HP Forum Archive 13

[Return to Index | Top of Index]

Matrix Convolutions

*Message #1 Posted by **Valentin Albillo** on 5 June 2003, 6:10 a.m.*

Here's another little HP-15C Quiz/Challenge to try your hand with this weekend. Despite being focused on the HP-15C, the techniques used in the intended solution may be applied to many other models, such as the 42S, 41C/Advantage, etc. Also, it's not just a *theoretical* quiz but the solution is *very useful* as well, in the spirit of that wonderful book, "Calculator Tips & Routines", by John Dearing.

The question is, as you may know the Identity matrix is defined as a square matrix with ones on the main diagonal and zeros elsewhere, like this example, particularized for a 3x3 square matrix:

```
1  0  0
0  1  0
0  0  1
```

The HP-71B/Math can assign the Identity matrix to any square matrix using just one command:

```
MAT A = IDN
```

but there's no such operation in the HP-15C instruction set, so the challenge is:

Write a subroutine (LBL A, ..., RTN) which will convert matrix A, of arbitrary dimensions from 1x1 to 7x7, into the Identity matrix. The subroutine may assume that matrix A has already been dimensioned as a square matrix (1x1 to 7x7). The subroutine must be as short and fast as possible.

By the way, this *exact* topic is covered in the amazing "HP-15C Advanced Functions Handbook", page 119, under the title "Constructing an Identity Matrix". There, the very creators of the HP-15C microcode software give their attempt at constructing the Identity Matrix. However, though most of the programs and routines in that book are extremely optimized, that's not so in this case. Their published routine, particularized for matrix A, is 12 steps long (including LBL and RTN), and takes 32 seconds in the case of a 7x7 matrix.

On the other hand, my solution is 2 steps shorter (i.e.: 10 steps long, including LBL and RTN) and some 10 times faster (i.e.: 3 seconds for a 7x7 matrix). Let's see yours !! :-)

Edited: 5 June 2003, 9:24 a.m. after one or more responses were posted

Re: Matrix Convolutions

*Message #2 Posted by **Vieira, Luiz C. (Brazil)** on 5 June 2003, 7:09 a.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin, folks;

This is my suggestion. It is 10 steps long when counting program body only (not counting LBL 0) and runs fast. I tested with a 5Ũ5 matrix and it took aprox. 4 seconds. I took the advantage of recalling row number an storing it as line number instead of testing if both are equal. That's the key for speed and size.

Thanks for the challenge; I had to stop and think of it carefully. Hope you like it.

Solution: enter matrix descriptor in X-register and press [GSB] 0

```
001 [f] LBL 0
002      0
003      Ũ
004     STO I
005 [f] MATRIX 1
006 [f] LBL 1
007     RCL 1
008     STO 0
009     EEX
010   ᵘ STO (i)
011     GTO 1
```

Obs:

1) steps #2 and #3 ensure matrix is cleared and may be suppressed if you are using only recently created matrices
2) step #4 sets descriptor in X-register as index
3) step #10 must be keyed in in USER mode

Best regards.

Luiz C. Vieira - Brazil

Edited: 5 June 2003, 1:38 p.m.

Re: Matrix Convolutions

*Message #3 Posted by **Patrick** on 6 June 2003, 10:55 p.m.,*
in response to message #1 by Valentin Albillo

I have been trying my hand at your latest quiz, Valentin, but I have so far been stumped as to how you get the speed you claim!! Hats off to Luiz for a very credible solution. Luiz, between us we have to beat this guy's time somehow!?

I have an idea for a solution which, alas, does not work for a matrix when all you have is the descriptor in the X register (I call this a paramterized matrix). I think it will work easily enough for, say, matrix "A", but not for a parameterized matrix.

I will describe my algorithm in the case of a 3x3 identity matrix, without providing a corresponding keystroke program:

1. First, create a 1x3 matrix and store all ones in it:

```
A = [ 1 1 1 ]
```

2. Resize this matrix to 4x3. The 15C will automatically supply zeroes for the new elements.

```
A = [ 1 1 1 ]
    [ 0 0 0 ]
    [ 0 0 0 ]
    [ 0 0 0 ]
```

3. Transpose into a 3x4

```
A = [ 1 0 0 0 ]
    [ 1 0 0 0 ]
    [ 1 0 0 0 ]
```

4. Resize to 3x3. When this happens, the 15C will read the elements from the current matrix in "book" order and place them into the new matrix in "book" order. Excess elements are discarded. This gives you:

```
A = [ 1 0 0 ]
    [ 0 1 0 ]
    [ 0 0 1 ]
```

Voila!

Most of this can be done easily with a parameterized matrix and I suspect that this would operate very quickly since most of the functions are native 15C MATRIX functions and not very complex ones at that (not like matrix inverse).

The part that is hard in the parameterized case is to store all 1's into the original 1x3 matrix. You can do this easily with a known matrix but not so simply when the target matrix is parameterized. For some reason, you cannot say [STO][(i)] when X is an integer and I is a matrix descriptor. Of course, you could always multiply the matrix by zero and then add one to it, but that is totally inelegant and I won't go there! ;-)

Try this listing

*Message #4 Posted by **Vieira, Luiz C. (Brazil)** on 7 June 2003, 1:34 a.m.,*
in response to message #3 by Patrick

Hey, Patrick;

Thank you for your words, but you know what? Your solution is a lot clever. I implemented it, but I used a large number of steps. Maybe you can reduce it.

```
001 f LBL any
002   STO RESULT
003   STO I
004   0
005   Ũ
006   1
007   +
008   RCL DIM I
009   1
010   x<>y
011 f DIM I
012   +
013   x<>y
014 f DIM I
015 g R^
016 f MATRIX 4
017   Rv
018   ENTER
019 f DIM I
```

The program will take the number of rows as a reference and consumes a lot of time filling matrix with ones (lines 4 to 7). I tried using these steps when indexed matrix was a single-line matrix, after step #11, but it would increase stack manipulation and a few extra steps, so I decided for a "shorter" routine instead of a faster one.

Again, if step #2 is removed, step #5 (Ũ) returns the matrix descriptor defined as result. Default definition for RESULT is matrix C and I forgot to add STO RESULT in my first listing, so step #5 (in fact, #4 if step #2 is removed) returned [C 3 3] in the display.

I hope this is what you had in mind.

Hey, Valentin, your turn now...

Best regards.

Luiz C. Vieira - Brazil

Edited: 7 June 2003, 3:11 a.m.

A faster, longer version

*Message #5 Posted by **Vieira, Luiz C. (Brazil)** on 7 June 2003, 8:08 a.m.,*
in response to message #3 by Patrick

Hey, Patrick;

I tried a faster (and it is) instead larger (2-bytes "plus") version for your algorithm and got this one:

```
001 f LBL any
002   STO RESULT
003   STO I
004   RCL DIM I
005   ũ
006 g LSTx
007 f DIM I
008   RCL I
009   0
010   Ũ
011   1
012   +
013   Rv
014   +
015 g LSTx
016 f DIM I
017 g R^
018 f MATRIX 4
019   Rv
020   ENTER
021 f DIM I
```

The program will take a square matrix in X-register. Filling a single-line matrix with ones (lines 9 to 12) is somewhat fast (faster than mine), but this routine is 2-steps longer than previous version. Maybe you can still reduce it (I did not try that hard to shrink it...)

Best regards.

Luiz C. Vieira - Brazil

Matrix Convolutions: no one else?

*Message #6 Posted by **Vieira, Luiz C. (Brazil)** on 8 June 2003, 1:11 p.m.,*
in response to message #1 by Valentin Albillo

Hey Valentin, Patrick;

It seems it's only us in this party. To be honest, I posted to keep it for two more days... Sorry, Dave :)

Patrick, I want to know if you tried the programs I posted. Any comments? I found no other way but the "unelegant" (you called it so...) *0 Ũ 1 +* to fill the indexed matrix with "ones". Have you seen how fast is your solution? Except for feeling the matrix (an internal loop), the absence of "user" loops makes it faster.

Best regards.

Luiz C. Vieira - Brazil

Edited: 8 June 2003, 2:09 p.m. after one or more responses were posted

Re: Matrix Convolutions: no one else?

*Message #7 Posted by **Patrick** on 8 June 2003, 2:03 p.m.,*
in response to message #6 by Vieira, Luiz C. (Brazil)

Hey Luiz

Thanks for your great implementations. I particularly liked the way you used divide to get the value 1 you needed in the second version. Quite clever stack manipulations.

I did key in your program and tried it in SST mode. Quite cool.

I've had another idea overnight that might make things better, although I haven't thought things through yet this morning. The idea is to overcome the difficulty of setting the parameterized matrix to all one's by using a second, non-parameterized matrix. You set, say, matrix "E" to all ones, then you do some sort of operation with E, with RESULT set to your parameterized matrix. It could turn out that "E" is the very same matrix as the parameterized one, but I don't think that matters.

What I don't like about this solution is the fact that you have to use another matrix. On the other hand, you can just barely fit all this into memory. If you think about memory usage you get:

Max size of parameterized matrix = 7x8 = 56 elements. Max size of temporary matrix = 1x7 = 7 elements.

Total = 56+7 = 63 elements out of 64 available. This doesn't leave you much program space, though! This leads me to believe we are on the right track. Unfortunately, Valentin has not been forthcoming on his incredibly optimized solution. We'll just have to wait for the master to educate us!!

If the HP15C has more memory...

*Message #8 Posted by **Vieira, Luiz C. (Brazil)** on 8 June 2003, 4:01 p.m.,*
in response to message #7 by Patrick

... we'd be claiming for more! Human beings are not easy to satisfy as for their (our) needs...

About the HP15C memory: if we look at the 2 Ũ 2^n growing ($64 = 2^3$ Ũ 2^3), than the next step would be a 16 Ũ 16 maximum matrix, that would lead us to 256 registers and 1792 bytes of memory, what is equivalent to the HP41's Quad Memory Module. Cool!

Back to business: thanks for your comments. I try to keep low-profile, stack-mostly solutions, but there is always a limit. When you have fixed-partition memory (HP65, HP55, HP67/97, HP25, HP33... any others?), using registers is up to you, but when movable partition is the case (HP34, HP38, all Voyagers, Coconuts, HP42... others?) then using numbered registers mean at least nine extra bytes. Yes, nine: the seven bytes used by the register itself and both STO / RCL to access it's contents. A lot of people forget about that: what is a register good for if you do not store something there for later retrieval? The best case is when you manually store something in a register before running a program, so you'll use only one RCL, meaning eight bytes. I thought an article about Stack Manipulation Ũ Storage Registers would be a good reading, but I didn't got through it so far.

In my first version, I considered any matrix descriptor in X-register, so I could not use the

```
   ũ
[f]LSTx
```

trick: X- and Y-register contents might be different, and I mentioned I used column numbers as reference, remember? As it was not an orthodox maneuver, I thought considering a square-matrix in X-register should not be accomplished. In the second version, this is mandatory to reduce program steps. Also, I could use the remaining "1" in LASTx (step #7):

```
001 f LBL any
002   STO RESULT
003   STO I
004   0
005   Ũ
006   1
007   +
008   RCL DIM I
009 g LSTx
010   x<>y
```

```
011 f DIM I
012   +
013   x<>y
014 f DIM I
015 g R^
016 f MATRIX 4
017   Rv
018   ENTER
019 f DIM I
```

provided matrix in X-register is a square matrix. Anyway, final listing has the same number of steps (I did not check for bytes).

As you mentioned, let's wait for Valentin's final words. I'm curious a lot to see his version.

You know, I tell my students that the best thing in a math problem is the fact that each math problem, in most of the cases, accepts many, many solutions. There are rare math problems with only one solution and others with no solution. Then we can use a human solution that, in most cases, is a simple behavioral change so the original math problem no longer exists and the new one has a solution, now.

Best regards.

Luiz C. Vieira - Brazil

Re: Matrix Convolutions

*Message #9 Posted by **Valentin Albillo** on 8 June 2003, 6:43 p.m.,*
in response to message #7 by Patrick

Hi, Patrick, Luiz, & everyone else following this thread:

First of all thank you very much for your interest in my humble 15C quiz, I've found your solutions and ideas very interesting and imaginative, to say the least. You both certainly are accomplished 15C programmers, knowing every hook and cranny.

However, may I point it out that interesting as your code is, it is trying to find a solution under *extended conditions* not originally asked for (such as starting from a matrix descriptor in the X register), which are not required in my original challenge. May I bring to your attention the fact that the only condition explicitly mentioned and required is that the routine expects a square matrix A to have been previously dimensioned (from 1x1 to 7x7). That's the solution the quiz is asking for.

So, you don't need to rack your brains for a general solution which will work for any matrix the user cares to pass in the X-register or I-register or whatever. Just make it work for a square matrix called 'A', which is passed *nowhere*: the subroutine simply assumes it has been previously dimensioned and works with it. Of course, the solution can then be easily altered to work with an arbitrary matrix, but that's not the point.

Under that original condition, I can state that there is a solution in *10 steps* (including LBL and RTN, i.e: 8 steps for the body of the subroutine) that can construct the Identity matrix for any matrix A ranging from 1x1 to 7x7, and taking *3 seconds* for the 7x7 case. (Your routine *must* include a final RTN, as you may not assume it can be placed right at the end of the program, there may be other subroutines as well and not all of them can be placed at the very end)

Luiz's first solution is a worthy first attempt, taking 12 steps (including LBL and RTN) and some 8 seconds for a 7x7 matrix. But you can do better ! Just stick to the original specifications, right ? :-)

I'll wait for any and all improved solutions under the condition given, and in any case will post my own next Tuesday. It really shows just how well all of HP-15C's command set instructions work together, complementing each other incredibly nicely.

Again, thanks for your interest and best regards.

[P.S.] Patrick, I did send you a second batch of documents, but never received your confirmation. Did you get them ?

This must be it!?

*Message #10 Posted by **Patrick** on 9 June 2003, 3:37 p.m.,*
in response to message #9 by Valentin Albillo

I think I have it, Valentin. Here is a 10 step solution which runs quite fast. I think the speed is in the same ballpark as yours, approximately 3.6 seconds for a 7x7: it took exactly 6 minutes to execute 100 iterations, including the timing loop's control code.

Here it is, written specifically for matrix A:

```
001   LBL A
002   0
003   STO MATRIX A    // zero out the matrix
004   f MATRIX 1      // set R0, R1 to 1
005   x<>0            // set R0 to 0, X to 1
006   LBL 1
007   STO + 0         // bring R0 up to the same value as R1
008 u STO A           // store 1 in next diagonal element
009   GTO 1
010   RTN
```

The small "u" on step 008 means to code this program step in User mode so that the values in R0 and R1 are automatically incremented. This also ensures that step 009 will be skipped at the end of the matrix.

The operation of this routine is quite simple. After storing zero into all elements of the matrix, it steps through just the diagonal elements setting them equal to one. Step 007 is used to make sure the row number keeps up with the (automatically incremented) column number.

It took me a long time to go from a previous 11 step solution to this one, the trick being step 005 which replaced two previous instructions.

Once again, thank you Valentin for providing such an entertaining challenge! Much enjoyed.

P.S.> Valentin... yes thanks for your second mailing. Will respond to you about it shortly.

Re: This must be it!? (parameterized)

*Message #11 Posted by **Patrick** on 9 June 2003, 4:23 p.m.,*
in response to message #10 by Patrick

The solution for the parameterized case is much less elegant, but executes in about the same amount of time.

Assuming a matrix descriptor for a square matrix is in X:

```
001   LBL A
002   STO I          // need to use indirect matrix calls
003   RCL DIM I      // get original dimensions into X,Y
004   0
005   f DIM I        // deallocate target matrix
006   STO 0          // prepare R0 for later iteration
007   Rv             // original dimensions --> X,Y
008   f DIM I        // restore original matrix dimension
009   1
010   STO 1          // prepare R1 for iteration
011   LBL 1
012   STO + 0        // make R0 equal R1
013 u STO (i)        // store 1 into diagonal element
014   GTO 1
015   RTN
```

The loop LBL 1 is pretty much the same as before. What is different is all the kafuffle in getting the matrix to be initially equal to zero. I chose to redimension the matrix to zero and then restore it rather than multiplying the matrix by zero. The latter technique would require fewer program steps but longer execution time -- the traditional tradeoff.

It is really too bad you can't say [STO][MATRIX][I] (or perhaps [STO][MATRIX][(i)]) with a scalar in X. This is just a strange quirk in the 15C's instruction set. Other very similar commands allow the use of the indirect register. Maybe the 15C Plutonium will fix this quirk!

Congrats, Patrick!

*Message #12 Posted by **Vieira, Luiz C. (Brazil)** on 9 June 2003, 5:21 p.m.,*
in response to message #10 by Patrick

Hi, Patrick;

if this is not the answer, than Valentin has another solution. Congrats! Great programming.

And Valentin, about this post: your comments are perfectly correct. I just should add "In a math analysis..." at the beginning.

I second Patrick in his words. This was teasing.

Best regards.

Luiz C. Vieira - Brazil

Re: This must be it!?

*Message #13 Posted by **Valentin Albillo** on 10 June 2003, 4:34 a.m.,*
in response to message #10 by Patrick

Hi Patrick, Luiz, and everyone following this thread:

Patrick posted:

"This must be it !? I think I have it"

Yes ! Congratulations !! :-) My original solution was:

```
    LBL A
    CLX
    STO MATRIX A
    MATRIX 1
    X<> 0
    LBL 0
    STO+ 0
"u" STO A
    GTO 0
    RTN
```

so, as you can see, yours is identical but for the trivial change of "LBL 1" instead of my "LBL 0", and yours using "0" instead of my "CLX". "CLX" executes *faster*, as it doesn't have to roll up the stack, etc, as "0" does.

I think this solution is unique and can't be bettered. Matter of fact, it relies on two happy 'coincidences':

- the fact that the "0" used to blank the matrix is precisely the value needed to initialize the index
- and the fact that the "1" needed to fill up the main diagonal is also the exact value required to sync up the row index with the column index !

Where it not for those two 'coincidences', the routine would be longer.

Now you can also see why I 'insisted' in a specific matrix A, instead of a parameterized, arbitrary matrix in the X or I registers: there's *no* STO MATRIX I or STO MATRIX (i) instruction, so the solution would be longer, less elegant. :-)

As a final remark, if this routine were to be used repeatedly in an actual program, it could be made *one step shorter and somewhat faster* by making use of the rarely seen *rapid reverse branching* technique: the main program would store the proper line number, then it would call the routine, like this:

```
      ...
36    92
38    CHS
39    STO I
      ...
47    GSB A
      ...
63    GSB A
      ...
87    LBL A
88    CLX
89    STO MATRIX A
90    MATRIX 1
91    X<> 0
92    STO+ 0
93"u" STO A
94    GTO I
95    RTN
```

The "GTO I" would branch to step 92, "STO+0", so the loop consists of only three steps, and runs faster. Besides, the GTO itself is *much faster as well*, as it goes *directly* to step 92, without performing a very time-consuming search through memory for "LBL 0".

Very glad that you liked my little quiz, congratulations again for finding the correct solution, and thanks for your kind comments and keen alternate techniques, the transposition trick is really neat.

Best regards.

Edited: 10 June 2003, 5:07 a.m.

To go where no one has gone before....

*Message #14 Posted by **Patrick** on 10 June 2003, 3:02 p.m.,*
in response to message #13 by Valentin Albillo

You know, I already had a program to do this very thing written down in my programming notebook before I read your challenge. Looking back on it now, I shudder at its awkward and inefficient algorithm. However, it was only when I read your challenge that I was inspired to improve upon it.

In fact, it is a common trait of problem solving that it is much easier to find an optimized solution when one is *known* to exist. It is much, much harder to find that solution in the first place, when failure and doubts are your constant companions.

Congratulations to you, Valentin, for deriving such a great piece of code without having the benefit we had of knowing such a thing even existed.

To say the least, I look forward to your next challenge.

Re: Matrix Convolutions: no one else?

*Message #15 Posted by **hugh** on 9 June 2003, 9:24 a.m.,*
in response to message #6 by Vieira, Luiz C. (Brazil)

i had a go, but couldnt come up with anything to beat your original posting.

i thought maybe i could divide the matrix by itself. eg DUP / as a cockey answer, but firstly, it doesnt give exactly the identity and secondly you cant set the result to the original.

.. im still wondering if there a mat op that might avoid a loop altogether.

Re: Matrix Convolutions: no one else?

*Message #16 Posted by **Vieira, Luiz C. (Brazil)** on 9 June 2003, 12:06 p.m.,*
in response to message #15 by hugh

Hi, hugh;

the ENTER / is actualy a solution, based on identity matrix definition and divide actual operation, that is 1/x Ũ (times the inverted matrix). I thought about it first as a guess, but we have two extra problems:

1 - memory space (only up to a 5 Ũ 5)
2 - running time

And for the HP15C, as you mentioned, the need to define a RESULT matrix. If you do'n't do that you'll get an error message.

Thanks; it's always good to hear from others. When I have no adequate or "competitive" answer, I seat and watch, too.

Luiz C. Vieira - Brazil

Re: Matrix Convolutions: no one else?

*Message #17 Posted by **Valentin Albillo** on 9 June 2003, 1:04 p.m.,*
in response to message #16 by Vieira, Luiz C. (Brazil)

Luiz posted:

the ENTER / is actualy a solution, based on identity matrix definition and divide actual operation, that is 1/x Ũ (times the inverted matrix)

Actually, regardless of the horrible timing (more than 1 minute for a 7x7 matrix, if it were possible) and accuracy (rounding errors while inverting and multiplying), it's not a solution at all, because:

- Obviously, filling a matrix with the elements of an Identity matrix must be independent of the actual matrix contents when the routine is called. Else, if the matrix happens to be singular (or nearly so), or even more commonly, has just been dimensioned and thus all its elements are zero, this procedure will fail and will not produce an Identity matrix at all. Matrix inversion is not defined for singular square matrices (analogous to division by zero), and though the HP-15C doesn't give an error and inverts instead a slightly changed matrix, the final result after multiplying is not an Identity matrix in that case.

- The HP-15C only allows the "division" operation between two matrices placed in X,Y if X is *not* the result matrix, so either it won't work at all or else you would need another temporary matrix of the same dimensions and specify the result matrix accordingly.

Best regards.

[Return to Index | Top of Index]