HP-71B Math ROM Baker's Dozen (Vol. 1)

Valentín Albillo (#1075, PPC #4747)

Among the many worthwhile plug-in ROMs ever made for the HP-71B[1], the Math ROM is one of the very best. Many of its principal functions and capabilities were intended to be part of the System ROMs from the very start, but were sadly left out when it became clear that the total code was going to exceed the allotted 64 Kb of System ROM space. So, it was excised from internal ROMs #0 and #1 and placed instead in external ROM #2, all 32 Kb of highly optimized machine-language code. Yet some leftovers still remain inside the System ROMs, now mostly issuing polls to the Math ROM code, if present, for it to handle a number of operations such as those involving complex numbers (which the mainframe *does* accept but *can't* deal with), and IMAGE statements involving complex numbers.

All in all, the Math ROM is the one *essential* addition to any HP-71B system, extending the already very powerful BASIC programming environment to its fullest for all things mathematical. As one of its original creators (Steve Abell, father of CALC mode) once posted when talking about the HP-71B system: *"It had math and stat software to *die* for."*. Indeed. And all later powerful HP models (such as the 42S and the 48/49 series) have inherited and expanded the advanced functionality of this incredible achievement in portable math computing, itself made possible by the previous experience acquired during the development of the HP-85 Matrix ROM, the HP-75 Math ROM, and the HP-15C highly-optimized **SOLVE** and **INTEGRATE** functionality (originally developed for the HP-34C !). The best algorithms were extracted and rewritten in Saturn assembly code, including a large number of refinements (just compare complex number handling) and additions (such as maintaining IEEE compliance, the one and only handheld product at the time to offer such).

This article (subdivided into two reasonably-sized "volumes" to make life a little easier for our kind Editor and readers) isn't yet another review of the Math ROM capabilities, that has been done a number of times before and to do it justice would probably require a whole issue of Datafile or two. Instead, this article features **13 assorted mini-topics**, each of them discussing some interesting or otherwise novel aspect of choice Math ROM capabilities, from complex numbers to matrices, root finding, numerical integration, and special functions. I hope you'll find at least some of them enjoyable, even intriging, and further, useful to increase your awareness of the incredible portable computing power the Math ROM has to offer even today.

---

[1] If you have no HP-71B or no Math ROM, just get Emu71, a free emulator for Windows by Jean-François Garnier, or HP-71X, a 48/49 based-one by Hrastprogrammer. Both are fantastic !

1. It's the norm !

RNORM and **CNORM** are two matrix functions which return the **Row Norm** (respectively, **Column Norm**) of a given rectangular matrix or vector, this is, the largest sum of the absolute values of the elements of each row (respectively, column). An example will make it clear: assume we're given this 2x3 matrix,:

```
|   -2    3   -5   |
|    4   -4    6   |
```

then the sums of the absolute values of the elements of each row are, respectively, 10 and 14, so the Row Norm will be **14**. Likewise, the Column Norm is **11**. However, beyond its intended purpose, these two functions can be used to help compute some other useful matrix functions not directly provided. For instance, consider this program code[2]:

```
DESTROY ALL @ OPTION BASE 1
R=2 @ C=3 @ DIM A(R,C) @ READ A(,)
DATA -2,3,-5,4,-4,6
DIM A(R*C) @ DISP CNORM(A), RNORM(A) @ DIM A(R,C)
```

Upon running it, two values will be computed and displayed: **24** and **6**. Assuming **OPTION BASE 1**, when matrix **A,** with **R** rows and **C** columns, is *temporarily redimensioned* to a column vector with R*C elements, **A**(R*C), then:

- **CNORM** gives the sum of the absolute values of ***all*** elements of **A**: ***24***
- **RNORM** returns the absolute value of the ***maximum*** element of **A**: ***6***

and you can then redimension matrix **A** back to its original dimensions. But what about getting the sum of *all* elements, taken with their proper signs, not just absolute values ?

Well, a pair of nested **FOR-NEXT** loops would do (or even just one if the matrix is previously redimensioned to a 1-dimensional column vector), but fortunately there's a *direct*, non-looping way to do it with the help of **RNORM** and **CNORM**, which is much faster than looping through all elements, specially for large matrices. The following subprogram **MSUM** accepts any two-dimensional matrix, and returns the *sum of all its elements, taken with their signs*. Notice there are *no* loops or branching, it's a *direct* computation:

```
SUB MSUM(T(,),S) @ F=UBND(T,1) @ C=UBND(T,2) @ P=LBND(T,1)
N=(F-P+1)*(C-P+1) @ OPTION BASE P @ DIM A(F,C),B(N-NOT P)
MAT A=T @ DIM A(N-NOT P) @ M=RNORM(A) @ MAT B=(M)
MAT A=A+B @ S=CNORM(A)-N*M @ END SUB
```

[2] In all program listings here, line numbers are present only if referenced; you may use whatever line numbering suits you as long as it's sequential from lower to higher line numbers.

This subprogram works only for real (not complex) matrices (not vectors, see below for a real vector version), and will accept *any* matrix in *any* **OPTION BASE** thanks to Math ROM functions **UBND** (Upper Bound) and **LBND** (Lower Bound) which find out the dimensions and **OPTION BASE**, thus eliminating the need to pass them as parameters. **MSUM** doesn't alter matrix **A** but temporarily uses two auxiliary matrices the size of **A**. As for speed, check these times (in seconds):

NxN	MSUM	LOOP VERSION
10x10	1.56	3.01
15x15	2.97	4.91
20x20	4.95	8.96

so **MSUM** is nearly two times faster then the fastest looping version. As a bonus, if you pass an additional parameter **H**, and also include the statement

```
H=RNORM(A)-M
```

after **MAT A=A+B** above, then **H** will return with the value of the *maximum positive element* of **A**. Further previous manipulations with **A,** such as changing the sign of all its elements by issuing a **MAT A=-A** statement, would get you the *maximum negative element* of **A**, etc. Here's the 1-dimensional vector version:

```
SUB VSUM(A(),S) @ P=LBND(A,1) @ N=UBND(A,1)-P+1
OPTION BASE P @ DIM B(N-NOT P) @ M=RNORM(A) @ MAT B=(M)
MAT A=A+B @ S=CNORM(A)-N*M @ MAT A=A-B @ END SUB
```

this works only for real vectors but needs just *one* auxiliary matrix instead of two, so it can deal with larger vectors for any RAM size. The caveat is, it needs an additional **MAT A=A-B** statement to restore **A**, which slows it down a bit and might introduce some (extremely small) rounding errors. It can also be used with matrices by simply redimensioning them to vectors and back, like this (**A** is originally a 20x20 matrix):

```
DIM A(20*20) @ CALL VSUM(A,S) @ DIM A(20,20)
```

which runs in 6.05 seconds, still faster than looping. By the way, there's yet another norm, the **Frobenius Norm**, **FNORM**, which also has its special uses. For instance, this code asks for any number of data and *directly* computes (*no* looping, *very* fast) the sum of their squared values:

```
DESTROY ALL @ OPTION BASE 1 @ FIX 4 @ INPUT "# data ? ";N
DIM P(N) @ MAT INPUT P @ DISP "Sum. squares = ";FNORM(P)^2
```

```
>RUN
     # data ? 20 [ENTER]
     P(1)? 5,4,-3,2,8,6,6,7,2,-4,8,-7,4,2,-6,2,3,-10,1,23 [ENT]
Sum. squares =  1071.0000
```

2. 'GIMME' GAMMA !

New times beget new applications, and new applications demand expanding the standard arsenal of transcendental functions to include new ones (or old as the case may be) which suddenly enjoy the limelights. But no calculator does include those functions right from the keyboard, and even programmable ones usually require a somewhat *lengthy* and *complicated* program to compute them.

Not so with the HP-71B+Math ROM ! The incredibly extensive and powerful instruction set provided by this ROM, with hyperbolics, **GAMMA**, complex-valued operations, matrices, and above all, recursive **FNROOT** and **INTEGRAL**, makes it extremely easy to implement in a few lines of code most any special function required in modern applications. Add to that the fact that user-defined functions can be multi-line, recursive, *and callable right from the keyboard*, and you'll find yourself with an "über-calculator" that seamlessly incorporates every required special function as if they were standard, built-in fare.

Let's give some hot examples. Recently, the old-but-little-known transcendental **Mittag-Leffler function**, a natural generalization of the exponential defined as:

$$E_\alpha(x) := \sum_{n=0}^{\infty} \frac{z^n}{\Gamma(\alpha n + 1)},$$

has proven to be essential for a number of problems in applications, due to its strong connections with *fractional calculus.* A simple implementation might be:

```
1 DEF FNE(A,X) @ S=1 @ N=1
2 T=X^N/GAMMA(A*N+1) @ IF S+T<>S THEN S=S+T @ N=N+1 @ GOTO 2
3 FNE=S @ END DEF
```

and now we're ready to merrily use it as desired. For instance, let's check that:

```
E(1,X)      → EXP(X)      E(2,X^2) → COSH(X)
E(2,-X^2)   → COS(X)      E(1/2,X) → EXP(X^2)*(1+ERF(X))
```

by trying some numeric cases from the keyboard. **ERF** stands for **Erf**, the **Error Function**, which we need previously define with this one-liner (!):

```
4 DEF FNF(X)=2/SQR(PI)*INTEGRAL(0,X,1E-12,EXP(-IVAR*IVAR))
```

Now :

```
>FNE(1,PI), EXP(PI)
     23.1406926328        23.1406926328
>FNE(2,2^2), COSH(2)
     3.76219569109        3.76219569108
>FNE(2,-2^2), COS(2)
     -.416146836548       -.416146836547
>FNE(1/2,SQR(2)), EXP(2)*(1+FNF(SQR(2)))
     14.4419081951        14.4419081954
```

and you may see the extremely close agreement with the theoretical values.

Another interesting "recently discovered" function is the **Lambert W-Function**, which is the functional inverse of:

$$y = x.e^{x}$$

This function keeps reappearing in a vast number of theoretical and practical applications, to the point where it has been semi-informally proposed as *the next elementary function*, joining the ranks of trigonometric, hyperbolic, exponential and logarithmic functions. If that were so, solutions given in terms of **W** would be considered as *closed-form solutions*, just as they would if given in terms of sines or exponentials. The derivative and the indefinite integral of **W** can both be expressed in terms of **W**, as can the solution to otherwise unsolvable ODE's and also the roots of whole classes of basic transcendental equations. For instance:

$$x^{x} = a \quad \rightarrow \quad x = e^{W(\ln(a))}$$

$$e^{x} + a*x + b = 0 \quad \rightarrow \quad x = -(a.W(e^{-b/a}/a) + b)/a$$

Let's check these out with some numerical examples. First we need a user-defined function to compute Lambert's W function. This one-liner (again!) will do:

```
5 DEF FNW(X) = FNROOT(0,10,FVAR*EXP(FVAR)-X)
```

Notice how the powerful **FNROOT** feature of the Math ROM (as **INTEGRAL** before) makes things extremely easy for us. Now for some numerical cases:

$$x^{x} = 5 \quad \rightarrow \quad x = e^{W(\ln(5))}$$

Just enter, right from the keyboard:

```
>EXP(FNW(LN(5))), RES^RES [ENTER]
      2.12937248276          5
```

which indeed is a root of the transcendental equation. Now let's solve:

$$e^{x} + 3*x - 5 = 0 \quad \rightarrow \quad x = -(3.W(e^{5/3}/3) - 5)/3$$

Again, from the keyboard:

```
>-(3*FNW(EXP(5/3)/3)-5)/3, EXP(RES)+3*RES-5
      .870573381313          -.00000000001
```

which confirms the value found as a root for that transcendental equation. Finally, though **HP** didn't know at the time, one of their classical, often-used examples (see for instance the ***HP-67 Math Pak 1 Owner's Handbook***, page 08-02, Example 2) can be solved in closed form using **W**(x), like this:

$$\ln(x) + 3*x - 10.8074 = 0 \quad \rightarrow \quad x = W(3e^{10.8074})/3 = \underline{3.21336087018}$$

3. COMPLEX rants

- The complex instruction set is *far* from *complete*. That special functions such as **GAMMA** aren't defined for complex arguments is to be expected (if regrettable), but the fact that inverse trigonometric (let alone inverse hyperbolic) functions **ASN** (arc sine), **ACOS** (arc cosine), and **ATAN** (arc tangent) aren't defined either for complex **Z** is really inexcusable: you can find the sine of a complex value, yet you *can't* compute the inverse, lest you get:

  ERR: WORD/XFN Not found

  LGT (base-10 logarithm) gives the same message (though both **LN** and **LOG** work fine), but inverse hyperbolic functions (and **GAMMA**) all result in **ERR: Data Type** instead. Even the HP-15C knew better than this ! (also the **DET** determinant matrix function doesn't work for complex matrices). This being so, these user-defined one-liners might do the job for suitable complex Z:

```
DEF FNS1(Z)=(0,-1)*LN((0,1)*Z+SQR(1-Z*Z))          !asn
DEF FNC1(Z)=PI/2-FNS1(Z)                           !acos
DEF FNT1(Z)=(0,1/2)*(LN(1-(0,1)*Z)-LN(1+(0,1)*Z)) !atan
DEF FNS2(Z)=LN(Z+SQR(1+Z*Z))                       !asnh
DEF FNC2(Z)=LN(Z+SQR(Z+1)*SQR(Z-1))                !acosh
DEF FNT2(Z)=(LN(1+Z)-LN(1-Z))/2                    !atanh
```

- User-defined functions can take complex arguments and return complex results. You need do nothing special, this is default behaviour. For instance:

  DEF FNF(Z)=SIN(Z)+COS(Z)

  will return a complex value if called with a complex **Z** [say, **FNF((2,3))**] and a real value if called with a real **Z**. Same goes for subprograms, you can pass and retrieve complex values from them. However, the functional result and the function parameters *must* be of the same type, real or complex. If you define:

  DEF FNF(Z)=(1,1)*Z

  you can then call it with a complex parameter **Z**, say **FNF((1,1))** and get **(0,2)** as a result. But calling it with a real parameter, say **FNF(1)**, will result in a **ERR: Data Type** message instead. Here, **FNF((1,0))** would work.

- The HP-71B system *does* know about complex values but doesn't include any functions to define or deal with complex variables, that's left for external ROMs (i.e.: the Math ROM) or binary files. You can check this easily, just enter this program line with *no* Math ROM plugged in:

  10 A=(2,3)+SIN((2,3))

  and see the mainframe accept it happily, no **Syntax error** whatsoever. However, if you then try to run it, you'll get the infamous **ERR: XWORD/XFN Not found** message (unless you've got the Math ROM plugged in, that is).

4. Missing Matrix Operations

Though the HP-71B's Math ROM matrix operations were heavily based in the HP-85 Matrix ROM set, a number of functions and capabilities were nevertheless *omitted*, most probably due to space and budget restrictions. However, we can try and implement the missing functions with the help of the available ones. For instance, consider implementing the HP-85 Matrix ROM operation:

```
MAT A = (K1)*B + (K2)*C
```

The obvious method would make use of an auxiliary matrix, like this:

```
MAT A=(K1)*B @ MAT D=(K2)*C @ MAT A=A+D
```

But provided **K1** (or **K2**) is *not* zero (which would be a trivial case anyway), a better method is:

```
MAT A=(K2/K1)*C @ MAT A=A+B @ MAT A=(K1)*A
```

which avoids using an auxiliary matrix altogether and runs as fast. The only minor inconvenience is that it may introduce some (small) rounding errors and it shouldn't be applied to **INTEGER** matrices, unless **K2** happens to be exactly divisible by **K1** (or vice versa).

The HP-85 Matrix ROM also allows all four arithmetic functions between a matrix and a scalar, as well as element-by-element matrix multiplication and division. These missing operations can be implemented like this:

```
MAT A=A + (X)  →   MAT B = (X) @ MAT A = A + B
MAT A=A - (X)  →   MAT B = (X) @ MAT A = A - B
MAT A=A * (X)  →   MAT A = (X) * A
MAT A=A / (X)  →   MAT A = (1/X) * A
MAT A=A . B    →   DIM A(N*N),B(N*N)
                   FOR I=1 TO N*N @ A(I)=A(I)*B(I) @ NEXT I
                   DIM A(N,N),B(N,N)
MAT A=A / B    →   same but instead A(I)=A(I)/B(I)
```

In the last two cases, redimensioning the matrices to one-dimensional vectors and back allows us to use a single **FOR-NEXT** loop and simpler indexing, which helps make the computation run appreciably faster.

Other missing matrix operations include assignment and copy of submatrices and ranges of rows/columns, as well as the possibility to manipulate *empty* arrays. In most practical cases, those operations can be implemented with relative ease, on a case-by-case basis.

5. FiNdROOT doesn't !

The manual makes it clear. **FNROOT** (Find Root) is a powerful *funny* function that quickly and accurately finds a real root of any given function in a given interval or, if unsuccessful, reports after a usually short while that none could be found.

By design, **FNROOT** should *always* terminate, no matter what. It must *never* get stuck on a loop, indefinitely searching for an elusive or nonexistent root. We want results and we want them as quick as possible, even if they turn out to be a report that no root could be found. Spending a reasonable time searching for a nonexistent root to make sure it can't be found is Ok. Spending an inordinate amount of time, or even forever, is not acceptable and should never happen.

But it does ! And, contrary to what you might expect, it doesn't take a particularly complicated function or a wide search interval. Matter of fact, say you try to find a root of the extremely simple quadratic equation **X*X+1 = 0**, supplying **X=0** as the (only) initial value. You would then execute this statement from the command line:

```
FNROOT(0, 0, FVAR*FVAR+1)
```

and, if like me, you happen to own a Math ROM Version 1A (as reported by executing **VER$**), you'll find out that it does *not* terminate ! At least not in any reasonable amount of time. This clearly qualifies as a serious bug in **FNROOT** !

You might want to investigate it further. If so, simply use this program, which defines the function **X*X+1** not directly as the 3rd parameter of **FNROOT**, but rather as a multi-statement user-defined function **FNF**. This exhibits the bug as well, and allows us to include a counter of the number of times **FNF** is called by **FNROOT**, and also to inspect both the value of the counter and the current value of the guess, **X**, upon temporarily pausing the execution by pressing the [**ATTN]** key:

```
10 DESTROY ALL @ N=0 @ X=FNROOT(0,0,FNF(FVAR)) @ DISP X
20 DEF FNF(X) @ N=N+1 @ FNF=X*X+1 @ END DEF
```

Run this for as long as you care, it *won't* end. After a *very* long run, I stopped the execution by pressing the [**ATTN]** key and inspected the values:

```
>N, X
  1208278     1.00001208231E-6
```

so you see, despite *more than 1,200,000 evaluations* of our function, **FNROOT** has neither found a root nor yet decided there's none, and the current guess, 0.000001000012+ is still *very* close to our initial guess, 0, so we're hardly making any substantial progress. Seems that at least *several billion* (US variety) evaluations would still be needed to reach a decision, if at all.

6. COMPLEX means ...

The Math ROM includes a number of utility-like functions some of which should have stayed in the System ROMs to begin with, such as **LBND** (Lower Bound), **UBND** (Upper Bound), and **TYPE**. These functions are very useful to help write *generalized subprograms*, which can deal with multiple data types, dimensions, and even **OPTION BASE** settings, without the calling program having to pass extra parameters or set flags in advance to specify the various possibilities.

For example, consider this subprogram, **VMEAN**, which will accept a vector passed to it and will return the *arithmetic mean* of all its elements in another parameter, which must be an scalar variable of the same type (real or complex) as the vector, passed by reference:

```
SUB VMEAN(V(),M) @ IF TYPE(V)<6 THEN REAL S ELSE COMPLEX S
FOR I=LBND(V,1) TO UBND(V,1) @ S=S+V(I) @ NEXT I
M=S/(UBND(V,1)-LBND(V,1)+1) @ END SUB
```

Notice that, thanks to making use of **LBND**, **UBND**, and **TYPE**:

- it works for *both* **REAL** and **COMPLEX** vectors (we use **TYPE** to discern whether **V** is real/complex and dimension auxiliary variable **S** to be the same type).
- if works for vectors of any size, created with any **OPTION BASE** (we use **LBND** and **UBND** to ascertain the lower and upper limits for the index and thus the total number of elements in the vector).
- you can also use it for matrices, simply redimension them temporarily to vectors, call **VMEAN**, and redimension them back to their original dimensions, like this:

```
OPTION BASE 1 @ DIM V(10,20),M @ COMPLEX W(15,17),N
...
DIM V(10*20) @ CALL VMEAN(V,M) @ DIM V(10,20)
COMPLEX W(15*17) @ CALL VMEAN(W,N) @ COMPLEX W(15,17)
```

- it can easily be extended to return both arithmetic (**M**) and geometric (**G**) means in the same call. This would be the extended version:

```
SUB VM(V(),M,G) @ IF TYPE(V)<6 THEN REAL S,P ELSE COMPLEX S,P
P=1 @ FOR I=LBND(V,1) TO UBND(V,1) @ S=S+V(I) @ P=P*V(I)
NEXT I @ T=UBND(V,1)-LBND(V,1)+1 @ M=S/T @ G=P^(1/T)
```

7. ... And COMPLEX ways

The extra capabilities provided by the Math ROM's extensive additions to the built-in function set brings us new choices as to the way to implement desired functionalities. Suppose you need to compute the very common expression:

$$X^2 + Y^2$$

the natural way to do it would be, of course:

```
X*X+Y*Y
```

which performs *two* multiplications and *one* addition. But that's not the only way, you can also compute it using complex operations and variables, like this:

```
REPT((X,Y)*(X,-Y))
```

This seems to require just *one* multiplication, plus ancillary extra operations. But performing the complex multiplication needs a larger number of operations with the real-valued components internally and thus the timing is actually *worse*, so no gain in this case. However, consider instead the square root of the original expression: this would be computed quite straightforwardly like this:

```
SQR(X*X+Y*Y)
```

But you can also make use of complex operations, namely the simpler expression:

```
ABS((X,Y))
```

And though, again, the former is some 14% faster, this time there's a gain to be made in terms of memory required: the complex-domain expression takes just 10 bytes, versus 14 bytes for the all-real one. This won't be a decisive factor in most applications, but it's good to know the Math ROM allows for alternate ways. Finally, consider the slightly more cumbersome expression:

$$\frac{X}{\sqrt{X^2 + Y^2}}$$

You'll be tempted to compute it straightforwardly, like this:

```
X/SQR(X*X+Y*Y)
```

However, making use of Math ROM functions provides this nice alternate way:

```
REPT(SGN((X,Y)))
```

which not only looks smart, but it's slightly faster as well, and even better, completely avoids the show-stopping **ERR: 0/0** *error* message which is displayed unmercifully if **X** and **Y** both happen to be 0. The complex-domain version returns the correct value instead, with no division by zero in sight. Anyway, if nothing else, I hope you'll agree it's exciting to look for alternate ways using Math ROM functions and capabilities, and sometimes it even pays !

Next issue

The 2nd part of this article will feature the remaining sections of this **Baker's Dozen** of HP-71B Math ROM's goodies. Hope you enjoyed the ride so far !