To Solve a Real Cubic Equation

(Lecture Notes for a Numerical Analysis Course)

W. Kahan
Mathematics Dep't
University of California
Berkeley CA 94720

Nov. 10, 1986

Abstract: A program to solve a real cubic equation efficiently and as accurately as the data deserve is not yet an entirely cut-and-dried affair. An iterative method is the best found so far. This method plus some other issues, like accuracy, scaling, preconditioning and testing, are discussed in these notes in enough detail to convey an impression of what Numerical Analysis is about.

1. Introduction:

Closed-form formulas for solving the real cubic equation

$$Ax^3 + Bx^2 + Cx + D = 0$$

in terms of its coefficients A, B, C, D were discovered in the sixteenth century by Italian mathematicians, but their triumph turned into disappointment when they discovered an *irreducible case*: the real cubic with three irrational real roots. This case entails unavoidably the computation of trigonometric functions and their inverses during the evaluation of cube roots of a complex number. Nowadays trigonometric functions and complex numbers seem unobjectionable in a procedure that solves a cubic, so they have been used freely in a modern version of the Italians' formulas presented below in §2 of these notes. Alas, the modern formula is disappointing too, because it is potentially unstable in the face of roundoff. Indeed, coefficients abound for which some of the roots computed from the formula are quite incorrect; several instances appear among the examples presented in §10 .

Whether a slight modification could protect the Italians' formulas from the worst effects of roundoff remains an open question. The simplest stable version of those formulas I know is tantamount to evaluating them twice, as is mentioned near the end of §2 . Two evaluations take long enough to make plausible the possibility that another approach might be faster.

Newton pioneered another approach when he first used the iteration that now bears his name to solve a cubic. Computers can follow his approach provided certain details like where to start and when to stop are mechanized. Those details are the subject of §3 , a long discussion that culminates in a brief but entirely automatic procedure presented as a program QBC in §4 . That discussion provides merely a motive for the program, not a proof of its correctness. A thorough proof would be far too lengthy to include in these notes. Instead, the issues that such a proof would have to address will be explored and its conclusions summarized.

The most difficult issue is inaccuracy caused by roundoff. Error analysis proves that every root computed by QBC is no more in error than if it had been computed exactly from a cubic whose coefficients differ from those given each by a few units in its last digit carried by the computer's floating-point arithmetic. This kind of *Backward Error Analysis* was first published in the late 1950's by James H. Wilkinson. It suggests that inaccuracy introduced by the process of solving the cubic is unlikely to be appreciably worse than inaccuracies previously introduced when the coefficients were computed and rounded off. Therefore, if roots obtained from QBC turn out too inaccurate for some ulterior purpose, the trouble may lie not so much with QBC as with the process that generated the coefficients. Thus does backward error analysis exculpate the programmer of QBC. And it does more.

The uncertainty contributed to the computed roots by roundoff in QBC can now be assessed by analyzing the effects upon those roots of tiny perturbations of the cubic's coefficients, regardless of the internal details of QBC . Even without those details, the analysis is tedious; only its conclusions are summarized in §5 . Computed roots turn out normally to be accurate in all but their last few digits; but in worst cases, when all three roots of the cubic almost coincide, the computed roots can lose as many as two thirds of the figures carried. Examples in §5, §7 and §10 bear out this gloomy prediction, to which we shall return later.

Besides being too long to include in these notes, the proofs of the foregoing claims to accuracy are at least as vulnerable to error as the short program they are supposed to vindicate. Such claims deserve credence only if they are supported by numerical experiments. But rounding errors committed during the experiments can confound the test results and obscure their implications. §6 discusses such issues and offers a partial remedy in the form of a program REVAL that combines the evaluation of a cubic polynomial with the simultaneous calculation of a rigorously correct bound for the effect of roundoff upon that evaluation. REVAL is based upon prior knowledge of a bound for the rounding error in every floating-point arithmetic operation; that bound is characteristic of the computer and deducible from attributes like the number of significant digits it carries. REVAL and programs like it permit the error in a computed root, regardless of its provenance, to be overestimated with ease as rigorously as one likes and without excessive pessimism provided the root lies far enough away from all the others. Clustered roots are a little harder to handle.

The previous two paragraphs may suggest (and it's widely believed) that clustered roots of a cubic cannot be calculated accurately unless arithmetic is performed carrying about three times as many significant figures as will be assuredly correct in the computed roots. That is untrue. Also untrue is another widely believed myth about numerical computation, namely that numerical error is *caused* by cancellation. In fact, on almost all modern computers, no new error is generated when subtractive cancellation occurs; the principal exceptions are CRAYs, CYBERs and UNIVACs. On IBM 370's, DEC VAX's, SUN's, APPLE Macintoshes and Hewlett-Packard calculators, to mention just a few, subtractive cancellation is exact. This fact can be exploited to *Precondition* a cubic with clustered roots, transforming it into a new cubic with relatively well separated roots that are easy to calculate and transform back into fully accurate roots of the original equation. A simplified version of preconditioning, applicable principally to cubics with integer coefficients, is described in §7 with examples that may suggest how the process would work in general. Thus have we confronted two myths about roundoff and cancelled them both.

After roundoff, the second hazard to be overcome during numerical computation is spurious over/underflow, an event that occurs when intermediate results would be so huge or so tiny as to lie outside the range of numbers normally representable in the computer even though the desired final results lie within range. This hazard is encountered only rarely, and then it can be overcome by *Scaling*, which is described in §8 .

The final few sections of these notes are archival. §9 presents a collection of cubics with known zeros that help to test programs like QBC or its competitors. §10 exhibits selected but typical results obtained from our versions of the Italians' formula and of Newton's iteration (QBC) programmed into an HP-15C handheld calculator. The program for QBC is supplied in §11, and the running times for both methods are compared briefly in §12 .

2. A Formula in "Closed Form" :

A cubic polynomial $Ax^3 + Bx^2 + Cx + D$ has three zeros $x = x_1$, x_2, x_3 that can be expressed explicitly in terms of its given coefficients A, B, C, D in many ways. The formula chosen below is one of the better ones, and has been arranged in the form of an algorithm that can easily be programmed into a computer:

```
A, B, C and D  are given real numbers.
If  $A = 0$
   then { $x_3 := (|B| + |C| + |D|)/A$ ; ...   $\infty$  or  $0/0$ .
             $p := -C/2$ ; ...    Next solve   $Bx^2 - 2px + D = 0$  ...
             $q := \surd(p^2 - B D)$ ; ... possibly an imaginary number.
             if  q is Real  ...,  in which case  $q \geq 0$ ,  ...
                then { $r := p + \mathrm{sign}(p)q$ ;  ... $= p \pm q$  ...
                          if  $r = 0$
                             then { ... Zeros are  0 or $\infty$ or  $0/0$ . ...
                                        $x_1 := D/B$ ;  $x_2 := -x_1$  }
                             else { $x_1 := D/r$ ;  $x_2 := r/B$  }  }
                else { $x_1 := p/B + q/B$ ;  $x_2 := p/B - q/B$  }  }
   else { $b := -(B/A)/3$ ;  $c := C/A$ ;  $d := D/A$ ;
             ...   Now solve   $x^3 - 3bx^2 + cx + d = 0$   ...
             $s := 3b^2 - c$ ;
             $t := (s - b^2)b - d$ ;
             ...   Now  $x = b - y$  where  $sy - y^3 = t$   ...
             if  $s = 0$
                then { $y_1 := -t^{1/3}$ ;  ...  the real cube root.
                          $y_2 := y_1(-1 + \imath\surd 3)/2$   }
                else { $u := \surd(4s/3)$ ; ...  possibly imaginary.
                          $v := \arcsin((3t/s)/u)/3$ ; ...  may be complex.
                          $w := (\pi/3)\mathrm{sign}(\mathrm{Re}(v)) - v$ ; ... $= \pm\pi/3 - v$ ...
                          $y_1 := u\sin(v)$ ;  $y_2 := u\sin(w)$   } ;
             $x_1 := b - y_1$ ;  $x_2 := b - y_2$ ;  $x_3 := y_1 + y_2 + b$   } .
```

This algorithm was programmed into an HP-15C calculator without difficulty. On many another machine programming might be impeded by the absence of complex *sin* and *arcsin* from its library of elementary functions. Then the following formulas may help:

If $z^2 > 1$ then $\arcsin(z) = (\pi/2 - \imath\,\mathrm{arccosh}(|z|))\,z/|z|$.
If z is real, $\arcsin(\imath z) = \imath\,\mathrm{arcsinh}(z)$,
$\cos(\imath z) = \cosh(z)$, and
$\sin(\imath z) = \imath\sinh(z)$. ($\imath = \sqrt{-1}$)

With the aid of these formulas and some algebraic manipulation, the algorithm can be freed from all nontrivial complex arithmetic, but only at the cost of introducing more case analysis. In place of the formulas involving complex *arcsin* and *sin*, there will be three cases. One case handles $s < 0$. If $s > 0$ (in which case $u > 0$ too), there are two more cases according to where $|3t/(su)|$ lies relative to 1 . But multiplying cases can only exacerbate the first of *three flaws* that mar the algorithm:

First, the algorithm is complicated, and therefore vulnerable to oversights. Have all singularities been considered and handled correctly?

Second, the algorithm is vulnerable to over/underflow. Even when all three zeros lie well within range, over/underflow can blight the intermediate quantities q, r, s and t . The natural defense against over/underflow is *scaling* , another complication.

Third, the algorithm is vulnerable to roundoff, particularly when the zeros are of wildly different magnitudes; then the zeros of smaller magnitude tend to be computed relatively inaccurately. (Examples of inaccuracy can be found at the end of these notes.) All figures can be lost in any zero whose magnitude is smaller than a rounding error in b . One way to calculate the tiniest zero more accurately is to obtain it as the reciprocal of the biggest zero of $A + Bz + Cz^2 + Dz^3$, which is tantamount to running the foregoing algorithm a second time. To compute the zero of middle magnitude, divide $-D/A$ by the other two zeros.

Another way to improve the accuracy of a zero is to use some kind of iteration that improves approximate zeros by exploiting the cubic's behavior near them; a short step past this thought finds us contemplating whether the cubic might be better solved by an altogether iterative method than by explicit formulas. Just such an iteration is the next topic discussed in these notes.

3. Newton's Iteration:

Given the real cubic polynomial $Q(x) := Ax^3 + Bx^2 + Cx + D$, we may use iteration $X_{n+1} := X_n - Q(X_n)/Q'(X_n)$ for $n = 0, 1, 2, \ldots$ to find a real zero of $Q(x)$ provided we can solve four problems:

- How shall $Q(X)/Q'(X)$ be calculated efficiently?
- Where is a good place to choose the starting iterate X_0 ?
- When should the iteration be stopped?
- Having found one zero, how do we find the other two?

The following scheme computes $Q(X)$ and $Q'(X)$ at the cost of 4 multiplications per iteration:

$$q_0 := AX \; ; \quad q_1 := q_0 + B \; ; \quad q_2 := q_1 X + C \; ;$$
$$Q'(X) := (q_0 + q_1)X + q_2 \; ; \quad Q(X) := q_2 X + D \; .$$

Three preliminary divisions of all the coefficients of $Q(x)$ by A could subsequently save one multiplication per iteration, but doing so would exacerbate roundoff and raise questions about over/underflow, questions best answered by scaling all coefficients of $Q(x)$ in advance in a way to be discussed in §8 below.

Finding a good starting iterate X_0 is a balancing act among many contending considerations. First comes the numerical stability of the *deflation process* by which, after a real zero has been computed, it will be removed from the cubic to yield a quadratic whose zeros are the remaining two zeros of the cubic. The process of deflation is numerically stable unless the zero being removed is much tinier than one zero of the quadratic but much bigger than the other. X_0 can be chosen to avoid that unstable situation.

A second consideration is speed. Newton's iteration converges very quickly if started close enough to a simple zero, but converges very slowly to a multiple zero. Therefore, X_0 should ideally be extremely close to a triple zero, if $Q(x)$ has one, or else much closer to a simple zero than to a double zero if $Q(x)$ has both of those. Here is a way to choose such an X_0 :

Assuming $AD \ne 0$, let $b := -(B/A)/3$; $r := |Q(b)/A|^{1/3} \ge 0$; and $s := \mathrm{sign}(Q(b)/A) = \pm 1$. If $Q'(b)/A \ge 0$ then $X_0 := b - sr$ else $X_0 := b - 1.324718\, s \max\{r, \sqrt{(-Q'(b)/A)}\}$. Why does this choice work? The next paragraph will explain. To better follow its argument, read it repeatedly with reference to the graphs of, say, $x^3 + \rho x + 2$ for $\rho = -9, -3, -1, 0, 1$ and 3 superposed upon each other to show how its leftmost real zero increases with ρ . That leftmost zero is the goal of the iteration.

Why start iterating at X_0 ? Observe that $Q''(b) = 0$; therefore $x = b$ at the inflexion on the graph of $Q(x)$, and furthermore $Q(b-y) = Q(b) - Q'(b)y - Ay^3$. If $Q'(b)/A > 0$ then this cubic is strictly monotonic with just one real zero y that must lie between $y = 0$ and $y = sr$; otherwise the real zero y farthest from 0 lies beyond $y = sr$ and beyond $y = s\sqrt{(-Q'(b)/A)}$ too, but not beyond both λsr and $\lambda s\sqrt{(-Q'(b)/A)}$, where λ is the real root $\lambda = 1.32471\ 79572\ 44746...$ of $\lambda^3 = \lambda + 1$. Since the desired real zero X lies between the starting iterate X_0 and the inflexion point b , and the cubic is monotone between X and X_0, Newton's iteration converges monotonically and rapidly to the desired real zero. In the special case that $X_0 = b$ no further iteration will occur because then b is the cubic's triple zero.

When should the iteration $X_{n+1} := X_n - Q(X_n)/Q'(X_n)$ be stopped? Except when $X_0 = b$, we would expect $\mathrm{sign}(X_{n+1} - X_n) = s$ for all n ; but that expectation cannot persist indefinitely in the face of roundoff. Ultimately roundoff must cause $X_{n+1} - X_n$ to vanish or take the wrong sign, or cause $Q'(X_n)$ to vanish; in either case we shall set $X := X_n$ and accept it as a real zero of the cubic. Since any iteration could take too long to home in to $X = 0$, which occurs if $D = 0$, that case is segregated. And the quotient Q/Q' must be replaced by $(Q/Q')/1.000...001$ to overcompensate for roundoff that could otherwise carry X_n too far beyond its goal. When X is extremely tiny, that extra division prevents X_n from jumping over X to 0 , as otherwise it would in one of the examples in §10 . Roundoff can cause yet another kind of overshoot when the cubic's three zeros are closely clustered; X_n can fall between two zeros. We avoid the worst effects of this overshoot by accepting $X = X_n$ instead of X_{n+1} . Our policies for handling roundoff and stopping the iteration are not the only possibilities, but they are among the simplest.

With one real zero X in hand, the next task is *deflation* to obtain the quadratic $Ax^2 + B_1x + C_2$ whose zeros are the two remaining zeros of the cubic. Here are the deflation formulas:

```
  If $|X^3| > |D/A|$ then { $C_2 := -D/X$ ;  $B_1 := (C_2 - C)/X$ }
                      else { $B_1 := AX + B$ ;  $C_2 := B_1X + C$ }
                             ... ( recall $q_1$ and $q_2$ above ) ...
```

One formula for C_2 comes from the product of the cubic's zeros, $-D/A = X\,C_2/A$. The choice for B_1 was derived from an error-analysis that looked at the sum of the zeros, $-B/A = X - B_1/A$, and at the sum of their reciprocals, $-C/D = 1/X - B_1/C_2$, to find out which is least perturbed by the error in X . Of course, different formulas have to be used when $A = 0$ or $D = 0$.

Finally, formulas for solving a quadratic equation are taken from the algorithm presented earlier.

4. Iterative Algorithm QBC :

The following algorithm, arranged to facilitate programming, is complete except for scaling precautions against over/underflow. It is broken into subprocedures that make it easier to understand.

```
Real Function  DISC(a, b, c)  := $b^2 - a c$ ;
...  Later, during the discussion of Preconditioning in §7 ,
...  another version of DISC will be presented that is more
...  accurate when a, b, c are all integers and not too big.
     End DISC .
```

```
Procedure  QDRTC( A, B, C,  $X_1+\imath Y_1$, $X_2+\imath Y_2$ ):
...  Given real coefficients A, B, C , this procedure delivers
...  the two zeros $X_j+\imath Y_j$ of the quadratic $Ax^2 + Bx + C$ .
        $b := -B/2$ ;   $q :=$ DISC(A, b, C) ;
        If $q < 0$
           then { $X_1 := b/A$ ;  $X_2 := X_1$ ;
                  $Y_1 := \sqrt{(-q)}/A$ ;  $Y_2 := -Y_1$ }
           else { $Y_1 := 0$ ;  $Y_2 := 0$ ;
                  $r := b + \mathrm{sign}(b)\sqrt{q}$ ; ... $= b \pm \sqrt{q}$ .
                  If $r = 0$
                     then { $X_1 := C/A$ ;  $X_2 := -X_1$ }
                     else { $X_1 := C/r$ ;  $X_2 := r/A$ }  } ;
        Return ;  End QDRTC .
```

```
Procedure  EVAL( X,  A, B, C, D,  Q, Q', $B_1$, $C_1$ ):
...  Given real  X  and real coefficients  A, B, C, D  of the
...  cubic  $Q(x) = Ax^3 + Bx^2 + Cx + D$ ,  this procedure computes
...  $Q := Q(X)$ ,  $Q' := Q'(X)$ ,  $B_1 := AX + B$  and  $C_2 := B_1X + C$ .
         $q_0 := AX$ ;  $B_1 := q_0 + B$ ;  $C_2 := B_1X + C$ ;
         $Q' := (q_0+B_1)X + C_2$ ;  $Q := C_2X + D$ ;
         Return ;  End EVAL .
```

```
Procedure  QBC( A, B, C, D,  X, $X_1+\imath Y_1$, $X_2+\imath Y_2$ ):
...  Given real coefficients  A, B, C, D  of the cubic
...  $Ax^3 + Bx^2 + Cx + D$ ,  this procedure computes a real zero  X
...  and two complex zeros  $X_j+\imath Y_j$  of the cubic.
         If  A = 0  then { X := ∞ ;  A := B ;  $b_1$ := C ;  $c_2$ := D ;
                             go to fin } ;
         If  D = 0  then { X := 0 ;  $b_1$ := B ;  $c_2$ := C ;
                             go to fin } ;
         X := -(B/A)/3 ;  call EVAL(X, A,B,C,D, q, q', $b_1$, $c_2$) ;
         t := q/A ;  r := $\sqrt[3]{|t|}$ ;  s := sign(t) ;  ... = ±1 .
         t := -q'/A ;  if  t > 0  then  r := 1.324718 max(r, $\sqrt{t}$) ;
         $x_0$ := X - sr ;  if  $x_0$ = X  then  go to fin  ;
         Do  {  X := $x_0$ ;  call EVAL(X, A,B,C,D, q, q', $b_1$, $c_2$) ;
                if  q' = 0  then  $x_0$ := X
                            else  $x_0$ := X - (q/q')/1.000..001 }
             until  $sx_0 \le sX$  ;  ...  stop when  $x_0 \doteq X$ .
         If  $|A|X^2 > |D/X|$
             then  {  $c_2 := -D/X$ ;  $b_1 := (c_2 - C)/X$  };
   fin:  call  QDRTC( A, $b_1$, $c_2$,  $X_1+\imath Y_1$, $X_2+\imath Y_2$ ) ;
         Return ;  End QBC .
```

5. Accuracy:

A rigorous assessment of the effects of roundoff upon QBC would be too complicated to include in these notes, but the conclusions from such an assessment will be stated here, followed later in §6 ("Testing Considerations") and §7 ("Preconditioning") by some suggestions about what can be done about those effects.

Provided over/underflow does not intrude, QBC's combination of iteration and deflation always produces results scarcely worse than if the cubic's coefficients had each been perturbed by a few rounding errors at the start. In the worst case, when the three zeros of the cubic are all relatively nearly coincident, they may be correct to as few as a third of the figures carried; such a loss of accuracy also may afflict the closed form formula in that case. The phenomenon is illustrated by the following example:

Consider the cubic $x^3 - 3x^2 + 3x - (1-\varepsilon)$, where $1-\varepsilon$ is the number next less than 1 representable in the floating-point format used during computation. The zeros of this cubic are the three values of $1 - \varepsilon^{1/3}$. For instance, if 12 sig. dec. are carried during computation, $1-\varepsilon = 0.9999\ 9999\ 9999$ and the real zero $1 - \varepsilon^{1/3} = 0.9999$. Changing the coefficient $1-\varepsilon$ in its 12th sig. dec. to 1 changes all three zeros in the 4th to 1 .

In other examples, with two nearly coincident zeros relatively far from the third, about half the figures carried can be lost regardless of how the cubic is solved. But QBC never loses all the figures carried, as the closed-form formulas can. Examples to show what can happen will be presented later. Here is a summary of the conclusions that can be drawn from error analysis:

Each zero Z computed by QBC's combination of iteration and deflation is accurate almost to whichever is the largest of ...

- as many figures as were carried less the sum of the numbers of figures to which the other two zeros agree with Z , or
- half of the excess of the number of figures carried over the number of figures of agreement between Z , one of a pair of coincident or nearly coincident zeros, and a third zero relatively different from the pair, or ...
- a third of the figures carried, if all three zeros are coincident or nearly coincident with Z .

No way is known to calculate the zeros of a cubic more accurately than if its coefficients had first been perturbed by roundoff, unless part of the calculation is performed exactly –– with no roundoff at all. That exact calculation is part of a process called "Preconditioning", which will be described later in §7 .

6. Testing Considerations:

The obvious way to test QBC is to supply it with arguments for which accurate results have been calculated by some other method, and then compare. On reflection, this test procedure is not so obvious. What other method will give accurate results? Cubics can be constructed with small integer coefficients and at least one zero expressible as a ratio of small integers; but small integer input data might fail to stimulate typical rounding errors. And if results differ from what might ideally have been expected, how does one decide whether the differences are tolerable consequences of unavoidable rounding errors, or symptoms of a defect in the program that must be repaired?

A simple procedure that seems at first free from the dilemmas is to reconstruct the cubic from its computed zeros X, Y, Z by expanding A(x–X)(x–Y)(x–Z) in powers of x . If the cubic so reconstructed matches the given cubic well enough, the program that computed the zeros cannot be too wrong. But how well is "well enough" ? Presumably the reconstruction need match no more accurately than if X, Y and Z were correct zeros each rounded off to working precision (though actually they might be far less accurate than that); and the rounding errors that accrue during the reconstruction process have to be allowed for too. It's not so simple after all.

Program testing is fraught with anxiety unless one can estimate mathematically how big the errors should not be. Such an estimate of uncertainty can be very difficult; I would much rather have to write a program than have to analyze its errors or test it.

The program REVAL below computes a rigorous and fairly sharp bound Δ for the contribution of roundoff to the computed value $\mathcal{Q}$ of a cubic $Q(z) := Az^3 + Bz^2 + Cz + D$ at the same time as it computes $\mathcal{Q}$. REVAL requires knowledge about bounds for every rounding error committed by the computer in response to statements like " $s := x+y$; $d := x-y$; $p := x*y$; " in a program. These assignments store in the computer's memory values s, d and p slightly different from the ideal sum, difference and product desired. Almost every modern computer's arithmetic has its own characteristic tiny constants ε and δ that satisfy

$$|s-(x+y)| \le \delta|s| \ , \quad |d-(x-y)| \le \delta|d| \ , \quad |p-x*y| \le \varepsilon|x*y|$$

for all non-pathological values x and y representable in the computer (ignore ∞ and over/underflow for now) . Ideally

$$\delta = \varepsilon = (1.000...001 - 1.000...000)/2 \ ,$$

but some computer arithmetics are somewhat worse, and many suffer larger values of ε for complex multiplication than for real.

To apply the foregoing inequalities to the error analysis of any program that computes Q , first decompose the program into a sequence of simple assignments like

$$q_0 := A*z \ ; \quad q_1 := q_0 + B \ ; \quad ... \ ; \quad q_3 := q_2*z \ ; \quad Q := q_3 + D \ .$$

Then replace them by the inequalities they actually satisfy:

$$|q_0 - Az| \le \varepsilon|Az| \ ; \quad |q_1 - (q_0+B)| \le \delta|q_1| \ ; \quad ...$$
$$... \ ; \quad |q_3 - q_2z| \le \varepsilon|q_2z| \ ; \quad |Q - (q_3+D)| \le \delta|Q| \ .$$

These several inequalities boil down to one of the form

$$|Q - (Az^3 + Bz^2 + Cz + D)| \le \Delta$$

wherein Δ is expressed in terms of ε and δ and various computed values. Hence, Δ can be computed too thus:

```
Procedure REVAL( Z, A, B, C, D, $Q$, $\Delta$ ):
...  Given real coefficients A, B, C, D , this procedure yields
...  an approximation $Q$ to $\mathrm{Q}(Z) := AZ^3 + BZ^2 + CZ + D$ and a
...  bound $\Delta \ge |Q - \mathrm{Q}(Z)|$ , which would be zero if no roundoff
...  occurred. Instead, constants $\delta$ and $\varepsilon$ that reflect the
...  computer's roundoff must be put into the program. A bigger
...  $\varepsilon$ may be needed for complex arithmetic than for real.
        $e := |A|\varepsilon/(\varepsilon+\delta)$ ;
        $q_1 := AZ + B$ ;  $e := |Z|e + |q_1|$ ;
        $q_2 := q_1Z + C$ ;  $e := |Z|e + |q_2|$ ;
        $Q := q_2Z + D$ ;  $\Delta := (\varepsilon+\delta)|Z|e + |Q|\delta$ ;
        Return ;  End REVAL .
```

How might REVAL be tested? After proving that no computed value of Q can differ from an accurate evaluation of $\mathrm{Q}(Z)$ by more in magnitude than Δ , we have to show also that the error bound Δ is not so pessimistic as to be useless. Among large collections of trial data, Δ should sometimes barely exceed $|Q - \mathrm{Q}(Z)|$; the only way to verify this is to compute $\mathrm{Q}(Z)$ more accurately.

This procedure REVAL can serve to test the quality of Z as an approximate zero of the cubic; compute the quotient $|Q|/\Delta$. A quotient no bigger than 2 , say, indicates that no substantial improvement in the accuracy of Z is likely to be achieved unless arithmetic is carried out to higher precision. Of course, if you believe QBC works correctly you must believe that $|Q|/\Delta$ will be fairly small at every computed zero, in which case you'll not bother to compute that quotient. But REVAL has another use.

A bound upon the error in any approximate zero Z can be derived from REVAL's bound $\Delta \geq |\mathcal{Q} - Q(Z)|$, among other things, no matter what the provenance of Z. If Z is accurate enough, one step of Newton's iteration from Z to $Z - Q(Z)/Q'(Z)$ nearly doubles its number of correct digits, in which case $Q(Z)/Q'(Z)$ must approximate the error in Z fairly closely. That quotient is never much smaller than the error because, in general, $Q(z)$ must have a (possibly complex) zero z no farther from Z than $3|Q(Z)/Q'(Z)|$, according to a theorem of Laguerre. REVAL's $|\mathcal{Q}|+\Delta$ overestimates $|Q(Z)|$; and an *estimate* of $Q'(Z)$ comes either from $AZ^2 + q_1Z + q_2$, as in EVAL, or from $A(Z-X)(Z-Y)$ where X and Y approximate the other two zeros of the cubic. One way or another, $(|\mathcal{Q}|+\Delta)/|Q'(Z)|$ provides at least a *rough* bound for the error in Z.

A *rigorous* error bound derived from Laguerre's theorem requires a rigorous lower bound for $|Q'(Z)|$, which could be obtained from an augmented version of REVAL that accounted for roundoff's contribution to $Q'(Z)$ as well as to $Q(Z)$. Alternatively, if approximate zeros X, Y, Z are in hand, three calls to REVAL would help overestimate the right-hand sides of the inequalities

$$|x-X| \leq 3|Q(X)|/|A(X-Y)(X-Z)| ,$$
$$|y-Y| \leq 3|Q(Y)|/|A(Y-Z)(Y-X)| \quad \text{and}$$
$$|z-Z| \leq 3|Q(Z)|/|A(Z-X)(Z-Y)| ,$$

which rigorously bound the true zeros x, y, z of Q unless they are clustered so closely that these three estimates overlap. But rigorous bounds differ significantly from the previous paragraph's rough bounds only when zeros are clustered, and then time spent to get rigorous but probably dismal bounds might be better spent computing more accurate zeros with the aid of preconditioning.

7. Preconditioning:

Since error bounds are so often pessimistic, one might suspect that error analysts are pessimists too. Actually, error analysts are less interested in over-estimating error than in diminishing it. One way to diminish roundoff error is *preconditioning*, a process that transforms a problem hypersensitive to roundoff into a problem that is similar but far less sensitive.

The simplest illustration of the process concerns a quadratic equation in the form

$$ax^2 - 2bx + c = 0 ,$$

a form more convenient for our purpose than the usual form $Ax^2 + Bx + C = 0$ from which we get the desired form by setting $a := -2A$, $b := B$ and $c := -2C$. This equation is hypersensitive to rounding errors and also to any other perturbations of its coefficients just when its roots are relatively nearly coincident, in which case computed roots can be inaccurate in almost half the figures carried. For instance, when $a = 100002$, $b = 100001$ and $c = 100000$, the true roots $x = 1$ and $x = 0.9999800004...$ differ in their 5th digits from the double root $x = 0.9999900002$ computed on a 10-digit calculator using the familiar formula

$$x = (b \pm \sqrt{}(b^2-ac))/a ;$$

but the computed roots are just what would have been obtained in exact arithmetic had the coefficients b and c first been altered in digits beyond their 10th to $b = 100001.00000\ 00004$ and

c = 100000.0000*1 00005 99996 00008* . Such tiny perturbations are enough to cause relatively serious errors in $\sqrt{(b^2-ac)}$, errors avoidable only by carrying in worst cases twice as many sig. dec. in our computations and honoring twice as many sig. dec. in the coefficients as we wish to guarantee correct in computed roots.

When are the coefficients likely to be known so accurately? Most likely when they are known *exactly*, and then most likely when they are integers. Therefore, let us consider the case when a , b and c are all integers and, to simplify the exposition, let us assume that they are representable exactly in floating-point with a digit to spare. This means integers with no more than 9 digits on a 10-digit calculator, no more than 23 bits on a computer that performs binary floating-point with 24 sig. bits. If the coefficients were rather smaller than that, so small that the products b^2 and ac were both representable exactly, then the discriminant $q := b^2-ac$ would be fully accurate enough to produce entirely satisfactory results from a program like QDRTC above. That state of affairs is the goal of the preconditioning function DISC presented below. Without changing $q = b^2-ac$, it successively diminishes the integers a, b, c until either ac is negative or it differs enough from b^2 that DISC $:= b^2 - ac$ can be computed contaminated only relatively slightly by roundoff.

```
Real Function  DISC(a, b, c):
...  Given integers  a , b , c  all small enough to fit exactly
...  into floating-point with at least a digit to spare,  return
...  DISC = $b^2 - a c$  with roundoff confined to its last sig. dec.
     If  $a c > 0$  then
         { $a := |a|$ ;  $c := |c|$ ;
     loop:  if  $a < c$  then  swap(a, c) ; ...  now  $0 < c \le a$ .
            $n$ := integer nearest  $b/c$ ;  ...  $|n - b/c| < 1/2$ .
            if  $n \ne 0$  then  ...  ( else  $b^2 < c^2/4 \le ac/4$ )
                 { $\alpha := a - n b$ ;  ...  exact if  $\alpha \ge -a$
                   if  $\alpha \ge -a$  then  ...  ( else  $2b^2 > 3ac$ )
                       { $b := b - n c$ ;  ...  $|b| \le c/2$
                         $a := \alpha - n b$ ;
                         if  $a > 0$  then go to  loop } } } ;
     Return  DISC := $b^2 - a c$ ;  End DISC .
```

After substituting this preconditioning function DISC for the function DISC that accompanies the procedure QDRTC above, we can compute the desired roots $X_j+\imath Y_j$ of our quadratic to nearly full accuracy by calling QDRTC(a, $-2b$, c, $X_1+\imath Y_1$, $X_2+\imath Y_2$) .

When applied to our example above, DISC(100002, 100001, 100000) finds $n = 1$ and reduces a , b , c successively to $\alpha = 100002 - 100001 = 1$, $b = 100001 - 100000 = 1$, $a = 1 - 1 = 0$ and then returns DISC = 1 correctly having exploited massive cancellation without error. Here are some more examples:

a	b	c	crude DISC	refined DISC	true b^2-ac
3234424085	1160927837	416690270	*398000000000*	397448345600	397448345619
3234413351	1160928203	416690636	*-89000000000*	-89060331630	-89060331627
8952751441	1557625	271	*0*	114	114
8952751442	1557625	271	*0*	-157	-157
5309162499	2301700899	997864924	*-6000000000*	-5110876875	-5110876875
5309162499	2301700899	997864923	*0*	198285624	198285624
5309162499	2301700899	997864922	*5000000000*	5507448123	5507448123

All columns but the last were obtained from versions of DISC programmed into the HP-15C, a ten-figure calculator. The last column comes from the HP-71B, a twelve-figure machine, using a faster version of DISC that exploits the INEXACT flag provided by IEEE standard p854, to which the HP-71B conforms:

```
    DEF FNq(a,b,c)  ! ... q  := b^2 - a*c  more accurately.  (in BASIC)
    i0 = FLAG(INX,0)  ! ... saves and resets INEXACT flag.
    'loop':  b0 = b*b @  a0 = a*c  ! ...  Are they exact?
               IF  FLAG(INX,i0)=0 OR a0<=0  THEN  GOTO 'fin'
                 IF  ABS(c)>ABS(a)  THEN  a0=a @  a=c @  c=a0  ! ...  swap(a,c)
                 b0 = RED(b,c) @  n = IROUND((b-b0)/c)  ! ...  RED is IEEE rem
                 i1 = FLAG(INX,0)  ! ...  resets INEXACT flag.
                 a0 = (a - n*b) - n*b0
                 IF  FLAG(INX)=0  THEN  a = a0 @  b = b0 @  GOTO 'loop'
    'fin': FNq = b*b - a*c @  END DEF
```

An idea similar to that in DISC , but applied very differently, serves to precondition the cubic equation

$$q(x) \;:=\; ax^3 - 3bx^2 + 3cx - d \;=\; 0$$

when all its coefficients except perhaps d are integers representable exactly in floating-point with at least a digit or two to spare. QBC will calculate the equation's roots but, in the light of error analyses mentioned above, we must expect the calculated roots to suffer badly from roundoff whenever they are clustered. Fortunately that possibility, clustered roots, can be recognized easily without any call upon QBC ; if all three roots are nearly coincident then all three quotients b/a, c/b and d/c must be nearly coincident too. In fact, a little algebraic manipulation suffices to prove that the quotients match to beyond twice as many sig. digits as are common to the roots. To exploit this phenomenon, choose λ to approximate all three quotients rounded to no more sig. digits than are left unoccupied by the first three coefficients; this means that all three products λa, λb and λc will be computed exactly in floating-point arithmetic. Next replace x by $\lambda+y$ in the given equation to get a new cubic

$$q(\lambda+y) \;=\; ay^3 - 3b'y^2 + 3c''y - d^* \;=\; 0$$

which QBC can solve for roots y , whence $x = \lambda+y$, much more accurately than before. New coefficients must be calculated thus:

$$d' := d - \lambda c \;;\quad c' := c - \lambda b \;;\quad b' := b - \lambda a \;;$$
$$d'' := d' - \lambda c' \;;\quad c'' := c' - \lambda b' \;;$$
$$d^* := d'' - \lambda c'' \;.$$

Cancellation will occur in the first row without error; and if rounding errors do occur later they will be far tinier than what QBC would likely inflict upon the original coefficients. When all three roots x are extremely close, so close that all three roots y must be relatively nearly coincident too, no rounding errors will occur during the calculation of the new coefficients b', c'' and d^*, and then the foregoing transformation may be repeated advantageously with a new tinier λ .

When two roots are nearly coincident but relatively far from the third, the three quotients above must be replaced by two values

$$(1/2)(bc - ad)/(b^2 - ac) \quad \text{and} \quad \pm\sqrt{(c^2 - bd)/(b^2 - ac)} \;.$$

They can be shown to match to about twice as many sig. digits as are in agreement between the two nearly coincident roots; and λ must approximate those two values rounded to at most half as many digits as are left unoccupied by the first three coefficients, so that all three products λ^2a, λ^2b and λc will be computed exactly in floating-point arithmetic. Then the new coefficients and the roots $x = \lambda+y$ may be calculated as above except when d turns out to be small compared with $a\lambda^3$. In that special case, the third root will be rather smaller than the two that are nearly coincident, so it may well be computed more accurately from the original coefficients than from the new ones. Moreover, in case d is small and not an integer, the formulas for d', d'' and d^* should be changed as follows for better accuracy in the nearly coincident roots $\lambda+y$:

$$D := \text{integer nearest } d \;; \qquad \delta := d - D \;;$$
$$d' := D - \lambda c \;; \quad d'' := d' - \lambda c' \;; \quad d^* := (d'' - \lambda c'') + \delta \;.$$

A detailed explanation to justify the foregoing procedures is too complicated to include in these notes. Instead, a few examples will illustrate the schemes' efficacy.

These examples were all worked out on an HP-15C calculator, which carries 10 sig. dec. First the zeros x of each given cubic $q(x)$ were obtained from a program like QBC , listed at the end of these notes, to see how inaccurately it computes clustered zeros. Then quotients of coefficients were examined to determine a choice of λ from which new coefficients of $q(\lambda+y)$ were derived. The intermediate results of this computation are displayed below with strings of leading "o's" to denote digits that cancelled off. Then QBC was rerun to compute the zeros y of $q(\lambda+y)$, from which were obtained improved zeros $x = \lambda+y$ whose correctness was verified on an HP-71B carrying 12 sig.dec.

```
Q(x)   =  658x³ - 190125x² + 18311811x - 587898164
QBC:  x = 96.297 ,  96.341 ,  96.305
b/a = 96.31458967    c/b = 96.31458777    d/c = 96.31458582    λ := 96.3
a = 658     b  = 63375        c  = 6103937       d  = 587898164
            b' = oooo9.6      c' = oooo924.5     d' = oooo89030.9
                              c" =      ooo.o2   d" =     ooool.55
                                                 d* =        -o.376
Q(λ+y)  =  658y³ - 28.8y² + 0.06y + 0.376
QBC:  λ+y = 96.22963935,  96.35706483 ± 0.06974975204 z
```

$Q(x) = 2212111x^3 - 73449x^2 + 813x - 3$

QBC: $x = 0.0110969266\mathit{5}$, $0.0110530996\mathit{1} \pm 0.00020090294\mathit{81}\,\imath$

```
b/a = 0.0110677     c/b = 0.0110689     d/c = 0.0110701     λ := 0.0111
a = 2212111     b  =  24483         c  =  271               d  =  3
                b' = -ooo71.4321   c' = -ooo.7613           d' = -o.oo81
                                   c" =     0.o3159631      d" =  0.00o35043
                                                            d* = -0.000ooo289041
```

$Q(\lambda+y) = 2212111y^3 + 214.2963y^2 + 0.09478893y + 0.000000289041$

QBC: $\lambda+y = 0.01109693006$, $0.01105309791 \pm 0.0002009034814\,\imath$

λ is not critical, nor is a small rounding error in d* . Here is the previous example repeated with a different λ := 0.01107 :

```
a = 2212111  b  =  24483          c  =  271                d  = 3
             b' = -oooo5.06877   c' = -ooo.o2681           d' = o.oooo3
                                 c" =    0.0293012839      d" = 0.0003267867
                                                           d* = 0.000oo24214872..
```

Yet QBC delivers practically the same final results λ+y as before.

$Q(x) = 6111x^3 - 51792x^2 + 109737x + 0.00623$

QBC: $x = -5.677209907_{10}{-8}$, $4.23747\mathit{7594}$, $4.23773\mathit{1105}$

$(bc-ad)/(2(b^2-ac)) = \surd(\ (c^2-bd)/(b^2-ac)\ ) = 4.237604349 \qquad \lambda := 4.24$

```
a = 6111   b  =  17264      c  =  36579        D  =  0        δ = d = -0.00623
           b' = -o8646.64   c' = -36620.36     d' = -155094.96
                            c" =  ooo41.3936   d" =  ooo175.3664
                                               d* =     -ooo.148694
```

$Q(\lambda+y) = 6111y^3 + 25939.92y^2 + 124.1808y + 0.148694$

QBC: λ+y = -o.oooooooo56 , 4.237583786 , 4.237624911

d is so tiny that the isolated root is best calculated directly from Q(x).

The foregoing discussion may promote a misleading impression that preconditioning is worth while only if the data (coefficients) are given exactly. Other circumstances do exist when preconditioning helps, however. For example, the errors in the data could be correlated in a way that is known to mostly cancel in the results. Or the coefficients, though uncorrelatedly erroneous, may figure subsequently in several related contexts among which consistency of some kind is essential even though ultimate accuracy is not. For instance, suppose a program uses the zeros of the cubic and also of its derivative; Rolle's theorem implies that the latter zeros should lie between the former when they are all real, and a theorem due to Gauss places the latter inside the convex hull of the former when they are complex. If those relationships are violated by clustered approximate zeros computed too inaccurately, the subsequent logic of the program could malfunction. Adapting that logic to disordered zeros can be far more complicated than preconditioning in a way that protects their order from roundoff. However, preconditioning procedures appropriate for noninteger data go far beyond the scope of these notes.

8. Scaling Invariance vs. Over/Underflow:

The factored form of the cubic

$$A x^3 + B x^2 + C x + D \;=\; A (x - X) (x - Y) (x - Z)$$

provides a factorization for the scaled cubic

$$(\sigma A) x^3 + (\sigma B \varrho) x^2 + (\sigma C \varrho^2) x + (\sigma D \varrho^3) \;=\; \sigma A (x - \varrho X) (x - \varrho Y) (x - \varrho Z) \;.$$

If the scale factors σ and ϱ are powers of the radix (10 for a decimal calculator, 2 for a binary computer), then the scaled coefficients σA, $\sigma B \varrho$, $\sigma C \varrho^2$, $\sigma D \varrho^3$ will have the same significant digits as the original coefficients A, B, C, D ; only the decimal or binary points will have shifted. Therefore the same should be true of the scaled zeros ϱX, ϱY, ϱZ , even in the face of roundoff. Of course, the relationship between the scaled zeros and the original zeros X, Y, Z must break down when the scale factors are so big or so tiny that the scaled coefficients or zeros over/underflow; ideally the relationship should not break down for any other reason. In practice, most algorithms are vulnerable to spurious over/underflow. For instance, the discriminant q in QDRTC and the quotients r and t in QBC can easily over/underflow even though the coefficients and zeros lie well within range. Conscientious programmers introduce scale factors into their programs either to forestall undeserved over/underflows or to recover from them. The task is not eased by the absence from most programming languages of any reference to over/underflow other than an implication that the crime will be punished by termination of the program's execution.

Here is how a scale factor σ can be chosen to prevent spurious over/underflow during the solution of a quadratic equation $Ax^2 + Bx + C = 0$. If $A = 0$ or $C = 0$ the solution is obvious. Otherwise choose σ to be a power of the radix near $\sqrt{|A|}\sqrt{|C|}$, and so chosen that neither A/σ nor C/σ can over/underflow. Then $|(A/\sigma)(C/\sigma)|$ cannot be orders of magnitude larger or smaller than 1 . Next compare $|B|$ with σ ; if $|B|$ is so much bigger than σ that $|B| + \sigma$ rounds to $|B|$, then the quadratic's roots are approximated accurately enough by $-C/B$ and $-B/A$. Otherwise call QDRTC$(A/\sigma,\ B/\sigma,\ C/\sigma,\ X_1 + \iota Y_1,\ X_2 + \iota Y_2)$, allowing underflows to flush to 0 if nothing better is available. No undeserved overflow will occur.

Similar ideas can help suppress spurious over/underflows when solving the cubic. Roughly speaking, when A/B is very tiny, much tinier than roundoff in numbers near 1 , but B/C is not tiny at all, then the cubic's biggest zero must be very nearly $-B/A$, and the other zeros can be found by setting $A := 0$ and solving the resulting quadratic equation. And when D/C is very tiny but C/B is not, the tiniest zero is very nearly $-D/C$, and so on. When neither A/B nor D/C is very tiny, the cubic and its zeros can be scaled and computed in the ordinary way.

9. Some Trial Data for Cubic Equation Solvers:

Notation:
Coefficients A, B, C, D of cubic $Ax^3 + Bx^2 + Cx + D$ are input.
Output are real zeros X, Z_1, Z_2 or complex zeros Z .
Parameters: M is a small integer; N is a big integer; usually
$|N|$ is almost as big as possible without roundoff.
$u := M/N$; $v := 1/(2N)$.
t is a tiny number; $1000 \pm t$ rounds to 1000 .
h is a huge number; $h \pm 1$ rounds to h .

Follow the formulas for coefficients EXACTLY; rounding them could change zeros drastically.

Cubics with small integer coefficients:
$A = 1$, $B = D = -6$, $C = 11$. $X = 3$, $Z_1 = 1$, $Z_2 = 2$.
$A = D = 1$, $B = C = 0$. $X = -1$, $Z = 0.5 \pm \imath\sqrt{0.75}$.
$A = -D = 1$, $B = C = 0$. $X = 1$, $Z = -0.5 \pm \imath\sqrt{0.75}$.
$A = 0$, $B = 1$, $C = 3$, $D = 2$. $X = \infty$, $Z_1 = -1$, $Z_2 = -2$.
$A = 1$, $B = -3$, $C = 2$, $D = 0$. $X = 0$, $Z_1 = 1$, $Z_2 = 2$.
$A = D = 1$, $B = C = 3$. $X = Z_1 = Z_2 = -1$.
$A = -B = -C = D = 1$. $X = -1$, $Z_1 = Z_2 = 1$.
$A=1$, $B=-30$, $C=299$, $D=-1980$. $X = 20$, $Z = 5 \pm \imath\sqrt{74}$.

Cubics with zeros of very different magnitudes:
$A=1$, $B=-30$, $C=299$, $D=-t$. $X \doteq t/299$, $Z \doteq 15 \pm \imath\sqrt{74}$.
$A = -D = t$, $-B = C = h$. $X = 1$, $Z_1 \doteq t/h$, $Z_2 \doteq h/t$.
$A = 1$, $B = -h$, $C = -t$, $D = ht$. $X = h$, $Z = \pm\sqrt{t}$.
$A = D = 1$, $B = C = 1 - N - 1/N$. $X = 1/N$, $Z_1 = -1$, $Z_2 = N$.

Cubics with ill-conditioned zeros:
$A = -C = N+1$, $D = -B = N-1$. $-X = Z_1 = 1$, $Z_2 = 1 - 2/(N+1)$.
$A = -D = N$, $C = -B = 3N+2M$. $X = 1$, $Z = 1+u \pm \sqrt{(2u+u^2)}$.
$A=B=3C$, $C = 9N^3$, $D = 1-N^3$. $X = (1-2v)/3$, $Z = 1+v \pm \imath v\sqrt{3}$.
$A=D= N^2+M^2$, $B=C= 3M^2-N^2$. $X=-1$, $Z = 1-2u^2/(1+u^2) \pm 2\imath u/(1+u^2)$.
$A=-D=N^2+M^2$, $-B=C=3N^2-M^2$. $X=1$, $Z = 1-2u^2/(1+u^2) \pm 2\imath u/(1+u^2)$.

10. Selected Results from the HP-15C :

Both algorithms above, one using the Formula with complex arcsin, one like QBC that iterates to solve a cubic, have been programmed into the HP-15C calculator along with a program like REVAL to compute $Q(x)$ and Δ . The results tabulated below show the coefficients A, B, C, D of the cubic $Q(x)$ and the zeros X, Y, Z obtained first from the programmed Formula, second from exact calculation on another machine, third from the iterative method QBC. Below QBC's results are shown corresponding quotients $|Q(X)|/\Delta(X)$, $|Q(Y)|/\Delta(Y)$, $|Q(Z)|/\Delta(Z)$ as computed by a program like REVAL .
The HP-15C rounds arithmetic to 10 sig. dec., corresponding to $\delta = \varepsilon = 5e-10$.

$A = 1$	Formula	$X = 1$	$Y = 2$	$Z = 3$
$B = -6$	Correct	$X = 1$	$Y = 2$	$Z = 3$
$C = 11$	Iter've	$X = 1$	$Y = 2$	$Z = 3$
$D = -6$	$\|Q\|/\Delta$	0	0	0

$A = -1$	Formula	$X = -1$	$Y = 0.4999999999 \pm 0.8660254037\,\imath$
$B = 0$	Correct	$X = -1$	$Y = 0.5 \pm 0.866025403784\,\imath$
$C = 0$	Iter've	$X = -1$	$Y = 0.5 \pm 0.8660254038\,\imath$
$D = -1$	$\|Q\|/\Delta$	0	0

```
A = 1             Formula  X = -1  =  Y  =  Z
B = 3             Correct  X = -1  =  Y  =  Z
C = 3             Iter've  X = -1  =  Y  =  Z
D = 1             |Q|/Δ        0           ... ,  so the programs work at least sometimes.

A = 1             Formula  X = 2.094551481    Y = -1.047275741  ±  1.135939889  ι  ...  Newton's
B = 0             Correct  X = 2.0945514815   Y = -1.0472757408 ±  1.1359398891 ι        own
C = -2            Iter've  X = 2.094551481    Y = -1.047275741  ±  1.135939889  ι       example.
D = -5            |Q|/Δ        0.27                 0.15

A = 1             Formula  X = 4e-10       Y = 1                          Z = 2  ...  X  is ominous.
B = -3            Correct  X = 0           Y = 1                          Z = 2
C = 2             Iter've  X = 0           Y = 1                          Z = 2
D = 0             |Q|/Δ        0              0                              0

A = 1             Formula  X = 4e-10        Y = 1                          Z = 2  ...    X  is wrong.
B = -3            Correct  X = -1.17e-89    Y = 1                          Z = 2         This is why  QBC  has
C = 2             Iter've  X = -1.17e-89    Y = 1                          Z = 2  :::     1.000..001  in it.
D = 2.34e-89      |Q|/Δ        0                  9.4e-81                     2.9e-81

A = 1             Formula  X = 7999999998      Y = 43545       Z = -43545   ...  Y and Z  are very wrong.
B = -7999999999   Correct  X = 8000000000      Y =  1          Z =  -2
C = -8000000002   Iter've  X = 8000000000      Y =  1          Z =  -2
D = 16000000000   |Q|/Δ        0                  0                0

A = 16000000000   Formula  X = 1.0e-10     Y = 0.9999999999  Z = -0.4999999999  ...  Y and Z  are  O. K.
B = -8000000002   Correct  X = 1.25e-10    Y =  1            Z = -0.5                now,  but  X  isn't.
C = -7999999999   Iter've  X = 1.25e-10    Y =  1            Z = -0.5              This cubic is the
D = 1             |Q|/Δ        0              0                 0.1                previous one reversed.

A = 1             Formula  X =  99999.99997   Y = -1.0443      Z = 0.04433  ... Y and Z  are wrong again,
B = -99999.00001  Correct  X = 100000         Y = -1           Z = 0.00001      and reversing the cubic
C = -99999.00001  Iter've  X = 100000         Y = -1           Z = 0.00001      won't improve  Y .
D = 1             |Q|/Δ        0                 0.1              0

A = 0.01          Formula  X = 1.01e-4 - 2e-6 ι      Y = 14999.99995 ± 8602.325194 ι
B = -300          Correct  X = 1.00000001003e-4      Y = 14999.99995 ± 8602.32517986 ι
C = 2990000       Iter've  X = 1.000000010e-4        Y = 14999.99995 ± 8602.32518 ι
D = -299          |Q|/Δ        0                               0.00015
```

The Formula's value X is wrong in the worst way: wrong enough to matter, but not obviously wrong.

```
A = -3            Formula  X = 0.3333333333  =  Y  =  Z
B = 3             Correct  X = 0.333178613706  Y = 0.333410693147 ± 0.000133991128129 ι
C = -1            Iter've  X = 0.3333333333    Y = 0.3333366667   Z = 0.33333
D = 0.1111111111  |Q|/Δ        0                   0                 0
```

Better results cannot be expected from calculations carried out to 10 sig. dec., since as many as two thirds of the figures carried can be lost if all three zeros are nearly coincident.

```
A = 10000000000   Formula  X = -0.9999999997   Y = 0.9999999998 ± 0.00001721325932 ι
B = -9999999998   Correct  X = -1              Y = 1                Z = 0.9999999998
C = -A            Iter've  X = -1              Y = 1.000014142      Z = 0.999985858
D = -B            |Q|/Δ        0.057              0.27                 0
```

Better results cannot be expected from calculations carried out to 10 sig. dec., since as many as half the figures carried can be lost if two zeros are nearly coincident but far from the third. Note that any program that computes X, Y, Z as well as can be expected should produce values for |Q|/Δ smaller than 1 or 2 , but the smallness of that quotient does not by itself tell how accurate a computed zero may be.

11. A Program for the HP-15C:

This program deals with cubics $Q(x) = ax^3 + bx^2 + cx + d$ and quadratics $rx^2 + sx + t$. Function keys [A], [B], [C], [D], [E] and [1] are invoked via [GSB] [A] etc. Stack register *X* is normally displayed; to see the other registers *Y*, *Z* and *T* , use the [R↓], [R↑] or [X≷Y] keys. Here is what the program does:

[A]: a [ENTER] b [ENTER] c [ENTER] d [A] stores a, b, c, d in cells #3, #4, #5, #6 resp. for ...

[B]: Using coefficients a, b, c, d stored by [A] , solves Q(x) = 0 for roots *X*, *Y*, *Z* by means of a formula involving complex arcsin. Scratches cells #7, #8, #9 .

[E]: Using coefficients a, b, c, d stored by [A] , solves Q(x) = 0 for roots *X*, *Y*, *Z* as QBC does by iteration and deflation. *X* is real and also in cell #9 ; *Y* and *Z* are complex-conjugates. Scratches cells #7 and #8 .

[C]: Using coefficients a, b, c, d stored by [A] , copies *X* into *Z* and *T* , writes |*X*| into cell #7, writes Q(*X*) over *X* and an error bound for Q(*X*) onto *Y* . *X* may be complex. Cf. REVAL.

[1]: Using coefficients a, b, c, d stored by [A] , writes *X* into *Z* and *T* , Q'(*X*) into *Y* and Q(*X*) over *X* ; and if *X* was real, then leaves a*X*+b in cell #7 , (a*X*+b)*X*+c in #8 .

[D]: r [ENTER] s [ENTER] t [D] solves a quadratic equation $rx^2 + sx + t = 0$ for its roots *X* and *Y* , which may be complex. $r \neq 0$. Cf. QDRTC.

Program Text:

```
LBL[A] STO6 R↓ STO5 R↓ STO4 R↓ STO3 RTN LBL[B] CF8 RCL4 RCL3 X=0? GTO9 ÷ 3 CHS ÷ STO7 X² STO8 3 x RCL5
RCL÷3 - STO9 RCL-8 RCLx7 RCL6 RCL÷3 - RCL9 X=0? GTO0 ÷ 3 x RCL9 .75 ÷ SF8 √X ÷ LSTX X≷Y SIN⁻¹ 3 ÷ CF0 X<0? SF0
π 3 ÷ F0? CHS X≷Y - LSTX SIN R↑ x X≷Y SIN R↑ x GTO2 LBL0 X≷Y CHS 3 1/X SF8 Yˣ ENTER ENTER 1 CHS LSTX Yˣ CHS
x LBL2 RCL7 X≷Y - RCL7 LSTX R↑ + RCL+7 X≷Y LSTX - RTN LBL[C] ABS STO7 4.006 STOI LSTX RCL3 ENTER ABS 2
÷ LBL3 RCLx7 R↓ x RCL+(i) ENTER ABS R↑ + ISGI GTO3 LSTX + 2 EEX 9 ÷ X≷Y RTN LBL[D] CF8 X≷Y 2 CHS ÷ CF0
X<0? SF0 STOI X² R↓ x LSTX X≷Y R↑ - CHS X≥0? GTO4 CHS √X R↑ ÷ ENTER CHS RCLI LSTX ÷ X≷Y fI ENTER R↑ fI RTN
LBL4 √X F0? CHS RCL+I X=0? RTN ÷ LSTX R↑ ÷ RTN LBL[E] CF8 EEX CHS 9 eˣ STO9 RCL6 X=0? GTO6 RCL4 RCL3
X=0? GTO9 ÷ 3 CHS ÷ GSB1 RCL÷3 CF0 X<0? SF0 ABS 3 1/X Yˣ X≷Y RCL÷3 CHS X≤0? GTO5 √X X>Y? X≷Y CLX 1.325 x ENTER
LBL5 CLX + F0? CHS - X=Y? GTO7 LBL6 GSB1 X≷Y X=0? GTO7 ÷ RCL÷9 - X=Y? GTO7 LSTX F0? CHS X>0? GTO7 R↓ GTO6
LBL[1] ENTER ENTER RCLx3 ENTER RCL+4 STO7 + x X≷Y RCLx7 RCL+5 STO8 + X≷Y LSTX x RCL+6 RTN LBL7 R↓ R↓
STO9 X=0? GTO8 X² RCLx3 ABS RCL6 RCL÷9 ABS X>Y? GTO8 LSTX CHS STO8 RCL-5 RCL÷9 STO7 LBL8 RCL3 RCL7 RCL8 GTO5
LBL9 1 TANH⁻¹ STO9 RCL4 RCL5 RCL6 LBL5 GSBD RCL9 RTN      ( 303 steps )
```

12. Program Timings:

For the selected results from the HP-15C exhibited above, the closed-form formula program [B] took about 14 sec. on average; the iterative QBC program [E] averaged roughly 27 sec. But program [B] was inaccurate at times; to get results as reliable as [E]'s , program [B] would have to be run twice, the second time with coefficients reversed, and then the two sets of results would have to be combined with some additional arithmetic. Thus, the iterative program runs faster on the HP-15C than would a *reliable* program based upon closed-form formulas despite that the complex inverse trigonometric functions available on that machine, but on few others, promote the implementation of the formulas.

13. Annotated Bibliography:

An old encyclopaedia like the *Britannica* is as good a place as any to look up the Italians Scipione Ferro, Tartaglia (Niccolo Fontana) and Hieronimo Cardano, and the Frenchman Franciscus Vieta, who first produced closed-form solutions for the cubic equation. Their formulas can be found there too under the heading "Equations, Theory of"; or in handbooks like the *Handbook of Chemistry and Physics*, the Chemical Rubber Publishing Co., Cleveland; or the *Handbook of Mathematical Functions* edited by M. Abramowitz and Irene Stegun, #55 in the Applied Mathematics Series published in 1964 by the U. S. National Bureau of Standards but obtainable now reprinted by Dover, N. Y. The algorithm QBC presented in §3 and §4 has not been published before.

The genesis of rounding errors on older electronic computers is described well by Patrick H. Sterbenz in his book *Floating-Point Computation*, published in 1974 by Prentice-Hall, N. J. A better arithmetic design is specified by the IEEE standards 754-1985 and p854, to which many of the newest computers conform; these standards have been described by W. J. Cody *et al.* in "A Proposed Radix- and Word-length-independent Standard for Floating-Point Arithmetic" in IEEE *MICRO*, August 1984, pp. 86 - 100.

An elementary overview of error analysis is provided in parts of the *HP-15C Advanced Functions Handbook*, Hewlett-Packard part no. 00015-9011, 1982. Backward error analysis in particular is the subject of *Rounding Errors in Algebraic Processes* by James H. Wilkinson, Prentice-Hall, 1963. The error analysis summarized in §5 has not been published yet; its approach is similar to that in Brian T. Smith's "Error Bounds for Zeros of a Polynomial Based Upon Gerschgorin's Theorem" in the *Journal of the ACM* vol. 17 (1970), pp. 661-674, wherein may be found also the proof of the claims for the three inequalities near the end of §6 .

§6's procedure REVAL is similar to one presented and explained in "A stopping criterion for polynomial root-finding" by Duane A. Adams, *Communications of the ACM* vol. 10 (1967), pp. 655-658. The preconditioning techniques in §7 and the scaling techniques in §8 are new although similar in spirit to techniques described in the author's lecture notes since 1963. The theorem by Gauss that relates the zeros of a polynomial and of its derivative, and Laguerre's theorem mentioned in §6 , can both be found in *Geometry of Zeros* by M. Marden (1966), American Mathematics Society, Providence, R. I.

14. Acknowledgements:

Though first assembled for use in my Numerical Analysis classes in 1985 and 1986, these notes summarize work performed in a desultory way over two previous decades, partially supported at times by a grant from the U. S. Office of Naval Research under contract number N00014-76-C-0013 , and more recently by grants from the Air Force Office of Scientific Research under contract number AFOSR-84-0158, and from the Army Research Office under contract number DAA629-85-K-0070. I am indebted also to Prof. Beresford N. Parlett for patient encouragement and advice while these notes were being compiled into their present form. Finally, I am grateful for the sympathetic hearing granted by Prof. George E. Forsythe at a time, twenty years ago, when many more people than nowadays thought the subject matter of these notes a subject fit only for nit-pickers.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
University of California, Berkeley

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
To Solve a Real Cubic Equation

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
CPAM report, November 1986

5. AUTHOR(S) (First name, middle initial, last name)
W. Kahan

6. REPORT DATE
November 1986

7a. TOTAL NO. OF PAGES
20

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.
N00014-85-K-0180

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

11. SUPPLEMENTARY NOTES
Also supported by
AFOSR-84-0158 (Air Force)
DAA629-85-K-0070 (U. S. Army)

12. SPONSORING MILITARY ACTIVITY
Mathematics Branch
Office of Naval Research
Washington, DC 20360

13. ABSTRACT

A program to solve a real cubic equation efficiently and as accurately as the data deserve is not yet an entirely cut-and-dried affair. An iterative method is the best found so far. This method plus some other issues, like accuracy, scaling, preconditioning and testing, are discussed in these notes in enough detail to convey an impression of what Numerical Analysis is about.