see http://www.cs.berkeley.edu/~wkahan/Math128/SOLVEkey.pdf

Personal Calculator Has Key to Solve Any Equation f(x) = 0

The HP-34C is the first handheld calculator to have a built-in numerical equation solver. That's why one of its keys is labeled SOLVE.

by William M. Kahan

BUILT INTO HEWLETT-PACKARD'S new handheld calculator, the HP-34C, is an automatic numerical equation solver. It is invoked by pressing the **SOLVE** key (see Fig. 1). For an illustration of how it finds a root x of an equation f(x) = 0 take the function

$$f(x) \equiv e^x - C_1x - C_2$$

with constants C_1 and C_2. Equations f(x) = 0 involving functions like this one have to be solved in connection with certain transistor circuits, black-body radiation, and stability margins of delay-differential equations. If the equation f(x) = 0 has a real root x three steps will find it:

Step 1. Program f(x) into the calculator under, say, label **A** (see Fig. 2).

Step 2. Enter one or two guesses at the desired root:
(first guess) **ENTER** (second guess if any)
Any x will do as a guess provided f(x) is defined at that value of x, but the closer a guess falls to a desired root the sooner that root will be found.

Step 3. Press **SOLVE A** and wait a little while to see what turns up.

Figs. 3a-3d show what turns up for a typical assortment of constants C_1 and C_2 and first guesses.

When a root is found it is displayed. But is it correct? When no root exists, or when **SOLVE** can't find one, ERROR 6 is displayed. But how does the calculator know when to abandon its search? Why does it not search forever? And if it fails to find a root, what should be done next? These questions and some others are addressed in the sections that follow.

What does SOLVE Do, and When Does It Work?

Neither **SOLVE** nor any other numerical equation solver can understand the program that defines f(x). Instead, equation solvers blindly execute that program repeatedly. Successive arguments x supplied to the f(x) program by **SOLVE** are successive guesses at the desired root, starting with the user's guess(es). If all goes well, successive guesses will get closer to the desired root until, ideally, f(x)=0 at the last guess x, which must then be the root. **SOLVE** is distinguished from other equation solvers by its guessing strategy, a relatively simple procedure that will surely find a root, provided one exists, in an astonishingly wide range of circumstances. The three simplest circumstances are the ones that predominate in practice:

1. f(x) is strictly monotonic, regardless of initial guesses, or
2. ±f(x) is strictly convex, regardless of initial guesses, or
3. Initial guesses x and y straddle an odd number of roots, i.e., f(x) and f(y) have opposite signs, regardless of the shape of the graph of f.

In these cases **SOLVE** always finds a root of f(x)=0 if a root exists.

About as often as not, **SOLVE** must be declared to have found a root even though f(x) never vanishes. For example, take the function:

$$g(x) \equiv x+2\cdot(x-5)$$

Fig. 1. *The HP-34C, a new handheld programmable calculator, has two keys that are new to handheld calculators– ∫(integrate) and* **SOLVE** *SOLVE, a numerical equation solver, is described in this article.*

PRGM RUN	Switch to Program Mode
CL PRGM	Clear Program Memory
LBL A	x Is in the X Register
e^x	e^x
LST x	Get x Back
RCL 1	C_1
×	C_1x
−	$e^x - C_1x$
RCL 2	C_2
−	$f(x) = e^x - C_1x - C_2$
RTN	Return f(x) in the X register
PRGM RUN	Switch to Run Mode
... C_1 ...	
STO 1	Store C_1 in Register 1
... C_2 ...	
STO 2	Store C_2 in Register 2

Fig. 2. *This is an HP-34C program for the function f(x) = $e^x - C_1x - C_2$. It replaces x by f(x) in the HP-34C's X register (display). It is labeled A, but labels B, 0, 1, 2, or 3 would serve as well.*

Of course g(x) = 3x−10, and when calculated as prescribed above (don't omit the parentheses!) it is calculated exactly (without roundoff) throughout $1 \leq x \leq 6.666666666$. Consequently, the calculated value of g(x) cannot vanish because the obvious candidate x=10/3=3.333... cannot be supplied as an argument on an ordinary calculator. **SOLVE** does the sensible thing when asked to solve g(x)=0; it delivers final guesses 3.333333333 and 3.333333334 in the X and Y registers in a few seconds. In general, when **SOLVE** finds a root of f(x)=0 it returns two final guesses x and y in the X and Y registers respectively; either x=y and f(x)=0, or else x and y differ in their last (10th) significant decimal digit and f(x) and f(y) have opposite signs. In both cases the Z register will contain f(x).

On the other hand, **SOLVE** may fail to find a place where f(x) vanishes or changes sign, possibly because no such place exists. Rather than search forever, the calculator will stop where |f(x)| appears to be stationary, near either a local positive minimum of |f(x)| as illustrated in Fig. 3d or where f(x) appears to be constant. Then the calculator displays ERROR 6 while holding a value x in the X register and f(x) in the Z register for which $f(y)/f(x) \geq 1$ at every other guess y that was tried, usually at least four guesses on each side of x. (One of those guesses is in the Y register.) When this happens the calculator user can explore the behavior of f(x) in the neighborhood of x, possibly by pressing **SOLVE** again, to see whether |f| really is minimal near x, as it is in Fig. 3d, or whether the calculator has been misled by unlucky guesses. More about this later.

So **SOLVE** is not foolproof. Neither is any other equation solver, as explained on page 23.

How Does SOLVE Compare with Other Root-Finders?

Program libraries for large and small computers and calculators usually contain root-finding programs, but none of them works over so wide a range of problems or so conveniently as does the HP-34C's **SOLVE** key. Other root-finders are hampered by at least some of the following limitations:

1. They insist upon *two* initial guesses that straddle an odd number of roots. **SOLVE** accepts any guess or two and does what it can to find a root nearby, if possible, or else farther away.
2. They may have to be told in advance how long they are permitted to search lest they search forever. Consequently their search permit may expire after a long search, but just moments before they would have found a root. **SOLVE** knows when to quit; it can't go on forever, but it can go on for a long time (e.g., when f(x)=1/x).
3. They may require that you prescribe a tolerance and then oblige you to accept as a root any estimate closer than that tolerance to some previous estimate, even if both estimates are silly. **SOLVE** will claim to have found a root x only when either f(x)=0 or $f(x)\cdot f(y)<0$ for some y differing from x only in their last (10th) significant decimal digit.
4. They may claim that no root exists when they should admit that no root was found. **SOLVE** will not abandon its search unless it stumbles into a local minimum of |f|, namely an argument x for which $f(y)/f(x) \geq 1$ at all other (usually at least nine) sampled arguments y on both sides of x.
5. They may deny to the program that calculates f(x) certain of the calculator's resources; for instance
   "begin with no label other than A' "
   "do not use storage registers 0 through 8"
   "do not use certain operations like CLR or =".
   SOLVE allows the f(x) program to use everything in the calculator except the **SOLVE** key. Moreover, **SOLVE** may be invoked from another program just like any other key on the calculator; and f(x) can use the HP-34C's powerful $\int_y^x$ key.

A lot of thought has gone into making **SOLVE** conform to Albert Einstein's dictum: "As simple as possible, but no simpler."

How Does SOLVE Work?

The **SOLVE** key's microprogram uses very little of the HP-34C's resources. Reserved for **SOLVE**'s exclusive use are just five memory registers for data and a handful of other bits. Those five memory registers hold three sample arguments α, β, and γ and two previously calculated sample values $f(\alpha)$ and $f(\beta)$ while the user's f(x) program is calculating f(x) from the argument $x = \gamma$, which it found in the stack. How does **SOLVE** choose that argument γ?

Suppose α and β both lie close to a root $x = \zeta$ of the equation f(x)=0. Then a secant (straight line) that cuts the graph of f at the points $[x=\alpha\ ,\ y=f(\alpha)]$ and $[x=\beta\ ,\ y=f(\beta)]$ must cut the x-axis at a point $[x=\gamma\ ,\ y=0]$ given by

$$\gamma = \beta - (\beta-\alpha)\cdot f(\beta)/(f(\beta)-f(\alpha)) \qquad (1)$$

Provided the graph of f is smooth and provided ζ is a simple root, i.e., $f(\zeta) = 0 \neq f'(\zeta)$, then as Fig. 4 suggests, γ must approximate ζ much more closely than do α and β. In fact the new error $\gamma-\zeta$ can be expressed as

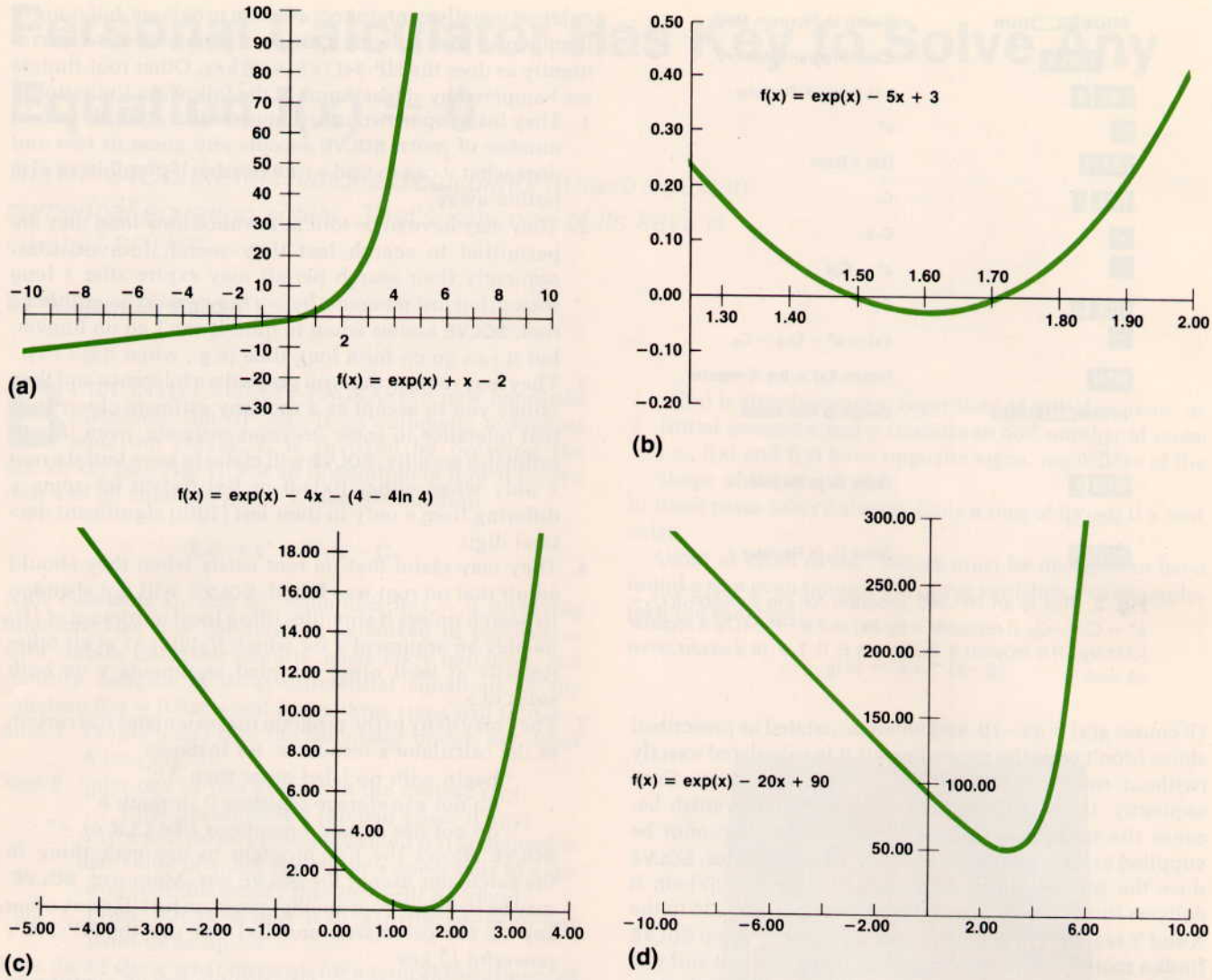

Fig. 3. *Examples of* SOLVE *results for different values of C_1 and C_2 and different first guesses for the root x in the program of Fig. 2. (a) If the first guess is −99 the root x = 0.442854401 is found in 25 seconds. The graph of f(x) on the negative-x side is relatively straight, so* SOLVE *works quickly. If the first guess is 99 the root is found in 190 seconds.* SOLVE *takes longer to get around a sharp bend. (b) With first guesses 0 and 2 the root 1.468829255 is found in 30 seconds. With first guesses 2 and 4 the root x = 1.74375199 is found in 20 seconds. Many root finders have trouble finding nearby roots. (c) With first guesses 0 and 2 the double root 1.386277368 is found in 50 seconds. Many root finders cannot find a double root at all. (d) Since no root exists,* SOLVE *displays* ERROR 6. *With first guesses of 0 and 10,* SOLVE *displays* ERROR 6 *in 25 seconds. After the error is cleared* SOLVE *displays 2.32677...., which approximates the place x = 2.99573.... where f(x) takes its minimum value 50.085.....*

$$\gamma - \zeta = K \cdot (\alpha - \zeta) \cdot (\beta - \zeta)$$

where K is complicated but very nearly constant when α and β both lie close enough to ζ. Consequently the secant formula, equation 1, improves good approximations to ζ dramatically, and it may be iterated (repeated): after $f(\gamma)$ has been calculated α and $f(\alpha)$ may be discarded and a new and better guess δ calculated from a formula just like equation 1:

$$\delta = \gamma - (\gamma - \beta) \cdot f(\gamma) / (f(\gamma) - f(\beta)) \qquad (2)$$

This process repeated constitutes the secant iteration and is the foundation underlying the operation of the **SOLVE** key.

A lot could be said about the secant iteration's ultimately rapid convergence, but for two reasons the theory hardly ever matters. First, the theory shows how strongly the secant formula (equation 1) improves good estimates of a root without explaining how to find them, even though the search for these estimates generally consumes far more time than their improvement. Second, after good estimates have been found, the secant iteration usually improves them so quickly that, after half a dozen iterations or so, the tiny calculated values of f(x) fall into the realm of rounding error noise. Subsequent applications of equation 1 are confounded by *relatively* inaccurate values $f(\alpha)$ and $f(\beta)$ that

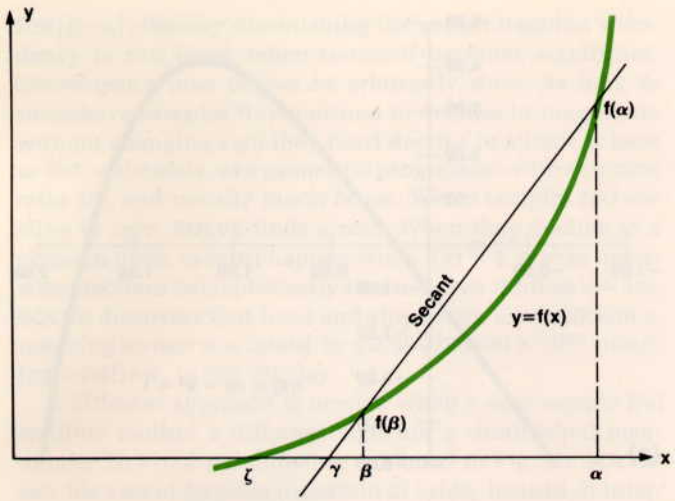

Fig. 4. *Given guesses α and β with corresponding function values $f(\alpha)$ and $f(\beta)$ the secant iteration produces a new guess γ by the formula $\gamma = \beta - (\beta-\alpha)\cdot f(\beta)/(f(\beta)-f(\alpha))$.*

produce a spurious value for the quotient $f(\beta)/(f(\beta)-f(\alpha))$. For these reasons the secant iteration is capable of dithering interminably (or until the calculator's battery runs down). Figs. 5a-5b show examples where the secant iteration cycles endlessly through estimates $\alpha, \beta, \gamma, \delta, \alpha, \beta, \gamma, \delta, \ldots$

Therefore, the secant iteration must be amended before it can serve the **SOLVE** key satisfactorily.

SOLVE cannot dither as shown in Fig. 5a because, having discovered two samples of f(x) with opposite signs, it constrains each successive new guess to lie strictly between every two previous guesses at which f(x) took opposite signs, thereby forcing successive guesses to converge to a place where f vanishes or reverses sign. That constraint is accomplished by modifying equation 2 slightly to bend the

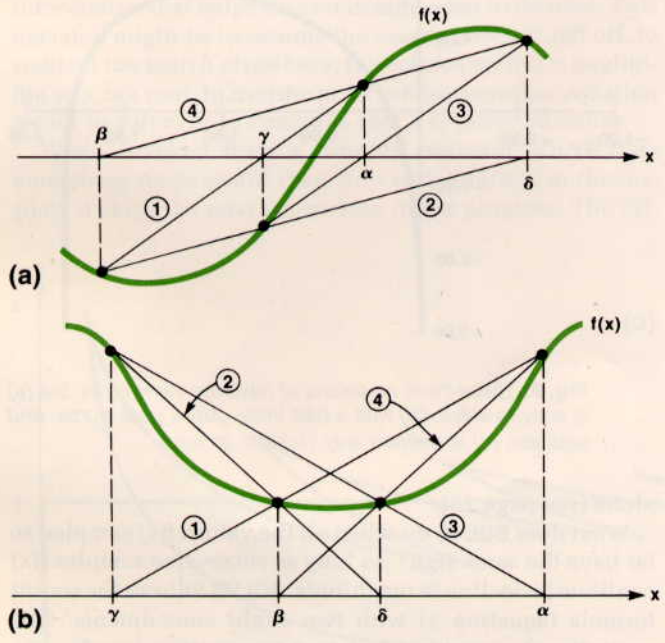

Fig. 5. *Examples of how the secant iteration can cycle endlessly through the values $\alpha, \beta, \gamma, \delta$. (1) $\alpha, \beta \rightarrow \gamma$ (2) $\beta, \gamma \rightarrow \delta$ (3) $\gamma, \delta \rightarrow \alpha$ (4) $\delta, \alpha \rightarrow \beta$ and so forth.*

Why Is Equation Solving Provably Impossible?

"The merely Difficult, we do immediately; the Impossible will take slightly longer." Old British naval maxim.

What makes equation solving merely difficult is the proper calculation of f(x) when the equation f(x) = 0 has to be solved. Sometimes the calculated values of f(x) can simultaneously be correct and yet utterly misleading. For example, let $g(x) \equiv x+2\cdot(x-5)$; this is the function whose calculated values change sign but never vanish. Next let the constant c be the calculated value of $(g(10/3))^2$; this amounts to $c = 10^{-18}$ on an HP handheld calculator, but another calculator may get some other positive value. Finally, let $f(x) \equiv 1-2\exp(-g^2(x)/c^2)$. The graph of f crosses the x-axis despite the fact that the correctly rounded value calculated for f(x) is always 1. None of the arguments x for which f(x) differs significantly from 1 can be keyed into the calculator, so it has no way to discover that f(x) vanishes twice very near 10/3, namely at

$$x = 10/3 \pm c\sqrt{\ln 2}/3$$

No numerical equation solver could discover those roots.

Worse, perhaps, than roots that can't be found are roots that aren't roots. Here is an example where the calculator cannot know whether it has solved f(x) = 0 or f(x) = ∞. Consider the two functions

$$f(x) \equiv 1/g(x) \text{ and } f(x) \equiv 1/(g(x) + c^2/g(x))$$

where g(x) and c are defined above. These two functions have identical calculated values, after rounding, for every x that can be keyed into the calculator, which consequently can't tell one from the other despite the fact that at x = 10/3 the first has a pole, f(10/3) = ∞, and the second a zero, f(10/3) = 0. Starting from straddling initial guesses x = 1 and x = 10 the **SOLVE** key finds a "root" of both equations f(x) = 0 to lie between 3.333333333 and 3.333333334 after only 49 samples. The user, not the calculator, must decide whether the place where f(x) changes sign is a root of f(x) = 0 or not. A similar decision arises when both initial guesses lie on the same side of 10/3, in which case **SOLVE** ultimately finds a "root" of f(x) at some huge x with $|x| > 3.33 \times 10^{98}$, where the calculated value of f(x) underflows to zero. That huge x must be regarded as an approximation to $x = \pm\infty$ where both functions $f(\pm\infty) = 0$.

The foregoing examples illustrate how our inability to perform calculations with infinitely many figures makes equation solving difficult. What makes equation solving impossible, even if rounding errors never happened, is our natural desire to decide after only finitely many samples of f(x) whether it never vanishes. Any procedure that claims to accomplish this task in all cases can be exposed as a fraud as follows:

> First apply the procedure to "solve" f(x) = 0 when $f(x) \equiv -1$ everywhere, and record the finitely many sample arguments $x_1, x_2, x_3, \ldots, x_n$ at which f(x) was calculated to reach the decision that f(x) never vanishes. Then apply the procedure again to $f(x) \equiv (x-x_1)\cdot(x-x_2)\cdot(x-x_3)\cdot(\ldots)\cdot(x-x_n)-1$. Since both functions f(x) take exactly the same value, −1, at every sample argument, the procedure must decide the same way for both: both equations f(x) = 0 have no real root. But that is visibly not so.

So equation solving is impossible in general, however necessary it may be in particular cases of practical interest. Therefore, ask not whether **SOLVE** can fail; rather ask, "When will it succeed?"

Answer: Usually.

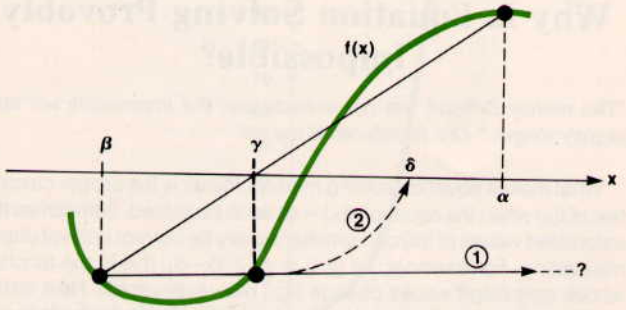

Fig. 6. *In the HP-34C, once two samples of f(x) with opposite signs have been discovered, the secant line (1) is bent to (2) whenever necessary to prevent an iterate δ from escaping out of the shortest interval known to contain a place where f(x) reverses sign.*

secant occasionally as illustrated in Fig. 6. Another small modification to compensate for roundoff in the secant formula (equation 1) protects it from the premature termination illustrated in Fig. 7. Although **SOLVE** can now guarantee convergence ultimately, that might not be soon enough since ultimately we all lose patience. Fortunately, convergence cannot be arbitrarily slow. At most six and normally fewer iterations suffice to diminish either successive errors $|x-\zeta|$ or successive values $|f(x)|$ by an order of magnitude, and rarely are more than a dozen or two iterations needed to achieve full ten-significant-digit accuracy. So fierce is the bent-secant iteration's urge to converge that it will converge to a pole (where $f(x)=\infty$) if no zero (where $f(x)=0$) is available, and this is just as well because poles and zeros cannot be distinguished by numerical means alone (see page 23).

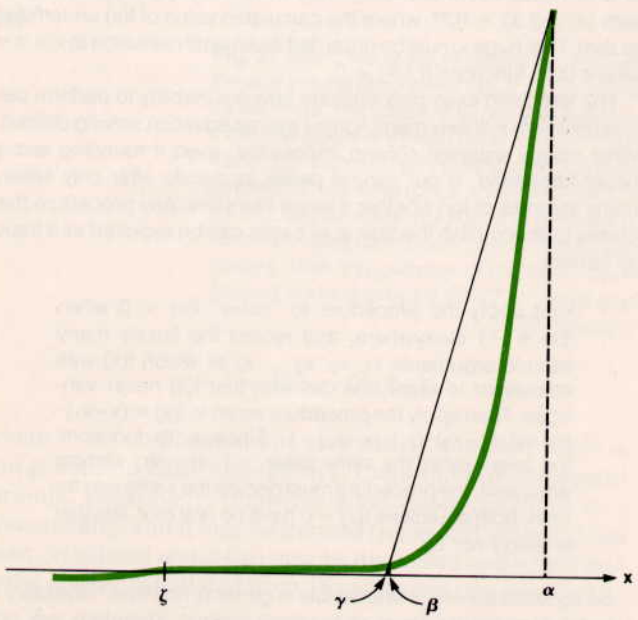

Fig. 7. *With a wild initial guess α the rounded value of γ may coincide with β. This convinces some equation solvers that γ is the root.* **SOLVE** *perseveres until it locates the root ζ correctly.*

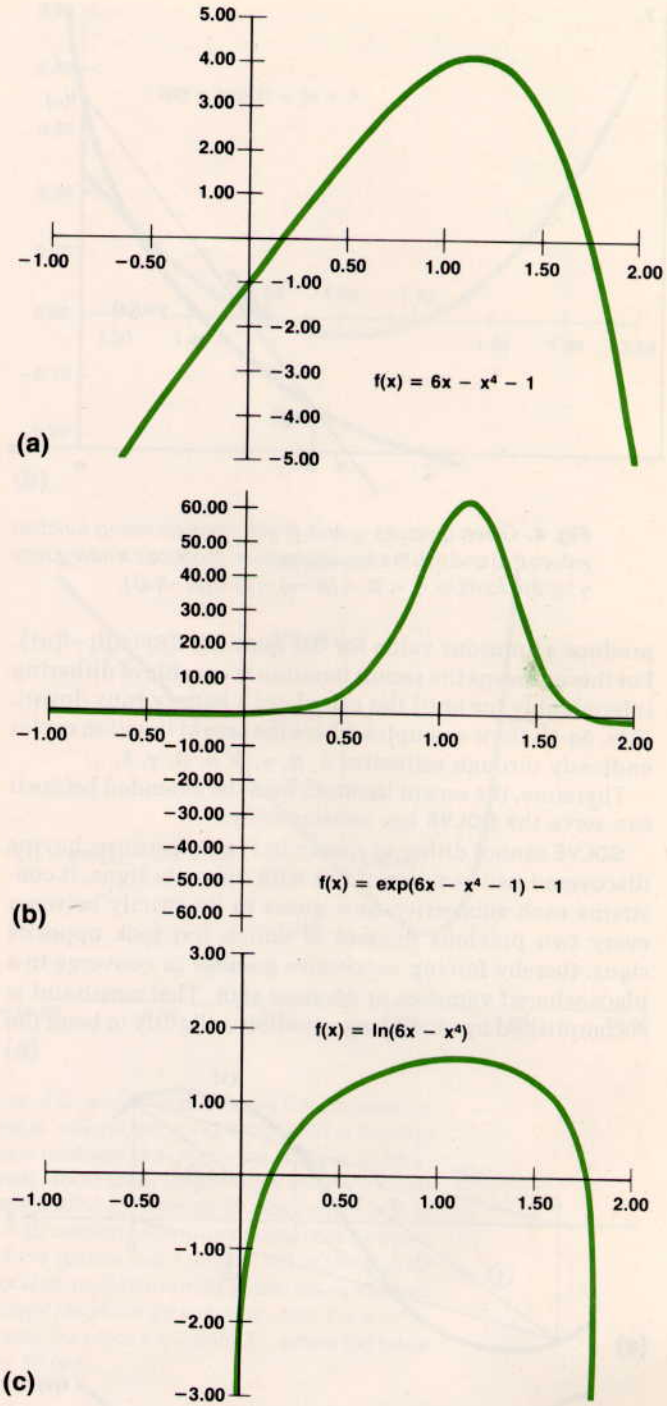

Fig. 8. *These three equations all have the same roots, but (a) is easy to solve, (b) with a bad initial guess gets worse, and equation (c) is defined only close to its roots.*

What does **SOLVE** do when all the values f(x) sampled so far have the same sign? As long as successive samples f(x) continue to decline in magnitude, **SOLVE** follows the secant formula (equation 1) with two slight amendments. One amendment prevents premature termination (see Fig. 7). The other deals with nearly horizontal secants, when $f(\alpha)=f(\beta)$ very nearly, by bending them to force $|\gamma-\beta|\leq$

$100|\beta-\alpha|$, thereby diminishing the secant iteration's tendency to run amok when roundoff becomes significant. Convergence now cannot be arbitrarily slow. As long as successive samples f(x) continue to decline in magnitude without changing sign they must decline to a limit at least as fast, ultimately, as a geometric progression with common ratio 1/2, and usually much faster. When samples f(x) decline to zero, **SOLVE** finds a root. When they decline to a nonzero limit, as must happen when $f(x) = 1 + e^x$ or otherwise declines asymptotically to a nonzero limit as $x \to \pm\infty$, **SOLVE** discovers that limit and stops with either ERROR 6, meaning no root was found, or $\pm 9.999999999 \times 10^{99}$, meaning overflow, in the display.

A different approach is needed when a new sample $f(\gamma)$ exhibits neither a different sign nor a diminished magnitude. To avoid the dithering exhibited in Fig. 5b, **SOLVE** sets the secant formula (equation 2) aside. Instead, it interpolates a quadratic through the three points $[\alpha,f(\alpha)]$, $[\beta,f(\beta)]$, $[\gamma,f(\gamma)]$ and sets δ to the place where that quadratic's derivative vanishes. In effect, δ marks the highest or lowest point on a parabola that passes through the three points. **SOLVE** then uses δ and β as two guesses from which to resume the secant iteration. At all times β and $f(\beta)$ serve as a record of the smallest $|f(x)|$ encountered so far.

But the parabola provides no panacea. Roughly, what it does provide is that if $|f(x)|$ has a relatively shallow minimum in the neighborhood of β and δ, the calculator will usually look elsewhere for the desired root. If $|f(x)|$ has a relatively deep minimum the calculator will usually remember it until either a root is found or **SOLVE** abandons the search.

The search will be abandoned only when $|f(\beta)|$ has not decreased despite three consecutive parabolic fits, or when accidentally $\delta = \beta$. Then the calculator will display ERROR 6 with β in the X register, $f(\beta)$ in the Z register, and γ or δ in the Y register. Thus, instead of the desired root, **SOLVE** supplies information that helps its user decide what to do next. This decision might be to resume the search where it left off, to redirect the search elsewhere, to declare that f(x) is negligible so x is a root, to transform $f(x)=0$ into another equation easier to solve, or to conclude that f(x) never vanishes.

When invoked from a running program **SOLVE** does something more useful than stop with ERROR 6 in the display: it skips the next instruction in the program. The calculator's user is presumed to have provided some program to cope with **SOLVE**'s possible failure to find a root, and then **SOLVE** skips into that program. This program might calculate new initial guesses and reinvoke **SOLVE**, or it might conclude that no real root exists and act accordingly. Therefore, **SOLVE** behaves in programs like a conditional branch: if **SOLVE** finds a root it executes the next instruction, which is most likely a **GTO** instruction that jumps over the program steps provided to cope with failure. Therefore the HP-34C, alone among handheld calculators, can embed equation-solving in programs that remain entirely automatic regardless of whether the equations in question have solutions.

Some Problem Areas

Equation solving is a task beset by stubborn pathologies; in its full generality the task is provably impossible (see page 23). Even though equations that matter in practice may not fall into the Chasm of the Impossible, yet they may teeter on the brink. Rather than leave the user teetering too, the HP-34C Owner's Handbook devotes two chapters to **SOLVE**, one introductory and one more advanced. The second chapter discusses equation solving in general rather than the **SOLVE** key alone, and supplies the kind of helpful advice rarely found in textbooks. Here follow examples of things that users might need to know but are unlikely to have learned except from bitter experiences, which the Handbook tries to forestall.

Hard versus Easy Equations. The two equations $f(x)=0$ and $\exp(f(x))-1=0$ have the same real roots, yet one is almost always much easier to solve numerically than the other. For instance, when $f(x)\equiv 6x-x^4-1$ the first equation is easier. When $f(x)\equiv\ln(6x-x^4)$ the second is easier. See Figs. 8a-8c.

In general, every equation is one of an infinite family of equivalent equations with the same real roots, and some of those equations must be easier to solve than others. If your numerical method fails to solve one of those equations, it may succeed with another.

Inaccurate Equations. Numerical equation solvers have been known to calculate an equation's root wrongly. That cannot happen to **SOLVE** unless the equation is wrongly calculated, which is what happens in the next example. This example resembles equations that have to be solved during financial calculations involving interest rates or yields on investments. For every $p \geq 0$ the equation

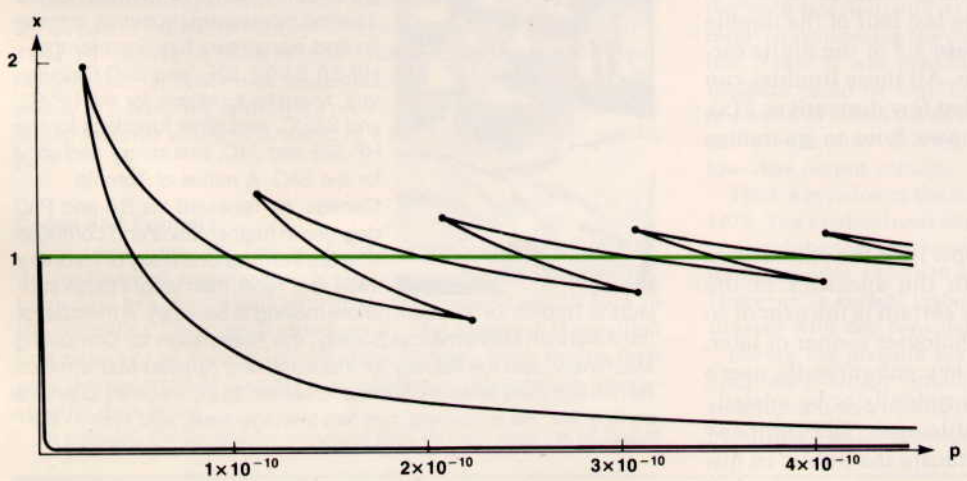

Fig. 9. *The jagged solid line is a graph of the ostensible roots of $x-(1-\exp(-xp))/xp = 0$ calculated carrying ten significant digits. The colored line is a plot of the correct root $x = 1$ (to nine significant digits) obtained by a rearranged calculation.*

$$x - h(px) = 0,$$

where $h(0) = 1$ and $h(z) \equiv (1 - \exp(-z))/z$ if $z \neq 0$, has just one root x, and $0 < x \leq 1$. The colored line in Fig. 9 plots this root x against p, and shows how smoothly $x \rightarrow 1$ as $p \rightarrow 0$. But when that root x is calculated numerically for tiny values of p using the most straightforward program possible, something awful happens, as shown by the black graph in Fig. 9. That serrated graph reflects the capricious way in which the calculated equation's left-hand side changes sign—once for $p = 10^{-11}$ at "root" $x = 10^{-88}$, seven times for $p = 2.15 \times 10^{-10}$ at "roots" $x = 4.65 \times 10^{-90}$, 0.233, 0.682, 0.698, 0.964, 1.163 and 1.181. All those "roots" are wrong; the correct root is $x = 0.999999999\ldots$. These aberrations are caused by one rounding error, the one committed when $\exp(-px)$ is rounded to 10 significant digits. Carrying more figures will not dispel the aberrations but merely move them elsewhere.

To solve $x - h(px) = 0$ correctly one must calculate h(z) accurately when z is tiny. Here is the easiest way to do that: if $\exp(-z)$ rounds to 1 then set $h(z) = 1$, otherwise set $h(z) = (\exp(-z) - 1)/\ln \exp(-z)$. This reformulation succeeds on all recent HP handheld calculators because the **LN** key on these calculators retains its relative accuracy without degradation for arguments close to 1 (see reference 1). Consequently, $\ln \exp(-z)$ conserves the rounding error in the last digit of $\exp(-z)$ well enough for that error to cancel itself in the subsequent division, thereby producing an accurate h(z) and a trustworthy root x.

Generally, wrong roots are attributable more often to wrong equations than to malfunctioning equation solvers. The foregoing example, in which roundoff so contaminated the first formula chosen for f(x) that the desired root was obliterated, is not an isolated example. Since the **SOLVE** key cannot infer intended values of f(x) from incorrectly calculated values, it deserves no blame for roots that are wrong because of roundoff. Getting roots right takes carefully designed programs on carefully designed calculators.

Equations with Several Roots. The more numerous the roots the greater is the risk that some will escape detection. Worse, any roots that cluster closely will usually defy attempts at accurate resolution. For instance, the double root in Fig. 3c ought to be $x = \ln 4 = 1.386294361$ instead of 1.386277368, but roundoff in the 10th decimal causes the calculated f(x) to vanish throughout $1.386272233 \leq x \leq 1.386316488$, thereby obscuring the last half of the double root's digits. Triple roots tend to lose 2/3 of the digits carried, quadruple roots 3/4, and so on. All these troubles can be attacked by finding where the first few derivatives f'(x), f''(x), etc. vanish, but nobody knows how to guarantee victory in all cases.

What Have We Learned?

The reader will recognize, first, how little the pathologies illustrated above have to do with the specifics of the **SOLVE** key, and second, how nearly certain is the user of so powerful a key to stumble into pathologies sooner or later, however rarely. While the **SOLVE** key enhances its user's powers it obliges its user to use it prudently or be misled.

And here is Hewlett-Packard's dilemma. The company cannot afford a massive effort to educate the public in numerical analysis. But without some such effort most potential purchasers will remain unaware of **SOLVE**'s value to them. And without more such effort many actual purchasers may blame their calculator for troubles that are intrinsic in the problems they are trying to **SOLVE**. To nearly minimize that required effort and its attendant risks, **SOLVE** has been designed to be more robust, more reliable and much easier to use than other equation solvers previously accepted widely by the computing industry. Whether that effort is enough remains to be seen. Meanwhile we enjoy the time **SOLVE** saves us when it works to our satisfaction, which is almost always.

Acknowledgments

I am grateful for help received from Dennis Harms, Stan Mintz, Tony Ridolfo and Hank Schroeder. Hank wrote the Handbook's chapters on **SOLVE**. Tony found ways to improve the **SOLVE** key's program while microcoding it. Dennis contributed some improvements too, both to the program and to this explanation of it, but I owe him most thanks for, along with Stan, supporting our efforts enthusiastically despite justifiable doubts.

Reference

1. D.W. Harms, "The New Accuracy: Making $2^3 = 8$," Hewlett-Packard Journal, November 1976.

William M. Kahan

William Kahan is professor of mathematics and computer science at the University of California at Berkeley. An HP consultant since 1974, he has helped develop increasingly accurate arithmetic and elementary functions for the HP-27, 67/97, 32E, and 34C Calculators, financial functions for the HP-92 and 38E/C, and other functions for the HP-32E and 34C, including ∫ and SOLVE for the 34C. A native of Toronto, Canada, he received his BA and PhD degrees in mathematics and computer science from the University of Toronto in 1954 and 1958, then taught those subjects at Toronto for ten years before moving to Berkeley. A member of the American Mathematical Society, the Association for Computing Machinery, and the Society for Industrial and Applied Mathematics, he has authored several papers and served as a consultant to several companies. He is married, has two teenage sons, and lives in Berkeley.

see http://www.cs.berkeley.edu/~wkahan/Math128/SOLVE.pdf