Boldly Going ... Going Back to the Roots

Valentín Albillo (HPCC #1075, PPC #4747)

Welcome to a new a **Boldly Going ...** article, this time paying homage to the brand new offspring in the ever-growing HP calculator family, the **HP35s**, a very significant, very interesting model which essentially is HP's attempt to go back to the roots by releasing a model which largely draws from classic look and feel. For my money, they've essentially succeeded and while the **HP35s** does have its share of valid criticism, the raw facts are that it is indeed a worthwhile addition to the lore, all the more interesting both for its strong points and its shortcomings.

A future **Long Live ... !** article will deal with both pretty soon (circumstances permitting), but for now, if HP can go back to the roots in the hardware side, it's only natural that we would do likewise in the software side, right ? Let's try !

Finding complex roots of complex equations is a complex business

Most specially when the built-in Solver won't do it *per se*. Though the **HP35s** includes pretty decent complex number handling to the point that each and every register (direct, indirect, stack) can hold a complex value and many arithmetic and transcendental functions are defined to work with them, there are also many others that aren't, and in particular you can't generally use the built-in Solver to find complex roots of arbitrary equations. This means that if you need to solve

$$2x^4 + 3x^3 + 4x^2 + 5x + 6 = 0 \qquad \textit{\{ 4 complex roots \}}$$

or

$$(2+3i)x^3 - (1+2i)x^2 - (3+4i)x - (6+8i) = 0 \qquad \textit{\{ 1 real, 2 complex roots \}}$$

or even

$$Sin(2x - 4i) + 3x^2 - (1+5i) = 0 \qquad \textit{\{ infinite complex roots \}}$$

you're definitely out of luck. But that sad state of affairs ends right now.

This small program I've written anew specifically for the **HP35s** will allow you to Boldly Go where no **HP35s** has gone before and find *a real or complex root of an arbitrary equation* with real and/or complex coefficients starting from just one real or complex initial guess. The root will be displayed as *labeled output* and left both in the **X** stack register and direct register **X**. Roots will be returned as *genuine* real/complex values as appropriate, i.e., a computed *real* root will be a proper *real* value, *not* a complex value with a zero or very small imaginary component.

Further, a *real* initial guess may find a *complex* root and vice versa. The program implements an optimized version of an advanced, *cubically*-convergent numerical method that typically converges *very quickly* to a root with speed comparable to that of the built-in Solver, and which, unlike Newton's method, will foray into the complex domain if need be, even starting from a real initial guess.

Program listing for the HP 35s

This small, 48-step **RPN** routine for the **HP 35s** will allow you to find real and/or complex roots of any *equation* or *program* you care to define under **`LBL F`** below:

```
A001  LBL A
A002  REGX*(1i0▷Z)▷X
A003  SQ(1E-4▷S)▷T
A004  0.5▷Y
A005  XEQ F001
A006  RCL/ Y
A007  STO U
A008  RCL S
A009  STO+ X
A010  XEQ F001
A011  STO V
A012  RCL S
A013  STO- X
A014  STO- X
A015  XEQ F001
A016  STO W
A017  RCL+ V
A018  RCL- U
A019  RCL/ T
A020  RCL V
A021  RCL- W
A022  RCL S
A023  STO+ X
A024   /
A025  RCL* Y
A026  STO W
A027   /
A028  STO V
A029  RCL* U
A030  RCL/ W
A031  RCL- Z
A032   +/-
A033  RCL Y
A034   y^x
A035  RCL- Z
A036  RCL/ V
A037  STO+ X
A038  RCL/ X
A039  ABS
A040  RCL T
A041  X<Y?
A042  GTO A005
A043  ABS(SIN(ARG(X)▷W))
A044  X<Y?
A045  SGN(COS(W))*ABS(X)▷X
A046  RCL X
A047  VIEW X
A048  RTN
F001  LBL F
F002  RTN
```

Notes:

- Lines *A002*, *A003*, *A004*, *A043*, and *A045* hold *equations*, so you should press **`EQN`** prior to keying them in. All include *store* operations (the "▷" symbol) which are entered by the **`STO`** key sequence. *A003* includes a **`+/-`**
- All "*" and "/" symbols are the multiply/divide operation, respectively.
- It uses *no* indirect registers, *no* flags, leaves direct *registers* ***`A-R`*** *free* for other purposes, and last-but-not-least, it works in any angular mode.
- Though they'll probably differ from yours due to the infamous *checksum bug*, f.t.r. these are my checksums for the above program and equations:

Program or Equation	Length	Checksum
LBL A	213	8501
A002	14	3025
A003	12	AE07
A004	5	7F8C
A043	18	412A
A045	20	4A9D

Usage instructions

1. This program is to be run in **RPN** mode so make sure the correct mode is active. Also, keep flag **10** cleared so that equations are *evaluated*, not *displayed*.

2. You can solve both *equations* and *programs.* The variable being solved is always X and your equation or program must take the X value from direct register **X** (*not* the display) and return the function value to the **X** stack register.

3. To define your equation, insert it right after line ***F001*** **LBL F**, using X as the variable to solve, and terminate the definition with a **RTN** instruction at line *F003*, like this example (2nd sample equation in the intro):

```
F002 2i3*X^3-1i2*X^2-3i4*X-6i8
F003 RTN
```

4. If you're solving a *program*, enter its lines after line ***F001*** **LBL F**, using direct register **X** (*not* the display, **X** stack register) to compute the functional value which should be left in the **X** stack register, finishing with a **RTN** instruction. For instance, to solve $x^x = \pi$ your **RPN** program would be:

```
F001   LBL F
F002   RCL X   {recall the X value to the X stack register}
F003   ENTER   {duplicate it in the Y stack register}
F004   y^x     {compute X^X}
F005   π       {place Pi in the X stack register}
F006   -       {compute X^X-Pi and leave the result in stk X }
F007   RTN     {return the result to the calling program}
```

   Your program can use any direct registers from **A** to **R** for its own purposes, as well as *all* indirect registers, *all* flags, and whatever display and angular modes it needs, plus any labels save **A** and **F**. Display mode has no effect on accuracy.

5. Enter a suitable *initial guess* in the display (**X** stack register) and execute the program (**XEQ A**). Unlike the built-in Solver, you only need to supply a *single* guess (not two), which can be real or complex and allows you to find other roots if they exist by varying it, as normally the closest root will be returned.

   A *real* initial guess will usually result in the closest *real* root being found, but if there are none nearby, or if the given equation has no real roots, it can and will find the closest *complex* root instead. Likewise, a *complex* initial guess will usually produce a *complex* root, but it can find and return a *real* root if no complex roots are nearby or the equation doesn't have any. In short, *any kind of guess can return any kind of root, irrespective of their real/complex type.*

   initial guess, XEQ A [ENTER] → X = computed root

6. After a while the root (real or complex) is labeled and output, remaining both in direct register **X** *and* in the display (**X** stack register), for you to store it somewhere else or use it right away in further computations. If desired, you can *check* that it is indeed a root by evaluating your equation or program right after finding it. With the root still in direct register **X** (the content of stack register **X** doesn't matter), simply press:

   XEQ F [ENTER] → value at root *{ should be zero or near zero }*

7. To try and find a *different* root, go to ***step 5*** and enter a different initial guess. To solve another *equation*, go to ***step 3***. To solve another *program*, go to ***step 4***.

Notes:

- As your equation or program will be called with *complex* values for X, you must use in your definition *only* those functions and operations which admit complex values as arguments, else the program will stop with an **INVALID DATA** message in the display as soon as a non-supported operation is encountered. Regrettably, non-supported **HP35s** complex operations include such common functions as $\sqrt{x}$ and x^2. You can replace x^2 by **X^2** or **ENTER**, ***** and $\sqrt{x}$ by **X^Y** or **RCL Y**, y^x, because, as an added convenience, direct register **Y** contains the constant **0.5** at all times while the program is running.

  See page ***9-3*** in the ***User's Guide*** for a comprehensive list of those functions and operations which work with complex values.

- Though unlikely, the algorithm might fail to converge in rare occasions. In that case simply stop the program and try a different initial guess.

- It's also possible to stumble upon a **DIVIDE BY 0** error which would halt the program. In that case try a different initial guess. This might happen for trivial 1st degree or *constant* polynomial equations (which are in no need for a full solver treatment anyway) or if either the initial guess or some intermediate X value happens to make some derivatives of the solved function equal zero. This is infrequent, however, and just slightly changing the initial guess will do in most cases.

- You must *never* delete line ***F001*** **LBL F** lest you risk the built-in automatic renumbering *wrongly* updating the **XEQ** instructions at lines *A005*, *A010*, and *A015* to point somewhere else. It shouldn't happen but I've seen it happen at least twice so I think a *caveat emptor* is in order.

  Should you delete it accidentally or if the program misbehaves, check those lines to ensure the **XEQ F001** instructions are *unchanged*. Likewise, never try to "*optimize*" the **XEQ F001** instructions to **XEQ F002**, for the same reason.

Programming techniques

- Powerful as it is, the **HP35s** is nevertheless rather a *slow* machine, most specially when *evaluating equations* (this includes *numeric constants* as well, be they real values, complex values, or vectors !) because they aren't syntactically checked until evaluation time so that they can be used to display messages as well. Upon evaluation, every character has to be *parsed*, *recognized* as a valid identifier, then eventually *executed*. If the equation is within a loop this time-consuming process gets redone anew *every time*.

  Thus, it's good programming practice to *avoid* using equations within loops altogether. They are best left for *non-iterative* sections of the program, such as *initialization* and *output*, which usually get done just once. That's the case in the listing above, where the main loop from *A005* to *A042* contains *just pure* **`RPN`** *code* for maximum speed, while the initialization section (*A001-A004*) and the output section (*A043-A048*) contain 5 equations in all. They allow for much more *concise* code, and the time penalty is irrelevant there.

- Program lines *A043-A045* constitute a very small but clever routine which makes sure a *real* root is returned as a *genuine real value*, not a complex value with zero or very small imaginary component. A threshold is tested and, if met, the complex value is converted to a properly *signed* real one.

  This is specially useful since there is *no* built-in command to extract the real component of a complex root and, if left as a complex value (with a zero or small complex component), many common functions won't accept it as a valid argument ($\sqrt{x}$ or x^2, for instance), complicating its further use. Not to mention its ungainly aspect in the display and diminished readability.

Assorted Examples

1. Find a root of : (a) $x^x = p$, (b) $x^x = i$

We'll solve both cases with a single, generalized equation depending on a free parameter (**`R`**) defined at line *F002* (don't forget to press **`EQN`** first):

```
      F002 X^X-R                                          {LN=5, CK=3D1C}
```

Now, let's solve (and check) both particular cases (assume **`ALL`** display):

```
p , STO R,  2,   XEQ A →  X = 1.85410596792   {4 seconds}
                 XEQ F →     -0.00000000001

i, STO R,    XEQ A →  X = 1.36062487029 i 1.11943916624
{it took 9 secs}  XEQ F →     4.45661923132E-12 i 0
```

2. Find all roots of : $(2+3i)x^3-(1+2i)x^2-(3+4i)x-(6+8i)=0$

Replace the equation at line *F002* (if any) by this equation:

```
F002 2i3*X^3-1i2*X^2-3i4*X-6i8      {LN=25, CK=AA4C}
```

As this equation is a 3rd degree polynomial, it will have *exactly 3 roots*, which we'll now proceed to find (assume **FIX 5** display):

```
    1, XEQ A →  X =  2.00000              {15 seconds}
   -1, XEQ A →  X = -0.70473 i –0.91388   {15 seconds}
1 i 1, XEQ A →  X = -0.67988 i  0.99081   {21 seconds}
```

Notice that although the equation, being a 3rd degree polynomial, must have *exactly* 3 roots, they do *not* necessarily come in complex *conjugate pairs*, as can be seen here; that's only the case for real-coefficient polynomial equations while the present one has *complex* coefficients. Further, despite its complex coefficients, the very first root found happens to be *real !.*

Also note that:

- in the case of the 1st root, a *real* guess has produced a *real* root
- in the case of the 2nd root a *real* guess has produced a *complex* root
- lastly, for the 3rd root a *complex* guess has produced a *complex* root.

Producing a *real* root from a *complex* guess is also possible as we'll see in the very next example.

3. Attempt to find a complex root of: $x^3-6x-2=0$

Replace the equation at line *F002* (if any) by this equation:

```
F002 X^3-6*X-2                     {LN=9, CK=EA16}
```

As we're trying to find a *complex* root it would seem fairly natural to start with a *complex* guess (assume **ALL** display):

```
2 i 3,  XEQ A →  X = 2.60167913189
        XEQ F →     -0.0000000001
```

but as you may see we've got instead a *real* root, thus demonstrating the 4th case mentioned before, i.e.: a *complex* guess can produce a *real* root. This particular equation has *no* complex roots, all its three roots are indeed *real.*

4. Solve Leonardo di Pisa's equation: $x^3 + 2x^2 + 10x - 20 = 0$

Replace the equation at line *F002* (if any) by this equation:

```
F002 X^3+2*X^2+10*X-20                    {LN=17, CK=73A0}
```

Being a 3rd degree polynomial equation with real coefficients, we know that it must have at least one real root and either a conjugate pair of complex roots or two additional (not necessarily distinct) real roots. Assuming **ALL** display:

```
 1, XEQ A →  X =  1.36880810782                {5 seconds }
-6, XEQ A →  X = -1.68440405391 i –3.4313313502  {15 seconds}
```

and we know the remaining root must be the complex conjugate of the 2nd one, thus it automatically is **X = -1.68440405391 i 3.4313313502** and no further computation is required (any number of first guesses would produce it if desired, **-2 i 3** for instance).

By the way, as the equations are defined for this program in a way compatible with the built-in Solver, we can check how **SOLVE** does with this example:

```
FN= F, 1, STO X, SOLVE X →  X =  1.36880810782  {5 seconds}
```

which completely agrees with this program's result and takes essentially the same time to find the root. Of course **SOLVE** cannot cope with the complex roots so testing that case is simply not possible.

5. Find several complex roots of : $Sin(2x - 4i) + 3x^2 - (1 + 5i) = 0$

Replace the equation at line *F002* (if any) by this equation:

```
F002  SIN(2*X-i4)+3*X^2-1i5          {LN=21, CK=A8EE}
```

Let's try several different initial guesses (assume **FIX 5** display):

```
 0, XEQ A →  X =  0.76368 i 1.11805     {18 seconds}
-1, XEQ A →  X = -1.37126 i 0.50438     {13 seconds}
 p , XEQ A →  X =  2.32883 i 0.29914     {16 seconds}
```

It seems likely that this *transcendental* equation has an *infinite* number of *complex* roots and we've found some of the smallest in absolute value.

Remember that although we're displaying them to 5 decimal places, they are found to *full* accuracy, and you can check them with **XEQ F** if desired.