HP-12C's Serendipitous Solver

Valentin Albillo (Ex-PPC #4747, HPCC #1075)

Serendipity, the faculty of finding valuable things not sought for. Does this apply to the realm of HP calculators ? Yes, certainly, the most dramatic example being the discovery of 41C's *synthetics* and their many valuable and unforeseen uses. Now we'll see another striking example, a major capability making an appearance where you would least expect it.

Suppose you were asked to select the *best* and the *worst* machines to be used in *finding real roots* of polynomial equations of arbitrary degree, i.e:

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0 = 0$$

among the models in the Voyager series, i.e: HP-10C, 11C, 12C, 15C, and 16C. It's quite probable that you'd select the 15C as the *best* for the task, as it has the largest capacity, the best programming features, and a built-in equation solver. On the other hand, the 12C would be likely to be selected as one of the *worst* for the task, due to its small program memory and minimal programming features. Yet, against all odds, we'll see that the 12C is actually *best* for this particular problem ! *"How come ?"* you might say, *"The 12C has no built-in solver, unlike the 15C"*. Oh, but it actually *does* have one! *"Does it ? Where in the instruction set is it ?"*.

Well, the HP-12C's instruction set does indeed include a lot of *financial* formulas. But financial or not, in the end they are but *mathematical* formulas that implement mathematical algorithms. The fact that those algorithms do have financial uses is only "*in the eye of the beholder*". If we can see them in their mathematical purity, perhaps we may discover another uses for them unrelated to their intended one.

Thus enlightened, a thorough reading of the HP-12C's manual reveals that, given a series of *cash flows*, the **NPV** built-in function computes the resulting *Net Present Value* (NPV) using this neat formula:

$$NPV = CF_0 + CF_1/(1+r) + CF_2/(1+r)^2 + CF_3/(1+r)^3 + \ldots + CF_n/(1+r)^n$$

Where:
- NPV = Net Present Value of a discounted cash flow
- CFj = Cash Flow at period j
- r = i/100 = periodic interest rate expressed as decimal

Further, the 12C has another built-in function, **IRR**, which can compute the *Internal Rate of Return* (IRR) which, by definition, is the value of ***r*** which makes NPV = 0.

But, as it turns out to be, the above formula for computing the NPV *is a polynomial equation in the variable 1/(1+r),* and computing the IRR is equivalent to solving for the value of 1/(1+r) which makes NPV=0. In other words, *we can use the built-in solver IRR to automatically find a root of an arbitrary polynomial equation* in 1/(1+r). **IRR** will give us ***r***, then a simple change of variable will give us the value of **x**. Which is more, unlike the 15C's solver, **IRR** does *not* require the user to supply *any* initial guesses, but will compute the root *without any input from the user at all*.

So, we see that **IRR** can help us find *one root* of an arbitrary polynomial equation. Is that all ? No ! The notes on **IRR** at the end of the *Owner's Handbook* warn us to the possibilty of more than one real root and what to do about it. In particular, we are informed that the IRR internal algorithm can be invoked in such a way that *it accepts a user-supplied initial guess*, by using the rather odd-looking sequence **RCL g R/S** ! *This will allow us to find multiple roots for a given equation !*

But there's more: not only does **IRR** find roots for us, but we can use the **NPV** built-in function to *evaluate* the polynomial for arbitrary user-specified values ! And we can also use the D**%** and **%** functions to perform necessary *changes of variable* !

Now you may ask: *"How do we specify the coefficients of our equation ?"* That one is easy: the coefficients are simply entered as *cash flows* using the built-in **CF_0** and **CFj** functions. Further, we can use the built-in **Nj** function to enter *multiple consecutive equal coefficients*, and we can correct any *input errors* using the sequence **RCL g CFj**. What more could we ask for ? Let's summarize: we have concluded that the *financial* functions

- **CF_0** and **CFj** can be used to *enter the coefficients* of the polynomial
- **Nj** can be used to enter *multiple consecutive equal* coefficients
- **RCL g CFj** can be used *to correct input errors*
- **IRR** can be used to *find a root* of an arbitrary polynomial equation
- **RCL g R/S** can be used *to find additional roots* of the same equation
- **NPV** can be used to *evaluate* the polynomial at user-specified values
- D% and **%** can be used to perform the required *changes of variable*

We'll see how this all works in full detail in the comprehensive **Cases** included.

Commented Program listing

- R↓ and X<>Y are the “roll-down” and “X exchange Y” stack operations
- ∆% is the “percent of change” function

01	g CF_0	*stores a_n as the first cash flow*
02	EEX	*puts input-detection constant in X*
03	9	*(10^9 is the arbitrary detection constant)*
04	R/S	*stops for input of the coefficients*
05	g CFj	*stores the coefficients as cash flows*
06	EEX	*puts detection constant in X again*
07	9	*to test if another coefficient was input …*
08	-	… we *subtract and compare against zero*
09	g X=0?	*was the value in X equal to 10^9 ?*
10	g GTO 12	*yes, just R/S, done entering coefficients*
11	g GTO 02	*no, loop to store the new coefficient*
12	RCL g CFj	*discards the spurious coefficient stored*
13	1	*needed for the change of variable*
14	f IRR	*computes the auxiliary root*
15	%	*we perform a change of variable to get …*
16	+	*… the true root from the auxiliary root*
17	R/S	*stops to show the computed root*
18	ENTER	*needed to terminate numeric input !*
19	1	*for the necessary change of variable*
20	X<>Y	*we need 1 in Y, the initial guess X_0 in X*
21	∆%	*… and thus we obtain the initial guess*
22	RCL g R/S	*using it, this computes the auxiliary root*
23	X<>Y	*we place 1 in Y and the auxiliary root …*
24	R↓	*… in X, ready for the change of variable*
25	g GTO 15	*goes to make the change and show the root*
26	ENTER	*needed to terminate numeric input !*
27	1	*for the necessary change of variable*
28	X<>Y	*we need 1 in Y, the value of X in X …*
29	∆%	*and thus we obtain the changed X*
30	STO i	*stores the changed X value for evaluation*
31	f NPV	*evaluates the polynomial for the changed X*
32	g LSTX	*retrieves X for the change of variable*
33	RCL n	*retrieves the degree of the polynomial*
34	Y^X	*performs the last change of variable …*
35	*	*… so that we now have P(X) as desired*
36	R/S	*shows the computed value and accepts input*

```
37 g GTO 26    loops to compute P(X) for any new input
```

Usage instructions

A) To find a real root r > 0 of $a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_2 x^2 + a_1 x + a_0 = 0$, proceed as follows:

1) press: **f PRGM**
2) key in $\mathbf{a_n}$ and press **R/S**. You should see ***1,000,000,000*** on the display.
3) repeat step (2) for all coefficients: $\mathbf{a_{n-1}}, \mathbf{a_{n-2}}, \dots, \mathbf{a_0}$, pressing **R/S** after each of them. You should always see ***1,000,000,000*** after pressing **R/S**
4) after the last one ($\mathbf{a_0}$), press **R/S** a *second* time. The program will automatically proceed to compute and display the real root, if any.

B) To find additional roots of the same equation, proceed as follows:

1) *If the program is stopped just after finding and displaying a root*, simply key in your *initial guess* for the new root, and press **R/S**. The program will try and find (hopefully) a new and distinct root (if one exists) based on your initial guess.

2) to find further additional roots, repeat step (1) above with another initial guess.

or

3) *If the program is stopped somewhere else* (for instance, after evaluating the polynomial for some x arguments, see **case C** below) or after the program has given an **Error** finding the first root (because of multiple roots, for instance, see the **Notes** below), first you may need to press **CLX** to clear the **Error** display, if any, then press **g GTO 18**, and follow the instructions on step (1) above.

C) To evaluate the polynomial for some **x** argument (x>0), proceed as follows:

1) If the coefficients have been already entered, go to step (2) below. Else first, press **f PRGM** and enter all the coefficients, *without pressing **R/S** a second time* after entering the last one (else, the program would try to find a root now).
2) Once all the coefficients have been entered, press **g GTO 26**
3) key in your **x** argument, then press **R/S**. The program computes and displays the value of the polynomial for your given argument.
4) for additional **x** arguments, repeat step (3) above.

Notes and limitations

- As written, the program will find *positive* real roots (r > 0). To find *negative* roots as well, see **Case 4** below.
- After inputting each coefficient $\mathbf{a_n}$, the constant ***1,000,000,000*** will be displayed to acknowledge the entry. This is so that you can terminate input by simply pressing R/S a second time after entering the last coefficient $\mathbf{a_0}$,thus none of your coefficients should equal precisely ***1,000,000,000***. If some does, either simply rescale all coefficients by 10, or else change the **EEX, 9** instructions in *steps 2,3 and 6,7* to some other suitable construct (say, **9, LN**. Any two-step expression which gives an unlikely constant can be used, but **EEX 9** is fastest).
- If the equation has multiple real roots or no positive root (>0) at all, you may get an **Error** display when finding a root. Simply follow the steps of **case B** above, but first have a look at **Case 4** below. A root exactly equal to 0 may give **Error 7**
- You can *correct input errors* easily, if immediately noticed. See **Case 1** below
- If all coefficients are distinct, *equations up to 14th degree* are possible. However, if the equation has repeated consecutive coefficients you can enter groups of up to 99 coefficients (e.g.: zeros) very easily. See **Cases 2,3** below. This way, finding roots for *equations of up to 1480th degree or more* is possible !
- *The range of x arguments for evaluation is x>0*. To evaluate for x=**0** use x=**1E-9** instead. For *negative* arguments use the same technique seen in **Case 4** below to find negative roots.

Case 1: Correcting input errors

Find a root of: $x^3 + 2x^2 + 10x - 20 = 0$

This is the historically famous *Leonardo de Pisa*'s equation, and it will serve us well to demonstrate how to *correct input errors*. We'll *erroneously* enter ***1*** for the coefficient of the ***x*** instead of ***10***, then we'll immediately correct our mistake on the fly. Enter the coefficients and correct the mistake as follows. Press:

`f PRGM, 1, R/S, 2, R/S,` **`1`**`, R/S`	[*oops ! mistake ! it should be **10***]
`RCL g CFj`	[*backs up last coefficient*]
`10``, R/S`	[*enters correct coefficient*]
`20, CHS, R/S,` **`R/S`**	[*solves for the root*]

As you can see, the sequence ***RCL g CFj*** backs up the *last* coefficient entered, so it can be reentered again. You can use it repeatedly to back up more than one coefficient, which can be useful if you entered several wrong coefficients in a row, or noticed a wrong coefficient after having entered some more.

After the final R/S the solver gets to work and just *11 seconds* later it finds:

`x = 1.368808108` [*press **f** 9 to see all decimal digits*]

Case 2: Equation with groups of repeated, consecutive coefficients

Find a root of: $x^7 + 2x^6 + 2x^5 + 2x^4 + 5x^3 + 5x^2 + 5x - 25 = 0$

This equation features two *groups* of three *repeated*, *consecutive* coefficients, namely ($2x^6$, $2x^5$, $2x^4$) and ($5x^3$, $5x^2$, $5x$). Our program can take advantage of the fact, so we will save entering all repeated coefficients but the first, we'll use less storage registers, and the root will be found faster as well.

Now, let's enter the coefficients and solve for the root as follows. Press:

```
f PRGM, 1, R/S, 2, R/S,   3, g Nj,
                 5, R/S,   3, g Nj,
                25, CHS, R/S,  R/S
```

Notice that the repeated coefficients have been entered as a ***2*** with ***3*** occurences and then a ***5*** with, again, ***3*** occurrences. After the final R/S, the solver proceeds to compute the root and *only 18 seconds* later it finds:

`x = 1.041948351`

Case 3: Very high degree equation with many repeated coefficients

Find a root of the 137th-degree(!!) equation: $x^{137} + 3x^{56} + 8x^2 + 5x - 2002 = 0$

This example perfectly illustrates how simply can we deal with *large groups* of *equal consecutive* coefficients. In this case, although we're dealing with a very high-degree equation, it is quite *sparse* with many zero coefficients. So much so that among the 138 coefficients only *five* are non-zero, namely ***1***, ***3***, ***8***, ***5***, ***-2002***.

We'll take advantage of this fact and won't enter most zero coefficients. First, we notice that the equation written in full would be like this:

$$x^{137} + (80 \text{ zero coefs.}) + 3x^{56} + (53 \text{ zero coefs.}) + 8x^2 + 5x - 2002 = 0$$

[no need to write down or count zeros: 80 = (137-56)-1 and 53 = (56-2)-1]

so we enter the coefficients and solve for the root as follows. Press:

```
f PRGM, 1, R/S, 0, R/S, 80, g Nj, 3, R/S,
                0, R/S, 53, g Nj, 8, R/S,
                5, R/S, 2002, CHS,   R/S, R/S
```

The final R/S without input signals that all coefficients have been entered, so the solver proceeds at once to search for the root and after just <u>*3 min. 20 sec*</u>. it stops, with the newly found root in the display:

x = ***<u>1.056741318</u>***

Case 4: Finding <u>several distinct real roots</u> of an equation

Find <u>all five real roots</u> of the quintic: $16x^5 - 180x^3 + 405x - 136 = 0$

This equation should be old hat to all readers of my article *"HP-12C Tried & Tricky Trigonometrics"* featured in ***Datafile V21 N1***. There, we found its five roots on a 12C using trigonometrics. Here, we'll let the solver do the dirty work instead.

First, we'll proceed to reset the program pointer to the beginning of the program, then we'll enter all the coefficients. Press:

```
f PRGM,   16, R/S, 0, R/S, 180, CHS, R/S, 0, R/S,
          405, R/S, 136, CHS, R/S, R/S
```

After only <u>*2 seconds*</u>, we'll get ***Error 3*** in the display, signaling the possible presence of *several* real roots. To find them all, proceed as follows. Press:

CLX [*to clear the **Error 3** display*],
g GTO 18 [*the entry point that allows initial guesses*],

And now we'll supply suitably different initial guesses to compute all three *positive* real roots. Press:

1, R/S:	***<u>1.463277004</u>***	[*1st positive root,* <u>*17 seconds*</u>]
2, R/S:	***<u>2.942932637</u>***	[*2nd positive root,* <u>*13 seconds*</u>]
0.1, R/S:	***<u>0.355555392</u>***	[*3rd positive root, 31 seconds*]

To obtain the *negative* real roots, we need to change the variable ***x*** for ***-x***, which for this particular example means changing the sign of just the last coefficient *(-136, stored in R5*), and then change back the signs of the computed roots. Press:

RCL 5, CHS, STO 5 [*changes the sign of the last coefficient*],

1, R/S, **CHS**: ***-2.038577713*** [*1st negative root, <u>25 seconds</u>*]
3, R/S, **CHS**: ***-2.723187320*** [*2nd negative root, <u>12 seconds</u>*]

Case 5: <u>Evaluating a polynomial</u> for given *x* arguments

Evaluate the polynomial P(x) = 16x⁵ -180x³ +405x -136 for x=e, 1/3, Ö(5)

This polynomial is the same whose five roots we found earlier. Now we'll use it to illustrate how to *evaluate* the right side of an equation for given **x** arguments. First, we'll assume the coefficients haven't been entered yet, so we'll enter them now. Press:

f PRGM, 16, R/S, 0, R/S, 180, CHS, R/S,
0, R/S, 405, R/S,
136, CHS, R/S [*watch out ! <u>no second R/S!</u>*]

Be careful *not to press R/S a second time* after entering the last coefficient, as this would automatically start the root-searching process, and we don't want to find any roots here but simply evaluate the polynomial for several ***x*** arguments. To that effect, press now:

g GTO 26, [*entry point for evaluations*]

1, g e^x, R/S: ***-275.8819605*** [*P(e), <u>3 seconds</u>*]
3, 1/x, R/S: ***-7.600823057*** [*P(1/3), <u>3 seconds</u>*]
5, g √x, R/S: ***-348.4264577*** [*P(Ö(5)), <u>3 seconds</u>*]

Final Remarks

Well, I think you'll agree with me that the capabilities and uses of the built-in 12C's IRR solver really are a case of *serendipity* indeed, and it qualifies as the *best* and *fastest* solver in the Voyager series for finding a root of polynomial equations. If in doubt, just try the above examples on your HP-15C, say, and check your times against the ones shown. See ? I told you …