HP-71B Math ROM Baker's Dozen (Vol. 2)

Valentín Albillo (#1075, PPC #4747)

Welcome to the second part of this Baker's Dozen compendium of **assorted mini-topics** discussing some interesting uses of the HP-71B's Math ROM[1]. The six remaining topics discussed here deal with *advanced applications* such as polynomial data fitting (real or complex data), multidimensional integration, solving systems of non-linear equations, and finding all eigenvalues of arbitrary matrices. As in **Volume 1**, it's the purpose of this second part to show just how *easily* these usually complicated tasks can be dealt with, almost in a casual manner, thanks to the incredibly powerful Math ROM features.

8. Tight fitting

Many real-life tasks require fitting a polynomial curve to a given set of data. The coefficients of the n^{th} degree polynomial:

$$P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

which includes **(N+1)** given data points (x_0,y_0), (x_1,y_1), ..., (x_n,y_n) can be found very easily using **MAT … SYS**. Here's a 4-liner that asks for the degree **N** of the polynomial and the **(N+1)** data points, then outputs the coefficients a_0, a_1 , ... , a_n of the polynomial[2]:

```
DESTROY ALL @ OPTION BASE 0 @ INPUT "N=";N
DIM X(N),Y(N),A(N),M(N,N) @ MAT INPUT X,Y
FOR I=0 TO N @ FOR J=0 TO N @ M(I,J)=X(I)^J @ NEXT J
NEXT I @ MAT A=SYS(M,Y) @ MAT DISP A
```

However, it would be more useful if rewritten as a *subprogram*, callable from anywhere:

```
SUB IPOL(X(),Y(),A()) @ N=UBND(X,1) @ DIM M(N,N)
MAT M=CON @ FOR I=0 TO N @ FOR J=1 TO N @ M(I,J)=X(I)^J
NEXT J @ NEXT I @ MAT A=SYS(M,Y) @ END SUB
```

Notice the use of **UBND** to find out the number of data points passed to the subprogram, and **MAT … CON** to save one iteration in the **J** loop.

An example of a separate 'tester' program calling this subprogram goes like this:

---

[1] Having no HP-71B and/or no Math ROM doesn't mean you can't try and enjoy this article, just search the web for Emu71 (a free emulator for Windows) or HP-71X (a 48/49 based-one).

[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.

```
DESTROY ALL @ OPTION BASE 0 @ INPUT "N=";N
DIM X(N),Y(N),A(N) @ MAT INPUT X,Y
CALL IPOL(X,Y,A) @ MAT DISP A
```

For instance, let's fit a 2nd degree polynomial to data points (2, 1), (5, 70), (3, 16):

```
>RUN                [ENTER]
N=2                 [ENTER]
X(0)? 2,5,3         [ENTER]
Y(0)? 1,70,16       [ENTER]
     -5   -5    4
```

so the fitting polynomial is: **$P(x) = -5 - 5*x + 4*x^2$**

Notice that the data points need *not* be equally spaced, and further they need *not* be entered in any particular order.

9. What about fitting complex data ?

The **IPOL** subprogram featured above can be generalized to fit a polynomial with *complex* coefficients to any given set of *complex data points*. To that effect, very few changes are needed to extend operations into the complex domain. Let's see:

The subprogram header can *automatically* accept complex arrays as parameters and return a complex array as a result, so no change needed here. **SYS** and arithmetic operations will happily work with complex arguments, too. So the only change **IPOL** needs is to create matrix **M** as *either* **REAL** or **COMPLEX**, depending on the *type* of the arrays passed to the subprogram. Thus, it suffices to replace in **IPOL** the original statement

```
DIM M(N,N)
```

by the conditional construct:

```
IF TYPE(X)=5 THEN DIM M(N,N) ELSE COMPLEX M(N,N)
```

and that's it, **IPOL** will now happily work for real *and* complex data alike. To test it with complex data, you need to change **DIM** in the tester program above to **COMPLEX** and now you can try and fit a polynomial to the 4 complex data points:

X	(1,1)	(-1,3)	(2,1)	(0,-2)
Y	(2,1)	(50,21)	(10,13)	(-12,19)

```
>RUN                                          [ENTER]
N=3                                           [ENTER]
X(0)? (1,1),(-1,3),(2,1),(0,-2)               [ENTER]
Y(0)? (2,1),(50,21),(10,13),(-12,19)          [ENTER]
 (-2,-3)  (3,-1)    (4.994083E-15,-3)   (2,1)
```

so the fitting polynomial is: **$P(x) = (-2, -3) + (3, -1)*x + (0, -3)*x^2 + (2, 1)*x^3$**

10. Multiple integrals made easy

There aren't that many examples in the ***Math ROM's Owner's Handbook*** for computing *multiple integrals* of functions in several variables and the few that are featured aren't general enough. But it's actually quite easy and systematic once you know how to do it.

Here's a *general example* particularized for triple integrals, but the essential details are the same for integrals of other multiplicities (up to 5 nested integrals which is the maximum nesting allowed by **`INTEGRAL`**). We aim to compute the value of this *general triple integral*, with given precision **P:**

$$I = \int_{a_1}^{a_2} \int_{b_1(x)}^{b_2(x)} \int_{c_1(x,y)}^{c_2(x,y)} f(x,y,z).\ dz\ .\ dy.\ dx$$

where the limits of integration are, respectively:

- a_1 and a_2, which are *arbitrary constants*
- $b_1(x)$ and $b_2(x)$, which are *arbitrary functions of x* (including them being constant values)
- $c_1(x,y)$ and $c_2(x,y)$, which are *arbitrary functions of x and y* (including them being functions of only **x**, only **y** or even constant values).

Finally, **f(x,y,z)** is the function of **x**, **y**, and **z** (including being independent of some or all the variables) which we want to integrate between those limits. In order to obtain the value of the integral, just use this *template*, *replacing* the precision (**P**), the limits (a_1, a_2, b_1, b_2, c_1, c_2) and function being integrated (**f**) by your own functions and constants (the line numbers are completely *arbitrary*).

```
10 DEF FNF(X,Y,Z)=f(x,y,z)
20 DEF FNG(X,Y)=INTEGRAL(c1(x,y),c2(x,y),P,FNF(X,Y,IVAR))
30 DEF FNH(X)=INTEGRAL(b1(x), b2(x), P, FNG(X,IVAR))
40 I=INTEGRAL(a1, a2, P, FNH(IVAR))
```

Note: You could optimize it a little by inserting the definition for your f(x,y,z) right into the innermost **`INTEGRAL`** at line 20 and replacing all occurrences of **z** by **`IVAR`**. Thus line 10 would not be needed and the computation would proceed somewhat faster, but from a didactic point of view I nevertheless think that it's much more understandable as written.

For instance, let's compute this triple integral with precision **P** = 1E-3:

$$I = \int_0^2 \int_0^x \int_0^{x.y} x \,.\, y \,.\, z \,.\, dz \,.\, dy.\, dx$$

Making the appropriate substitutions into the template above, we have:

```
10 DEF FNF(X,Y,Z)=X*Y*Z
20 DEF FNG(X,Y)=INTEGRAL(0, X*Y, 1E-3, FNF(X,Y,IVAR))
30 DEF FNH(X)=INTEGRAL(0, X, 1E-3, FNG(X, IVAR))
40 I=INTEGRAL(0, 2, 1E-3, FNH(IVAR))
```

which, upon running, returns the desired value, **I = 4.0000**, correct to 5 digits.

11. And Systems of Simultaneous Non-linear equations too !

Thanks to the multiple levels of nesting allowed by **FNROOT** (Find Root), solving systems of up to 5 *non-linear* equations in five independent variables is a distinct possibility. However, as it was the case for multiple integration, the ***Math ROM's Owner's Handbook*** isn't flooding with examples. Actually, it's got *none at all*. You can find one example which shows how to find a solution pair **(x,y)** for a *single* equation in two variables, and another example which shows how to find the *minimum* of another very similar function in two variables, but that's it.

To overcome this bothersome lack of much needed examples, here you'll find a couple of *templates* you can use to solve a system on non-linear equations, particularized for simplicity to the case of two equations in two variables, but easily generalized to more variables if needed. First consider this template, where **f(x,y)** and **g(x,y)** are your two equations (i.e., **f(x,y)=0** and **g(x,y)=0**):

```
10 DEF FNF(X,Y)=f(x,y)
20 DEF FNG(X,Y)=g(x,y)
30 DEF FNH(X,Y)=ABS((FNF(X,Y),FNG(X,Y)))
40 DEF FND(X) @Y=FNROOT(Y,Y,FNH(X,FVAR)) @FND=FVALUE @ENDDEF
50 DESTROY ALL @ RADIANS @ FIX 6
60 INPUT "X,Y=";X,Y @ X=FNROOT(X,X,FND(FVAR))
70 DISP X;Y,FNF(X,Y);FNG(X,Y)
```

This attemps to *minimize the sum of the squares* of your **f(x,y)** and **g(x,y)**. Of course, the minimum possible sum (**0**) is achieved *when and only when* both

f(x,y) and **g(x,y)** are equal to **0** simultaneously, so finding a pair **(x,y)** which minimizes this sum is equivalent to solving the system. Let's use this template to solve the simple system:

$$x^2 + y^2 = 5$$
$$x^2 - y^2 = 3$$

First, enter the proper definitions for **f(x,y)** and **g(x,y)** into the template:

```
10 DEF FNF(X,Y)=X*X+Y*Y-5
20 DEF FNG(X,Y)=X*X-Y*Y-3
```

Now, using $x_0 = y_0 = 3$ as initial approximations:

```
>RUN       [ENTER]
X,Y=3,3    [ENTER]
 2.000000  1.000000   0.000000  0.000000
```

And, as you can see, the values **x=2** and **y=1** do make **f(x,y)** and **g(x,y)** equal 0 and are thus a valid root pair.

Another somewhat simpler approach, usually *faster* and with *improved convergence,* can be put to work by using this alternate template:

```
10 DEF FNF(X,Y)=f(x,y)
20 DEF FNG(X,Y)=g(x,y)
30 DEF FNY(X)=FNROOT(Y1,Y2,FNF(X,FVAR))
40 DESTROY ALL @ RADIANS @ FIX 6
50 INPUT "X1,X2,Y1,Y2=";X1,X2,Y1,Y2
60 X=FNROOT(X1,X2,FNG(FVAR,FNY(FVAR))) @ Y=FNY(X)
70 DISP X;Y,FNF(X,Y);FNG(X,Y)
```

Let's use this new template to solve the not-so-simple system (values in *radians*):

$$x = \sin(x + y)$$
$$y = \cos(x - y)$$

First, we enter the proper definitions for **f(x,y)** and **g(x,y)** into this template:

```
10 DEF FNF(X,Y)=X-SIN(X+Y)
20 DEF FNG(X,Y)=Y-COS(X-Y)
```

Now, using $x_1 = y_1 = 1$ and $x_2 = y_2 = 5$ as the initial search range:

```
>RUN                      [ENTER]
X1,X2,Y1,Y2=1,5,1,5       [ENTER]
 0.935082  0.998020       2.000000E-12  3.000000E-12
```

So, the values **x=0.935082** and **y=0.998020** are indeed the sought for root pair.

12. Synergic Eigenvalues

The need to compute the *eigenvalues* of arbitrary matrices frequently arises in many engineering applications, and so there's a number of methods available, mainly for special cases such as *symmetric* matrices where all eigenvalues are real. But the general case of arbitrary square matrices, where the eigenvalues can be *complex*, usually requires complicated algorithms and computing muscle.

Not so with the Math ROM. Its matrix functions combine *synergically* with its root finding capabilities to make computing eigenvalues extremely *easy* and *fast*. In particular, finding *just one* eigenvalue of any given square matrix is almost trivial, as this 4-liner demonstrates:

```
DESTROY ALL @ OPTION BASE 1 @ STD
INPUT "N=";N @ DIM A(N,N),B(N,N),U(N,N) @ MAT INPUT A
DISP "Eigenvalue = ";FNROOT(-10,10,FND(FVAR)) @ END
DEF FND(X) @ MAT U=IDN @ MAT U=(X)*U @ MAT B=A-U @ FND=DET(B)
```

It simply uses **FNROOT** to find *one* eigenvalue as a root of a user-defined function **FND** which uses matrix operations to construct the equation an eigenvalue must satisfy. Let's test it with a 5x5 matrix:

```
>RUN      [ENTER]
N=5       [ENTER]
A(1,1)? 5,1,2,0,4,1,4,2,1,3,2,2,5,4,0,0,1,4,1,3,4,3,0,3,4 [ENT]
Eigenvalue =  5.67255139609
```

The general case of finding *all* eigenvalues of any square matrix is hardly any more difficult, as demonstrated by this 5-liner that will accept any square matrix and will promptly compute *both* the coefficients of its Characteristic Polynomial and all its eigenvalues, real and/or complex:

```
DESTROY ALL @ OPTION BASE 0 @ FIX 5 @ INPUT "N=";N @ M=N-1
DIM A(M,M),U(M,M),C(N),D(N,N) @ COMPLEX E(N) @ MAT INPUT A
MAT D=CON @ MAT U=IDN @ FOR I=0 TO N @ IF I THEN MAT A=A-U
C(I)=DET(A) @FOR J=1 TO N @ D(I,N-J)=I^J @ NEXT J @ NEXT I
MAT C=SYS(D,C) @ MAT DISP C @ MAT E=PROOT(C) @ MAT DISP E
```

Once the matrix has been entered, all it does is use **DET** (Determinant) to find **N+1** values of the Characteristic Polynomial, then **MAT … SYS** is used to *explicitly* find (and display) the *coefficients* of this polynomial (see **IPOL** in section **8** above), then all its roots (the eigenvalues) real and/or complex are computed at once by **PROOT** and displayed as well. Let's test it with the same 5x5 matrix as before:

```
>RUN      [ENTER]
N=5       [ENTER]
A(0,0)? 5,1,2,0,4,1,4,2,1,3,2,2,5,4,0,0,1,4,1,3,4,3,0,3,4 [ENT]
     -1.00000       19.00000       -79.00000      -146.00000
     1153.0000     -1222.00000
     (1.49766, 0.00000)  (3.36188, 0.00000)  (-3.55784,0.00000)
     (5.67255,-0.00000)  (12.02575,0.00000)
```

so the Characteristic Polynomial is :

$$P(x)= -x^5+19*x^4-79*x^3-146*x^2+1153*x-1222$$

and, as expected, all five eigenvalues are *real* (imaginary parts = 0):

$x_1 = 1.49766$, $x_2 = 3.36188$, $x_3 = -3.55784$

$x_4 = 5.67255$, $x_5 = 12.02575$

You can verify them by checking that their *product* equals the matrix *determinant* (**DET(A)=-1222**). To verify the Characteristic Polynomial, you can apply it to the matrix itself: the resulting value should be a *zero* matrix, i.e.: **P(A) → ZER**

13. Enter the MOB (Matrix Operations Benchmarks)

For comparison purposes, the HP-71B used in these tests displays **9.41** (seconds) when executing this multi-statement sentence right from the command line:

```
DESTROY ALL @ SETTIME 0 @ FOR I=1 TO 1000 @ NEXT I @ DISP TIME
```

All matrices are **NxN** square matrices, **REAL** or **COMPLEX**, filled up with random elements. Times for **SHORT** precision are very much the same as for **REAL**. Times for **INTEGER** precision (real values only) are slightly *worse.*

MAT X=SYS(A,B)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.11	0.26	10	5.50	30.79	20	30.36	-
5	1.23	5.60	15	14.73	89.10	25	54.02	-

MAT A=INV(A)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.07	0.16	10	8.40	60.89	20	61.32	-
5	1.26	8.37	15	26.65	199.72	25	117.09	-

MAT A=A+B and MAT A=A-B

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.06	0.06	10	0.53	0.94	20	1.95	3.62
5	0.18	0.27	15	1.12	2.06	25	3.03	-

MAT A=A*B and MAT A=TRN(A)*B

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.06	0.09	10	8.03	34.80	20	62.04	-
5	1.10	4.45	15	26.60	115.85	25	-	-

MAT A=TRN(A)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.04	0.04	10	0.33	0.38	20	1.42	1.62
5	0.11	0.12	15	0.78	0.90	25	2.33	2.63

MAT A=ZER

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.04	0.03	10	0.06	0.08	20	0.12	0.22
5	0.04	0.04	15	0.09	0.14	25	0.18	0.33

MAT A=(1) and MAT A=((1,1))

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.03	0.04	10	0.07	0.11	20	0.17	0.28
5	0.04	0.05	15	0.12	0.18	25	0.24	0.42

FNORM(A)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.06	0.07	10	0.75	1.46	20	2.87	5.72
5	0.22	0.40	15	1.65	3.23	25	4.48	8.95

CNORM(A) and RNORM(A)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
10	0.06	0.25	500	1.30	10.38	1000	2.61	-
100	0.29	2.11	700	1.96	14.53	-	-	-

DET(A)

N	REAL	COMPLEX	N	REAL	COMPLEX	N	REAL	COMPLEX
1	0.06	-	10	3.06	-	20	21.24	-
5	0.53	-	15	9.45	-	25	40.01	-

Summary

That's all. I hope this article succeeds in making you aware that not only is the Math ROM a real *must* add-on for the HP-71B in terms of power and capabilities, but also from the *enjoyment* perspective as well. A brave new world to explore !