Long Live the Advantage ROM !

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

No small part in the success of the HP-41C family, the possibility of increasing the features in the basic machine by means of *plug-in ROM modules* was in fact an essential key point in the overall product philosophy. You'd begin with a fairly complete intruction set, and could then *expand* it as needed with extra functionality which would be instantly available by simply plugging the module in. This would also deter obsolescence, allowing the addition of features available in newer models.

By 1985, the 41C family was firmly stablished as the most successful and arguably most useful of all HP calculator-type products. But the Voyager series, while not as powerful, did include *state-of-the-art functionality* dispersed in specialized models, such as the all-powerful scientific HP-15C, with its fast matrix capabilities, automated Solve and Integrate algorithms, and complex number handling; the computer science HP-16C, with its amazing abilities to work with numbers in various bases and word sizes, and the business *belle*, the HP-12C, with its powerful financial functions.

The 41C had none of this built-in, and though most of those advanced functionalities could be programmed in user code (and for the most part they were), they lacked the convenience and speed of their Voyager *m-code* counterparts. Once data were input, any HP-15C user could easily solve a 7x7 system of linear equations in 20 seconds flat, using 13-digit internal precision, and taking up a single step of program memory and no additional resources such as auxiliary registers, etc. In contrast, even the most efficient 41C user code programs were *slow* in comparison, less accurate, needed auxiliary registers which had to be allocated for them, and typically took *hundreds* of steps of program memory. Thus the 41C owner, while feeling rightfully proud, would nevertheless look with envious eyes to the Voyager models, and it wasn't infrequent that 41C owners would also buy one of them, as a convenient backup and complement.

It seems that this state of affairs made its way up to HP, and a very special ROM was developed to cater for most of the missing functionality and fight obsolescence, the **Advantage ROM**. It was the first HP-produced 12K ROM, all previous ones being 4K/8K. Bank-switching made it possible, transparently to the user, which would get a tremendous amount of advanced functionality all in a single ROM, taking up only a single port. This included *outstanding matrix-handling capabilities*, Solve and Integrate, operations with complex numbers and vectors, coordinate transformations, number conversions to various bases and boolean logic operations, as well as curve fitting, basic Time Value of Money functionality, and even numerical solving of first

and second-degree differential equations. Further, many of those advanced features were written in 41C *machine language*, and so ran at the *fastest* possible speed while taking the *least* user resources, thus maximizing its usefulness both from the keyboard and in user programs. To top it all, HP made it easy for all 41C users to get an Advantage ROM, by giving away one *for free* with every 41C sold as part of a sales promotion. 41C users needed never look at Voyager users with envious eyes again.

As this is not a full review, but a commemorative article, I'll focus on a single but important design issue: ***compatibility***. It's interesting to note that though HP gave 41C users many of the Voyager series capabilities, *they didn't aim for exact compatibility*, so that existing programs or techniques could be ported. Let's consider the case of matrix functionality: HP could have commissioned **Firmware Specialists** to adapt HP-15C matrix operations to the 41C, as they did for the root-finding and numerical-integration routines. That would have ensured maximum portability but they opted instead for **W&W Software Products' CCD ROM** existing matrix routines, thus completely *losing* all direct compatibility. This bold decision was probably based on costs: the CCD ROM matrix routines were *already* implemented for the 41C architecture, and it was just a matter of pulling them out straight from the CCD ROM and placing them as a 4K block in the new Advantage ROM, so HP had to pay just for the rights to *use* them, not for adapting or rewriting HP-15C routines to the 41C.

Anyway, despite the loss of compatibility, the decision was *sound*: the matrix routines included in the Advantage ROM are way more powerful and comprehensive than the ones in the HP-15C. Whereas the 15C includes some 30 different matrix operations, the Advantage ROM boast more than *twice as many*, including two full-featured matrix editors for real and complex-valued matrices, as well as a fully user-friendly program for the most usual operations. Furthermore, the support for complex-valued matrices is much more extensive than the extremely primitive 'transformation' functions available on the 15C. The fact that the Advantage ROM allows the user to dimension matrices either in Extended Memory, where they consume *no* numbered user registers, or else allocated over a user-defined range of numbered user registers, makes it extremely easy to *integrate* matrix handling with other user programs, unlike the 15C's paradigm, where matrices reside in a *separate* pool of memory, and so their elements *cannot* be accessed as normal, numbered registers.

Additionally, the Advantage ROM matrix function set includes lots of functionality unavailable to 15C users, such as the capability to return maximum and minimum elements in a matrix, both value *and* position (useful for sorting), matrix arithmetic between corresponding elements extending to multiplication and division, returning *sums* for all columns, rows, or the whole matrix, the possibility of *copying* all or part of a source matrix to a target matrix, *exchanging* rows or columns within a matrix, traversing a matrix forward and backwards by rows or columns while autoincrementing or decrementing the indexes, and many other assorted utility functions. All of this

essentially means that power user will have to *rewrite* most HP-15C programs in terms of the matrix functions available in the Advantage ROM, but will find a *larger* function set at hand for the task and can even make big *improvements* on the fly easily.

Nevertheless, as it's to be expected, the HP-15C matrix instruction set, however modest in comparison to the Advantage's, *does* include some nice touches which the later doesn't, such as the computation of the Residual (**MATRIX 6**, useful to refine iterative processes), which the 15C does in a *single* instruction, at full speed and with 13-digit precision. This can't be done using Advantage ROM's functions because intermediate results are rounded to 10 digits. In the 15C you also have the possibility of computing the result of multiplying the transpose of a matrix by another one in a single operation (**MATRIX 5**), and there's the concept of *matrix descriptors*, which can be stored in the stack or registers, to be used in logical tests, or even for indirect manipulation of matrices. Also, the clever implementation of "user" **STO** instructions makes it possible to store a value in a matrix element, automatically increment the indexes to point to the next one, and test for loop termination, all with a *single* instruction. The Advantage ROM equivalents do not act as test operators but simply affect flags 9 and 10, and it's up to the user to make the test and branch accordingly.

To better illustrate some of the matrix capabilities of the Advantage ROM, as well as to show some of the compatibility issues just mentioned, here`s a program I wrote specifically for the purpose.

The program

FP (short for **F**it **P**olynomial) is a small, user-friendly, fully-prompting 62-line program (124 bytes) that I wrote to specifically demonstrate the excellent matrix capabilities of the Advantage ROM, and further, to compare them to the ones available in the HP-15C. FP can find the coefficients of a polynomial of degree N which exactly fits a given set of N+1 arbitrary data points (not necessarily *equally spaced*), where N is limited only by available memory.

Note: The equivalent program for the HP-15C can be found elsewhere in this issue, though for completeness' sake, I'm including here a brief description of some mathematical details.

Among the many functions we could fit to data, polynomials are by far the easiest to evaluate and manipulate numerically or symbolically, so our problem is:

Given a set of **n+1** data points (x_1, y_1), …, (x_{n+1}, y_{n+1}), find an Nth-degree polynomial

$$y = P(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + \ldots + a_{n+1} x^n$$

which includes the (n+1) data points (x_1, y_1), (x_2, y_2), ..., (x_{n+1}, y_{n+1}). The coefficients $(a_1, ..., a_{n+1})$ can be determined solving a system of (n+1) equations:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & . & x_1^n \\ 1 & x_2 & x_2^2 & . & x_2^n \\ . & . & . & . & . \\ 1 & x_{n+1} & x_{n+1}^2 & . & x_{n+1}^n \end{bmatrix} \begin{bmatrix} a_1 \\ . \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_{n+1} \end{bmatrix}$$

FP *requires* either an HP-41CX with the Advantage ROM plugged in, or else an HP-41C/CV with the Advantage ROM *and* the X-Functions ROM. As the Advantage ROM can directly operate with matrices residing in EM, *no* adressable registers in main RAM are used at all, which means this program can run even at SIZE 000 !

Program listing

```
01  LBL "FP"    to use, simply XEQ "FP"
02  "N=?"       prompts for the degree N of the polynomial
03  PROMPT      .. and waits for the user to enter N
04    1         add 1 to get the number of data points
05    +         .. N+1
06  1.001       the required multiplier
07    *         forms the matrix dimensions [N+1].00[N+1]
08  "MX"        specifies matrix MX to be created in X-MEM
09  MATDIM      creates and dimensions matrix MX in X-MEM
10    0         specifies first row, first column and ..
11  MSIJ        .. resets the row/column indexes
12  LBL 00      loop to ask for data & compute MX elements
13  MRIJ        recalls the current value of the indexes
14  "X"         forms the prompt to ask the user to enter x_i
15  AIP         .. appends the index to the prompt
16  "|-=?"      .. appends "=?" to the prompt
17  PROMPT      prompts to enter x_i and resume execution
18  ENTER       fills the stack with the value of x_i ..
19  ENTER       .. in order to compute all powers of x_i ..
20  ENTER       .. from 1 to x_i^N, and store them in MX
21    1         initializes the value of x_i^0 [i.e.: 1]
22  MSR+        stores it in MX and updates the indexes
23  LBL 01      loop to compute the powers of x_i
24    *         computes x_i^j
25  MSR+        stores it in MX and updates the indexes
```

```
26  FC?09       are we done with this row ?
27  GTO 01      not yet, go back for the next x_i power
28  FC? 10      row done. Are we done with all rows ?
29  GTO 00      not yet, go back to ask for the next x_i
30  CLA         all rows done, MX complete. Make it current
31  DIM?        get its dimensions: [N+1].00[N+1]
32  INT         get N+1 (avoid using a register)
33  "MY"        specify vector MY to be created in X-MEM
34  MATDIM      creates and dimensions vector MY in X-MEM
35  LBL B       ask for y_i data and store them in MY
36    0         specifies 1st element of the vector and ...
37  MSIJ        .. resets the index to the 1st element
38  LBL 02      loop for next data and store them in MY
39  MRIJ        recalls the current value of the index
40  "Y"         forms the prompt to ask for y_i
41  AIP         .. appends the index to the prompt
42  "|-=?"      .. appends "=?" to the prompt
43  PROMPT      prompts the user to enter y_i
44  MSR+        stores it in MY and updates the index
45  FC? 10      are we done with all elements ?
46  GTO 02      not yet, go back to ask for the next y_i
47  "MX,MY"     all y_i stored. Specify MX,MY for the system
48  MSYS        solves the system for the coefficients
49  LBL C       retrieve and display each coeff.
50    0         specifies 1st element of the coeffs. vector
51  MSIJ        resets the index to the 1st coefficient
52  LBL 03      loop to retrieve the next coefficient
53  MRIJ        recalls the current value of the index
54  "A"         forms the prompt to display each coeff.
55  AIP         .. appends the index to the prompt
56  "|-="       .. appends "=" to the prompt
57  MRR+        retrieves the value of the current coeff.
58  ARCL X      .. appends the value to the prompt
59  PROMPT      shows the value to the user
60  FC? 10      are we done outputting all the coeffs. ?
61  GTO 03      not yet, go back for the next coefficient
62  END         all done. End of execution.
```

Notes

- As the Advantage ROM can work with matrices directly in EM, **FP** doesn't use any main RAM registers and so it will run *even at SIZE 000*. This has the added

advantage (pun intended) of avoiding any register *conflicts* with other programs.

- FP creates two matrices in X-MEM, namely MX and MY, which aren't destroyed upon termination. Retaining MX allows the user to compute the coefficients of another polynomial using the same x data but *different* y data. In that case, the x data need not be entered again, only the new y data must be entered. Further, as the MX matrix is left in *LU-decomposed form* after the first fit, the second fit will proceed *much faster*. Retaining MY allows the user to employ the polynomial for interpolating purposes, root finding, numeric integration or differentiation, etc.
- Lines 2-11 prompt the user for the degree of the polynomial, then allocate the system matrix in Extended Memory (**MATDIM**) and reset the indexes (**MSIJ**).
- Lines 12-22 set up a loop which will fill up the rows of MX. Notice the use of the miscellaneous function **AIP** to build the prompt, and **MSR+** to store the value *and* automatically advance the indexes to point to the next element.
- Lines 23-27 form a tight loop which computes each power of x_i and uses **MSR+** to store it and advance the indexes. Flag 9 logs if we're done with the column in which case we would proceed to the next row. If so, Flag 10 is then checked to see if we're done with all the rows. In the HP-15C, an "user" **STO** can do the whole job of storing elements, updating indexes, and checking for termination.
- Once the system matrix has been populated, lines 30-45 do likewise dimension, and populate the MY matrix, prompting the user for the required y_i values. Then, once all the data have been input and both matrices are allocated and populated, lines 46-47 solve the system for the coefficients of the polynomial (by using **MSYS**. The HP-15C version used the 'overloaded' division key [÷] instead).
- Finally, lines 48-59 stablish a loop which labels and outputs all the coefficients.

Usage instructions

1. After keying in the program, you can execute it by pressing:

   XEQ "FP" → N=?

2. Enter the degree of the polynomial you want to fit:

   N, **R/S** → X1=?

3. You must enter *all the x values first,* then *the y values*:

   x_1, **R/S** → X2=?
   x_2, **R/S** → X3=?

...
x_{n+1}, **R/S** → Y1=?

4. Now the program is prompting you to enter the y values

y_1, **R/S** → Y2=?
y_2, **R/S** → Y3=?
...
y_{n+1}, **R/S** → A1=(a_1), **R/S** → A2=(a_2), ... , **R/S** → A(n+1)=(a_{n+1})

5. If you want to display all the coefficients again, set ***User*** mode and press:

C → A1=(a_1), **R/S** → A2=(a_2), ... , **R/S** → A(n+1)=(a_{n+1})

6. If you want to compute the coefficients for *another* set of data with the *same* x_i values but *different* y_i values, you don't need to reenter the x values. Press:

B → Y1=?

and go to (4) above to enter the new y_i values. You'll notice that it takes *much less time* to compute the coefficients now, as the matrix is already *in LU-decomposed* form, and thus can be used as is, saving a lot of processing time.

Example

Rumour has it that the seemingly trigonometric function ***y = cos(5 arccos x)*** *is actually a 5th-degree polynomial in disguise. Attempt to retrieve its true form.*

If it is indeed a 5^{th}-degree polynomial, we can retrieve its true form by fitting a 5^{th}-degree polynomial to a set of 6 *arbitrary* data points (x,y). Any set with different x values (-1.0 <= x <= +1.0) will do, but for simplicity's sake we'll use x=0, 0.2, 0.4, 0.6, 0.8, and 1. Proceed like this:

- set Radians mode, 4 decimals:	**XEQ "RAD", FIX 4**
- start the program:	**XEQ "FP"** → N=?
- specify degree 5:	**5 R/S** → X1=?
- enter 1^{st} x value:	**0 R/S** → X2=?
- enter 2^{nd} x value:	**0.2 R/S** → X3=?
- enter 3^{rd} x value:	**0.4 R/S** → X4=?
- enter 4^{th} x value:	**0.6 R/S** → X5=?
- enter 5^{th} x value:	**0.8 R/S** → X6=?

- enter 6th x value: 1 **R/S** → Y1=?
- enter 1st y value: **0, COS⁻¹, 5, *, COS, R/S** → Y2=?
- enter 2nd y value: **0.2, COS⁻¹, 5, *, COS, R/S** → Y3=?
- enter 3rd y value: **0.4, COS⁻¹, 5, *, COS, R/S** → Y4=?
- enter 4th y value: **0.6, COS⁻¹, 5, *, COS, R/S** → Y5=?
- enter 5th y value: **0.8, COS⁻¹, 5, *, COS, R/S** → Y6=?
- enter 6th y value: 1, **COS⁻¹, 5, *, COS, R/S** → A1=-1.0250E-9
  R/S → A2=5.0000
  R/S → A3=7.0867E-8
  R/S → A4=-20.0000
  R/S → A5=2.6188E-7
  R/S → A6=16.0000

So, disregarding the very small coefficients due to rounding errors, the undisguised polynomial is:

$$\mathbf{P(x) = \cos(5 \arccos x) = 5x - 20x^3 + 16x^5}$$

If you've got an HP-41CX, you might want to execute now **EMDIR**, to see that the matrices used are still available so that you can *redisplay* the coefficients, solve for a *new* set of ***y*** values, or use the polynomial for interpolation, etc.

XEQ "EMDIR" → MX @036 [*the system matrix is 6x6 = 36 elements*]
→ MY @006 [*the y/coeff. matrix is 6x1 = 6 elementss*]
→ 554.0000 [*this value varies with your configuration*]

Final remarks

I hope this article has given you some insight on the amazingly powerful functionality of the Advantage ROM and the ways in which it can extend the capabilities of the 41C. It's a real pity that it was introduced a little late in the product life of the 41C, one can just wonder its *impact* should it have been introduced shortly after the 41C was released. For one thing, the **PPC ROM** itself would have been *drastically* different, as much of its code is devoted to perform as efficiently as possible many tasks that the Advantage ROM deals with much better. Its effect on engineers and other professionals, as well as students, would have been tremendous. Come to think of it, in a way it was: a whole series of technical books (**Grapevine**'s "**Using your Advantage ROM**" series), were successfully launched, including titles such as "*Statics for Students*", "*Electrical Circuits for Students*", and "*Computer Science*", among others.

My final advice: if you've got an HP-41, any model, you absolutely **MUST** have an Advantage ROM. You'll never, *ever*, look at 15C/42S users with "envious eyes" (or to any other model for that matter). It'll increase your 41 programming pleasure **tenfold**.

Note: For an HP-15C implementation, see my article "HP-15C Nth-degree Polynomial Fitting" elsewhere in this issue, featuring 3 additional examples you can try out with this program as well !