HP Forum Archive 13

Matrix Trilogy [LONG]

*Message #1 Posted by **Valentin Albillo** on 17 June 2003, 4:44 a.m.*

Hi ya all, specially HP-15C & RPN lovers:

A new week has begun, so there you are, a new, original HP-15C quiz/challenge I've composed and solved for you to sharpen your RPN teeth. As always, though the actual solutions are particularized to the HP-15C, *the techniques featured can be useful for many other HP models* (HP-71B/Math, HP-42S, HP-41/Advantage) so you may want to have a go at it with said models as well; the intended solutions are certainly rewritable for those models (I've done that for the 71B, actually).

The quiz's theme is as follows: comprehensive and well-designed as the HP-15C's matrix capabilities are, there are some very useful functions that regrettably were *not* included in the instruction set. For instance, there's no "SUM" function which will return the sum of all the elements of a given matrix (!)

So, this week's quiz (kind of a trilogy, actually) centers around the *missing* "SUM" matrix function, asking for a general solution and two particular cases, as follows:

1) Given an arbitrary MxN matrix, which contains *arbitrary* values (positive, negative or zero), write a routine (LBL A, ..., RTN) which takes the matrix descriptor in the X-register as input, and returns *the sum of all the elements in the matrix*. For instance:

```
A =  |  3  -21  7  |  must return S = 3-21+7+2-10+8 = -11
     |  2  -10  8  |
```

The routine must be primarily optimized for *speed*, and subject to that, be as short as possible and consume the least resources (registers, labels, subroutine levels, flags, etc). We are only interested in the *sum* of the elements, *not* the matrix itself (which for all we care, might be destroyed in the main program as soon as your routine has returned, to free RAM).

Under these conditions, there is a solution in *20 steps* or less (LBL and RTN included), which takes some *10 seconds* to return the sum of all 60 elements of a 10x6 matrix.

2) Write an *specialized* routine for the particular case in which all the elements of the matrix happen to be arbitrary *sines and/or cosines* [i.e.: values of sin(x) or cos(x)]. For instance: (values obtained in radians, displayed in FIX 4)

```
A = | sin(2)   sin(4)   sin(1)  |  must return S = -0.1033
    | sin(6)   sin(5)   sin(3)  |
```

(of course you don't have to evaluate any sin/cos, they already come evaluated, you only have to add them up quickly).

For this special case, there is a solution in *13 steps* or less (LBL-RTN included) which takes some *5 seconds* to return the sum of all 60 elements of a 10x6 matrix.

3) Write another specialized routine for the case in which all the elements of the matrix happen to be arbitrary values of the *exponential function* [i.e.: values of e^x]. For instance (result displayed in FIX 4):

```
A = | e^1.2  e^1.4  e^1.1  |  must return S = 23.4835
    | e^1.6  e^1.5  e^1.3  |
```

For this particular case, there is a solution in *9 steps* or less (LBL-RTN included) which takes only *2.5 seconds* to return the sum of all 60 elements of a 10x6 matrix.

That's all. Give it a fair try, it's quite an interesting, genuinely useful problem, and I eagerly await for your own implementations and ideas (remember to test your routines against the 2x3 numerical cases given, and to time their speed for a suitably filled-up 10x6 matrix).

Best regards :-)

Edited: 17 June 2003, 6:21 a.m.

My Solution, Part 1

Message #2 Posted by ***Patrick*** *on 18 June 2003, 3:07 a.m.,*
in response to message #1 by Valentin Albillo

Once again, thanks Valentin for the opportunity to use the little grey cells.

For pedagogical reasons, I provide the answers to your questions in reverse order. I will also post separate messages for each question to allow the interested reader to pick up and continue the solution.

3) First, we consider a simpler case where the matrix consists of a single row. If you look at the instruction set of the HP-15C, you will find that [f][MATRIX][7] computes the matrix *row norm*. In the case of a one row matrix, this is almost what we want (i.e., the sum of values of the elements of the matrix). The difference is that the row norm takes the *absolute values* of the matrix elements first, before summing. However, in the restricted case of Valentin's question 3, this is not a problem: exponentials like e^1.2 are always *positive*. Putting this together, here is the solution to the problem for a one row, non-negative matrix:

```
001 LBL A
002 f MATRIX 7
```

```
003 RTN
```

Like, good thing the 15C is programmable, eh?

Well, consider now the more general case of a multi-row, but still positive, matrix. The solution here is not much harder: we simply re-dimension the matrix to have a single row! Of course, the sum of the matrix elements is not changed by the re-dimensioning operation, so here is the complete solution to Question 3:

```
001 LBL A
002 STO I
003 1            // = number of desired rows
004 RCL DIM I    // X,Y = matrix dimensions
005 *            // Total number of elements = number of desired columns
006 f DIM I      // Re-dimension the matrix to have one row
007 RUP          // Roll up the stack
008 f MATRIX 7   // Row norm = our answer
009 RTN
```

Re: My Solution, Part 1

Message #3 Posted by ***Valentin Albillo*** *on 18 June 2003, 6:17 a.m.,*
in response to message #2 by Patrick

Hi, Patrick & everyone following this thread:

Absolutely correct. My own original solution was as follows, with comments describing how it deals with

```
     A = | e^1.2  e^1.4  e^1.1  |  which must return S = 23.4835
         | e^1.6  e^1.5  e^1.3  |
```

For this special case (*all matrix elements are >= 0*), there is a solution in 9 steps or less (LBL-RTN included) which takes only 2.5 seconds to return the sum of all 60 elements of a 10x6 matrix

```
      LBL A         begins routine. Say we've got 2x3 matrix above in the X-register
      STO I         for indirect addressing of the matrix
      EEX           1 (EEX is slightly faster than 1 itself)
      RCL DIM I     recall matrix dimensions to X,Y: X -> 2,  Y-> 3
       *            total number of elements = 2*3 = 6, Y holds 1
      DIM I         redimension the matrix to 1x6
      RCL I         recall the matrix descriptor to X
      MATRIX 7      find the sum of the absolute values of all elements = 23.4835
      RTN           returns to the calling program
```

Of course, this works not only for values of e^x, but for any other values as long as they all are >= 0.

Best regards.

My Solution, Part 2

Message #4 Posted by ***Patrick*** *on 18 June 2003, 3:32 a.m.,*
in response to message #1 by Valentin Albillo

2) Now to answer Valentin's second question, where the matrix elements consist of the sines and cosines of various angles. Well, this type of matrix is no longer non-negative, so the technique we used in Part 1 does not apply directly. However, the essential attribute of sines and cosines is that they are never less than -1. So, if we were to add 1 to every element of the matrix, we *would* have a non-negative matrix again and the earlier technique could be used. Of course, we've changed the sum of the elements of the matrix by doing this, but in an entirely predictable way: we've added to the sum a number exactly equal to the number of elements in the matrix!

Here is a program which is based on this algorithm. It adds one to each element of the matrix, finds the row norm, then subtracts from that value the total number of matrix elements.

```
001 LBL A
002 STO I
003 1
004 RCL DIM I
005 *
006 f DIM I          // redimension matrix to a single row
007 x<>I             // put num elements into I, matrix into X
008 STO RESULT       // prepare for matrix arithmetic
009 +                // add one to matrix
010 f MATRIX 7       // calculate row norm
011 RCL - I          // subtract num of matrix elements from total
012 RTN
```

Re: My Solution, Part 2

Message #5 Posted by ***Valentin Albillo*** *on 18 June 2003, 6:34 a.m.,*
in response to message #4 by Patrick

Hi, Patrick & everyone following this thread:

Again, absolutely correct. My own original solution was as follows, with comments describing how it deals with

```
     A = | sin(2)   sin(4)   sin(1)  |  which must return S = -0.1033
         | sin(6)   sin(5)   sin(3)  |
```

For this special case, there is a solution in 13 steps or less (LBL-RTN included) which takes only 5 seconds to return the sum of all 60 elements of a 10x6 matrix.

```
        LBL A         begins routine. Say we've got 2x3 matrix above in the X-register
        STO I         for indirect addressing of the matrix
        STO RESULT    we'll operate on the matrix itself
        EEX           1 (EEX is slightly faster than 1 itself)
         -            subtracting 1 from all elements makes them all negative or 0
        EEX           1 (again, faster)
        RCL DIM I     recall matrix dimensions to X,Y: X -> 2,  Y-> 3
         *            total number of elements = 2*3 = 6, Y holds 1
        DIM I         redimension the matrix to 1x6
        RCL I         recall the matrix descriptor to X, Y holds 6
        MATRIX 7      find the sum of the absolute values of all elements = 6.1033
         -            subtracting it from 6 gives the result sum = 6-6.1033 = -0.1033
        RTN           returns to the calling program
```

As you may see, your solution is one step shorter (thanks to that nifty X<>I, RCL-I technique, though the byte count is the *same*, because RCL-I requires two bytes) but there is another difference: you *add* 1 to all elements in the matrix, to make them all *positive* or zero, while my solution *subtracts* 1 from all the elements in the matrix, to make them all *negative* or zero! :-) In this particular case of sines and cosines, it doesn't matter at all, of course. Otherwise, your routine would work unchanged if all elements where >= -1 while mine would work unchanged if all elements where <= +1.

Of course, this works not only for values of sin(x), but for any other values as long as they all are <= 1 (for my solution, or >= -1 for yours). By changing it slightly, it will work as well whenever you know that your values are all of them less than some limit or all of them greater than some limit. The (very easy) details are left as an exercise for the reader. :-)

Best regards.

My Solution, Part 3 (start)

Message #6 Posted by ***Patrick*** *on 18 June 2003, 3:42 a.m.,*
in response to message #1 by Valentin Albillo

1) Finally, to the general question that Valentin posed, his question number 1.

No longer do we have either a positive matrix, nor a matrix nicely bounded below so that we can easily make it positive ... or do we?

It is late here and I do not have the time to give the entire solution right now, so I encourage others to fill in the remainder of the solution. Here is a hint. We've seen what the row norm does for a matrix with a single row. What does it do for a matrix with a single *column*?

Good night for now and happy programming...

[Return to Index | Top of Index]

GTO Go back to the main exhibit hall