is the real part of root I and RI(I) is its imaginary part;

IFLAG is an INTEGER array of dimension 1:3; for each I $\in \{1, 2, 3\}$, IFLAG(I) takes on one of the following values:

1 if the root returned in RR(I) and RI(I) meets the subroutine's specification;

0 if the cubic has infinitely many roots (i.e., A(3) = A(2) = A(1) = A(0) = 0); in this case, the values returned in RR and RI are arbitrary;

–1 if the I$^{\text{th}}$ root does not exist because the "true degree" of $c(x)$ is $<$ I; in this case, the values returned in RR(I) and RI(I) are arbitrary (note this case includes A(3) = A(2) = A(1) = 0 but A(0) $\neq$ 0);

–2 if any of the coefficients A(I) are not floating point numbers, (e.g., they may be ± Infinity or NaN), (for this purpose consider denormalized numbers to be floating point numbers but be careful about guaranteeing the accuracy of your results);

–3 if the "true" I$^{\text{th}}$ root is outside the range of normalized floating-point numbers (including zero); in this case, RR(I) and/or RI(I) may be ± Infinity, a denormalized number or zero (where, in the last case, the "true root" is not zero);

–4 if the "true" I$^{\text{th}}$ root is within the range of normalized floating-point numbers (including zero), but at least one of RR(I) and RI(I) cannot be guaranteed to meet the subroutine's specification; in this case RR(I) and RI(I), which may be any IEEE floating point value except NaN, contain a "rough" approximation to root I;

–5 if you cannot guarantee that any of the IFLAG values above would be correct, but you have calculated a "rough" approximation to root I, which may be any IEEE floating point value except NaN,

–6 if you cannot guarantee that any of the IFLAG values above would be correct and you have not calculated a "rough" approximation to root I; in this case the values returned in RR(I) and RI(I) are arbitrary.

[1] Except that you may use the subroutine quad1.f in ∼krj/quadratic on the CDF system to solve quadratic equations; quad1.f uses double precision to compute the discriminant accurately.

The second article above is meant to be a starting point for this assignment, but be warned that several suggestions made in it may lead to a poor solution: free advice is often worth just what you pay for it.

The first few years I gave this assignment, most students attempted to implement non-iterative methods, putting a lot of effort into essentially an extended-precision arithmetic package in an attempt to compute very accurate roots. This lead to very large, cumbersome, inefficient subroutine which, in the end, often failed to achieve its author's accuracy goals. I would prefer to see a clean, robust, reliable, efficient subroutine that is "as a accurate as possible" given your single precision constraint and the conditioning of the problem class. To this end, you are encouraged to look through any references you choose for formulas and solution techniques. You should note your sources in your inline documentation, and give some reasons for your algorithmic choices in either your inline or external documentation. In conjunction with the second part of this assignment, I think it will be instructive to compare a variety of well-implemented solution techniques for this problem written by different students.

You may find it helpful to look at my quadratic equation solvers and test routine in ~krj/quadratic on the CDF system. Be warned, though, that my older version, quad2.f, is very poorly documented; the newer one, quad1.f, is a little better, but still does not meet the standards I expect of you. Moreover, quad2.f is far too "cautious" for most applications, using a variety of special techniques to avoid the possibility of overflow and underflow. The style of programming in quad1.f should serve as a better model for this assignment. You should take reasonable precautions to avoid overflows and "disastrous" underflows, but, in the spirit of the IEEE floating-point standard, "harmless" underflows are quite acceptable in this assignment. As noted in footnote 1, you may use quad1.f in your program to solve quadratics.

One approach that might be interesting to explore is to solve the problem in a straightforward manner, but at select points throughout your program check if the computation has incurred a floating-point exception or has become inaccurate. If it has, switch to a more robust, but possibly much slower, method. This approach mimics exception handling (which we will discuss in class) but without appropriate language support.

The ~krj/quadratic directory also contains some other codes related to VAX and SUN machine arithmetic and floating point exceptions that you may find useful. More details on IEEE floating-point numbers and arithmetic can be found in one of the CSC 2307 course folders as well as in the SUN documentation. Also, further information on quadratic and polynomial root finders can be found in another CSC 2307 course folder.

One last point, the S and T used in the solution of the cubic should either both be real or else a complex conjugate pair (i.e., $S = \bar{T}$, with the imaginary part of $S \neq 0$).

Second Assignment.

Please bring 3 copies of your program and external documentation to class on February 22. One copy is for me and the other two are for the two students who will evaluate your subroutine. Please also make your subroutine, test examples, external documentation, etc.

available in machine readable form. The easiest way to do this is to leave all your files for this assignment in a subdirectory called *quadratic* on the CDF system. Make the directory and files readable by everyone after all the assignments have been submitted to me. (Keep them protected until then.)

Each student will write an evaluation of two other students' subroutines. In addition to considering the usual aspects of good software design, the appropriateness of the specification should be evaluated as well as whether the code actually meets it. In attempting to determine the latter, I recommend that you run the subroutine on some carefully chosen test problems in addition to examining the code.

Your evaluations are due in class on March 15. Each should be five to ten pages long (with approximately as many words per page as this assignment). Append to each an annotated listing of the test results that you used in your evaluation. Please give a copy of both evaluations to me as well as a copy of your evaluation of each student's cubic solver to that student. (Because of the latter requirement, each evaluation should be self contained.) Please leave all the files that you used in preparing your evaluation in a subdirectory called *evaluation* on the CDF system. Make the directory and files readable by everyone after all the assignments have been submitted to me. (Keep them protected until then.)

Leave all your files for these two assignments in the directories *quadratic* and *evaluation* until I return the assignment to you.

Some References.

You might find some of the following references useful. I will put a class folder containing photocopies of several of the papers listed below in the Engineering and Computer Science Library (which is located on the second floor of the Sandford Fleming Building). I believe all references can be found in the UofT Library system. Many other papers and texts may be helpful to you also. I encourage you to look through a few. Reference the ones that you use in your documentation for your cubic solver.

1. D. A. Adams, "A stopping criterion for polynomial root finding", *CACM*, 10 (1967), pp. 655–658.

2. G. E. Forsythe, "What is a satisfactory quadratic equation solver?", *Constructive aspects of the fundamental theorem of algebra*, Proceedings of a symposium conducted at the IBM Research Laboratory, Zurich-Ruschlikon, Switzerland, June 5–7, 1967, edited by Bruno Dejon and Peter Henrici, pp. 53–61.

3. M. A. Jenkins and J. B. Traub, "Principles for testing polynomial zerofinding programs", *ACM Transactions on Mathematical Software*, 1 (1975), pp. 26–34.

4. J.M. Mcnamee, "An updated supplementary bibliography on roots of polynomials", *J. Computational and Applied Mathematics*, 11 (1999), pp. 305–306. See also the webpage http://www.elsevier.nl:80/inca/publications/store/5/0/5/6/1/3/

5. W. Miller, *The Engineering of Numerical Software*, Prentice-Hall, 1984.

6. G. Peters and J. H. Wilkinson, "Practical problems arising in the solution of polynomial equations", *J. Institute of Mathematics and Its Applications*, 8 (1971), pp. 16–35.

7. W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes — The Art of Scientific Computing*, Cambridge University Press, 1986.

8. Murray Spiegel, "Mathematical Handbook of Formulas and Tables" *Schaum's Outline Series*, 1968.

9. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, 1963.

10. J. H. Wilkinson, "The perfidious polynomial", in *Studies in Numerical Analysis*, G. H. Golub ed., Mathematical Association of America, 1984, pp. 1–28.