22nd September, 2023, 22:22

Post: #1

Valentin Albillo
Senior Member

Posts: 1,003
Joined: Feb 2015
Warning Level: 0%

[VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hi all,

> *This is my **1,000th** post in this New Forum (plus ~1,600 posts in the old one) since I joined back in 2015 and I want it to be a particularly noteworthy one, so here you are !*

Welcome to this second part of my latest thread ***SRC #015 - HP-15C & clones: Big NxN Matrix Inverse & Determinant***, which I created to commemorate the availability of the **HP-15C *Collector's Edition***.

> **Note**: This second part will also be included together with the first part in my future full article ***HP Article VA058 - Boldly Going - HP-15C CE Big Matrix Woes***. Although I created both parts at the same time I eventually decided to split the whole into two separate threads lest it would be too long and unwieldy if left in one piece.

The first part dealt with computing the *NxN real matrix* ***inverse*** and ***determinant*** for large values of *N (2 ≤ N ≤ 16,)* in particular for *N* exceeding the firmware limitation of *8x8* matrices. Additional posts to that thread by ***Werner***, ***J-F Garnier*** and myself further dealt with:

- *Real* matrix inversion and determinant up to 13x13 (my *OP*)
- *Real* linear systems up to *13x13*,
- ***Complex* matrix inversion** up to ***6x6***
- *Complex* systems of equations up to *6x6*,
- Computing the absolute value of the determinant of a *complex* matrix up to 6x6,

using various approaches. See that thread for the details, a ***must*** read. So far so good.

However, in the particular case of ***complex matrix inversion*** we can do *better* than ***6x6***, as we'll see now.

The Problem

The approach that *Werner* and *J-F Garnier* followed in *Part 1* for inverting complex matrices is based in the technique featured in both the ***HP-15C CE Owner's Handbook*** *(p.165)* and the ***Advanced Functions Handbook*** *(p.128,)* consisting in converting the ***NxN*** *complex* matrix to a ***2Nx2N*** *real* matrix, which is then dealt with by using the built-in functions for real matrices, as well as pre- and post-applying a number of *ad-hoc transformations* (`MATRIX 2`, `MATRIX 3`, `Py,x`, `Cy,x`.)

This is most fine and dandy but it has a number of serious problems, among them:

1. It's quite a *cumbersome* procedure, with up to *4 ad-hoc* transformations to remember and apply and up to eight or nine *manual* steps to obtain the complex matrix inverse, let alone solving a *system* of complex equations. As ***Maximilian Hohmann*** said in this recent post *(my **bold**)*:

   > *"[...] I have been into this calculator thing since 1976 or so. Yes, I have a "real" 15C and the "SE" and now the "CE", but, for example, **the way complex matrices are dealt with is beyond my comprehension** [...] I could work my [way] through the manual and follow the examples, but if I then need to perform such a calculation three months later I would have to start from scratch again [...] by today's standards, **the way it is done is totally counter-intuitive** [...]"*

2. Converting the original ***NxN*** *complex* matrix to a ***2Nx2N*** *real* matrix requires *twice* the memory, which seriously limits the size of the largest complex matrix that can be inverted within the available memory.

   For instance, an *8x8 complex* matrix requires *128 registers ("regs" for short)* just to be stored, but *inverting* it using *HP*'s described conversion to a *2Nx2N real* matrix would require *twice* that, i.e. *256 regs*. This is extremely wasteful and matter of fact the *HP-15C CE* in *192-reg Mode* can't invert complex matrices larger than *6x6* using *HP*'s approach, as *7x7* and *8x8* require *196* and *256 regs*, respectively, which simply aren't available. So, **what to do ?**

My Solution

Once more, an entirely *new* strategy is needed. The main problem is that conversion to a *2Nx2N real* matrix uses ***4x*** the memory that an *NxN* real matrix would need, instead of *2x*, but implementing an *LU*-based procedure would result in a long program much slower than the *2Nx2N* approach.

To sidestep this problem I use an approach which avoids wasting memory by ***not*** converting the *NxN complex* matrix to a *2Nx2N real* one, while also using the built-in *microcode* real inversion instruction (`1/x`) for maximum speed, making sure it never has to invert a matrix larger than *8x8*. Enter ***split complex matrices***.

A *split complex matrix* is an *NxN complex* matrix **M** considered to be split into two *real NxN* matrices **A**, **B**, containing the *real* and *imaginary* parts, respectively, of the complex elements of **M**, like this:

```
     | 5 + 3i   4 + 7i     8      1 + 2i   6 + 6i |
     | 7 + 2i     5i     3 + 4i   8 +  i     1    |
M =  | 8 + 4i   2 + 5i   6 + 7i   3 + 8i     5    | = A + iB
     | 6 +  i   4 + 2i   3 + 7i   4 + 2i   6 + 5i |
     | 3 + 7i     i      8 + 7i   1 + 3i   2 + 4i |
```

where **A** *(real parts)* and **B** *(imaginary parts)* are:

```
     | 5   4   8   1   6 |         | 3   7   0   2   6 |
     | 7   0   3   8   1 |         | 2   5   4   1   0 |
A =  | 8   2   6   3   5 | ,  B =  | 4   5   7   8   0 |
     | 6   4   3   4   6 |         | 1   2   7   2   5 |
     | 3   0   8   1   2 |         | 7   1   7   3   4 |
```

and as it happens, there are efficient algorithms to invert such split complex matrices by interacting directly with their parts stored in *real* matrices, thus ***no*** complex arithmetic is ever needed as all computations are carried out in the *real* domain, with a noticeable speed advantage which is further increased because my routine spends most of its time executing *microcode*, not *RPN* user code.

Another important consideration is **code size**. Like the original one, the *HP-15C CE* has no way to save/load programs to/from mass storage (say by connecting to a laptop,) thus entering programs has to be done by *manually* typing them in, which becomes *exceedingly bothersome*, time-consuming and *error-prone* when the programs are long (say *100-200 steps* or more,) and all you see are numeric keycodes.

It's one thing if you're using a single long program all the time but entirely another if you want to use your *CE* for various purposes requiring assorted programs, as eventually you'll be forced to erase one to make room for another and this will get utterly inconvenient, slow and *annoying* pretty soon.

To wit, a program might be a fine achievement and work well but if the user has to painstakingly key in a large number of program steps to use it, chances are it won't be used much if at all, not even to just check it out. Thus, I think it's important that my solution is *very short*, to maximize its potential use and minimize the burden of loading it into the calculator.

The implementation: Complex Matrix Inversion up to 8x8

This ***31****-step (32-byte)* routine will invert *in place* an ***NxN*** split complex matrix **M** for ***1*** ≤ ***N*** ≤ ***8*** (subject to available memory.) It takes no inputs but the caller (the user or another program) must have previously dimensioned and populated the two real matrices **A**, **B** with the corresponding parts of **M**'s elements.

Once it returns, the original values in **A**, **B** will have been replaced with those of the *complex inverse matrix*, which the user or the caller program may then proceed to output or use as desired. In other words, this routine can be called from the keyboard or another program (in which case it could be directly embedded into it if called from a single location) but it doesn't perform any input or output operations, it just does the inversion *in place*.

Program listing

```
LBL C           001- 42,21,13
RCL MATRIX A    002- 45,16,11
RCL MATRIX B    003- 45,16,12
RESULT E        004- 42,26,15
 +              005-       40
1/x             006-       15
RCL MATRIX B    007- 45,16,12
RCL MATRIX A    008- 45,16,11
RESULT B        009- 42,26,12
 -              010-       30
X<>Y            011-       34
RESULT A        012- 42,26,11
 x              013-       20
```

```
CHS             014-       16
LASTX           015-    43 36
RESULT E        016- 42,26,15
1/x             017-       15
X<>Y            018-       34
RCL MATRIX B    019- 45,16,12
MATRIX 6        020- 42,16, 6
1/x             021-       15
RCL MATRIX A    022- 45,16,11
RESULT B        023- 42,26,12
 x              024-       20
RESULT A        025- 42,26,11
 +              026-       40
LASTX           027-    43 36
RCL MATRIX E    028- 45,16,15
RESULT B        029- 42,26,12
 -              030-       30
RTN             031-    43 32
```

Notes:

- This routine doesn't use or alter *any* numbered storage registers, including the three permanent index registers **R0**, **R1** and **RI**.
- Also, it uses **no** flags, labels (save `LBL C`), branching, loops (simple or nested), logic tests of any kind or scalar operations and it runs *sequentially* from its first to its last step, executing each *just once*, which means it executes *exactly **31** user-code instructions in all* (not hundreds or thousands like other approaches,) so it runs very fast (*0.93"* to invert a *5x5* complex matrix, *2.4"* for a *7x7* one.)
- The inversion is performed *in place*: once the process ends the elements of the complex inverse replace those of the original matrix (so *reinverting* the computed inverse would get back the original complex matrix) and for what is worth, on completion it leaves the **A** matrix *(real parts)* in ***Y*** and the **B** matrix *(imaginary parts)* in ***X***.
- The approach used requires ***3NxN** regs* instead of the *4NxN regs* required by the conversion to a real *2Nx2N* matrix. The resulting *NxN regs* saved can be used to allow processing larger matrices or for other purposes. After the inversion, yet *another* batch of *NxN regs* can be freed, as discussed next.
- Once the inversion is over, you can redimension auxiliary matrix **E** to ***0x0***, which frees *NxN regs* for other uses such as additional data or code (up to *7NxN* program steps.) For instance, **to solve an *NxN* system of *N* complex equations** you could dimension the constant and solutions matrices to be *Nx2* each, which would still leave enough memory for the program code to matrix-multiply the *inverse* matrix and the *constant* matrix to produce the complex *solution* matrix.

  > **Hint**: To redimension a matrix (say **E**) to ***0x0*** you don't need to execute **0, ENTER, DIM E**, just **0, DIM E** or **CLX, DIM E** will do no matter what's in the ***Y*** register (even if it holds a huge or negative number or even a *matrix descriptor*).

Requirements:

- The *maximum* size *NxN* complex matrix **M** you can invert depends on the memory available in your physical or virtual device, as per this table:

M	A,B,E	Regs	+Prog	Comments
1x1	1x1	3	8	-
2x2	2x2	12	17	-
3x3	3x3	27	32	-
4x4	4x4	48	53	Max. size w/ ***15C/64*** **but see (*)**
5x5	5x5	75	80	ditto ***CE/96***
6x6	6x6	108	113	-
7x7	7x7	147	152	ditto ***CE/192*** **but see (**)**
8x8	8x8	192	197	ditto ***DM15/M1B***

  so e.g. if you want to invert an *8x8* complex matrix you need *197 regs* available, which includes matrices **A**, **B** (and **E**,) plus *5 regs* to hold the routine itself. Auxiliary matrix **E** also needs to be *NxN* because the **HP-15C** can't multiply two *NxN* matrices *in place*, a third *NxN* matrix (**E**) is needed to store the result.

  > **Implementation note**: A ***7x7*** real matrix occupies *49 regs* so you can have only *three* such matrices simultaneously at any given time because using a fourth matrix would require *196 regs*, exceeding the *192 regs* available in the *CE/192*.
  >
  > To overcome this limitation, I've used a trick to make do with only three matrices. If there was room for a fourth matrix (which is the case when using a *CE/192* and dealing with ***6x6*** or smaller complex matrices,) a version of my routine limited to those sizes would be *~30% faster*.

- The routine works *as-is* for complex matrices up to and including ***8x8***. Larger complex matrices wouldn't work because of both the *8x8* firmware limitation for *(real)* inversion and memory constraints. As of *Sept. 2023* no physical or virtual devices exist which provide more than *229 regs* because the **HP-15C** has a *RAM* limit of ***256** regs* and *27* of those are used internally by the firmware.

  This means that matrices *9x9* or larger can't be tackled using this routine on any currently existing device. However, it

might be possible that some future patch would override those restrictions, in which case this routine would invert complex matrices larger than *8x8* without any modifications.

- The matrix **A+B** must be *invertible*, i.e. ***det(A+B) # 0***. For most real-life uses this will be the case but if this condition isn't met there are slight variations to this routine that would work Ok in those cases.

(*) Observation re the *original* HP-15C and *4x4* complex matrices:

- The original **HP-15C** could invert a ***4x4*** complex matrix but it took all *64 regs* available and was a complicated, completely *manual* process as there wasn't *any* memory left for program code. On the other hand, running my routine is a fast, *automated* process that leaves out all the drudgery (transformations, etc.) and still leaves *11 regs* (up to *77* extra program steps) free for additional code or data.

  Furthermore, now it seems possible to **solve a system of 4 complex equations** with equal ease, like this (see **Worked Example** below for basic details):

  - Initialize and store the data in the **A**, **B** matrices.
  - Call my *complex inversion* routine. When it ends, the complex inverse is stored in **A**, **B** and there's still *11 regs* free. Optionally, output the *inverse* by the usual procedure (`RCL A, B` in `USER` mode) either now or after the system is solved, as the solving procedure doesn't affect the inverse.
  - Get rid of auxiliary matrix **E** (redimension it to ***0x0***.) This frees another *16 regs*, so there's now ***27*** *regs* available for what follows.
  - Dimension both the *constant* matrix (say **C**) and the *solution* matrix (say **D**) to be *4x2* and populate the constant matrix. This will leave *11 regs* (up to *77* program steps) still available for the matrix-*multiplication* code, which is left as a fairly easy exercise for the reader (just a loop which multiplies each row of the *inverse* by the *constant* matrix using complex arithmetic.)
  - Run said matrix-multiplication code, which should use the inverse's data in **A**, **B** to multiply the *inverse* and the *constant* matrix **C**, storing the result in the complex *solution* matrix **D**.
  - Output or otherwise make use of the complex solution matrix.

  All these steps can be coded as a *single* program which inverts the complex matrix, gets rid of **E** to make room, dimensions the *constant* and *solution* matrices and ***stops*** for the user to populate the constant matrix; once done, the user presses **R/S** to continue execution and the *solution* matrix is computed for the user to output or other code to use it.

  The inverse matrix is left *undisturbed* so other systems having the same original matrix and different constant matrices can be solved outright using the same inverse matrix. Moreover, the *original* matrix can be recovered if desired (negligible rounding errors aside,) by ensuring that the inverse remains intact and there's at least *16 regs* available, then *reinverting* the inverse matrix with a **GSB C**.

  This would completely *automate* the task of *inverting* a *4x4* complex matrix or *solving* a system of *4* complex equations on an *original* **HP-15C**, with **no** complicated transformations or manual steps whatsoever.

(*) Observation re the HP-15C CE and *8x8* complex matrices:

- Though there's not enough room in the **HP-15C CE** in *192-regs Mode* to invert an ***8x8*** complex matrix **M** by running the routine featured here (*197 regs* would be needed; the routine itself wouldn't fit) there's just enough room for the split matrices **A**, **B** and the auxiliary matrix **E** (*192 regs* in all,) so the *8x8* complex matrix **can** be inverted in a pinch if the user executes the *29* program instructions *manually* in sequential order. The procedure would be like this:

  - Initialize and store the data in the **A**, **B** matrices.
  - Carefully execute *manually* the *29* instructions from `002 RCL MATRIX A` to `030 -` in sequential order. Assuming you're reasonably proficient with the *HP-15C*, this should take ~ *3 min*. or less.
  - Output or otherwise make use of the inverted complex matrix data in **A**, **B**. Now you can redimension auxiliary matrix **E** to ***0x0*** to free ***64*** *regs* (up to *448* program steps) for further processing using the inverse matrix just computed (e.g. solving a *system* of ***8*** *complex equations*.)

  This procedure is perfectly workable and reasonably fast if you want to invert an *8x8* complex matrix using a *CE/192* but you should be *very careful* as most errors keying in an instruction could mean having to *restart* from the beginning, including re-storing the elements anew in **A**, **B**.

Worked example

The following example can be run on the **HP-15C CE** *(Collector's Edition)* in its *96-* or *192-regs Mode*, as well as in any physical/virtual clone with at least *80 regs* allocatable.

Note: The usefulness of this relatively small *5x5* example is *twofold*: first, to get to know how to use the routine and get comfortable using it and second, to ascertain that you've loaded it correctly into program memory by running it and checking the results it produces.

Invert the following *5x5 complex* matrix M: *(matrix taken from this J-F's post)*

```
      | 5 + 3i   4 + 7i     8      1 + 2i   6 + 6i |
      | 7 + 2i     5i     3 + 4i   8 +  i     1    |
M  =  | 8 + 4i   2 + 5i   6 + 7i   3 + 8i     5    |
      | 6 +  i   4 + 2i   3 + 7i   4 + 2i   6 + 5i |
      | 3 + 7i     i      8 + 7i   1 + 3i   2 + 4i |
```

- The **A** *(real parts)* and **B** *(imaginary parts)* matrices are:

```
      | 5   4   8   1   6 |         | 3   7   0   2   6 |
      | 7   0   3   8   1 |         | 2   5   4   1   0 |
A  =  | 8   2   6   3   5 | , B  =  | 4   5   7   8   0 |
      | 6   4   3   4   6 |         | 1   2   7   2   5 |
      | 3   0   8   1   2 |         | 7   1   7   3   4 |
```

- Ensure disabled complex stack and set 4 decimal places:

```
CF 8, FIX 4
```

- Allocate all memory to matrices, initialize them and dimension the real/imag parts **A**, **B** to ***5x5***:

```
1, DIM (i), MATRIX 0
5, ENTER, DIM A, DIM B
```

- Store the *real* parts of **M**'s elements into matrix **A**:

```
USER, MATRIX 1

5 STO A, 4 STO A, 8 STO A, 1 STO A, 6 STO A,
7 STO A, 0 STO A, 3 STO A, 8 STO A, 1 STO A,
8 STO A, 2 STO A, 6 STO A, 3 STO A, 5 STO A,
6 STO A, 4 STO A, 3 STO A, 4 STO A, 6 STO A,
3 STO A, 0 STO A, 8 STO A, 1 STO A, 2 STO A
```

- Store their *imaginary* parts into matrix **B**:

```
3 STO B, 7 STO B, 0 STO B, 2 STO B, 6 STO B,
2 STO B, 5 STO B, 4 STO B, 1 STO B, 0 STO B,
4 STO B, 5 STO B, 7 STO B, 8 STO B, 0 STO B,
1 STO B, 2 STO B, 7 STO B, 2 STO B, 5 STO B,
7 STO B, 1 STO B, 7 STO B, 3 STO B, 4 STO B
```

- **Compute *in-place* the complex inverse matrix M'**:

```
GSB C -> B      5 5    {in 0.93"}
```

- Recall the inverse matrix elements from **A** and **B**:

Real parts:

```
RCL A: -0.1197, RCL A: -0.0070, RCL A:  0.0635, RCL A:  0.0322, RCL A:  0.0285,
RCL A:  0.0247, RCL A: -0.0206, RCL A:  0.0232, RCL A: -0.0363, RCL A:  0.0638,
RCL A:  0.0277, RCL A: -0.0416, RCL A:  0.0183, RCL A: -0.0348, RCL A:  0.0439,
RCL A:  0.0397, RCL A:  0.0886, RCL A: -0.0975, RCL A:  0.0319, RCL A: -0.0036,
RCL A:  0.0587, RCL A:  0.0388, RCL A: -0.0482, RCL A:  0.0509, RCL A: -0.0549
```

Imaginary parts:

```
RCL B: -0.0115, RCL B: -0.1044, RCL B:  0.0663, RCL B:  0.0940, RCL B: -0.1395,
RCL B: -0.1038, RCL B: -0.0652, RCL B: -0.0403, RCL B:  0.0796, RCL B:  0.0945,
RCL B:  0.0430, RCL B:  0.0005, RCL B: -0.0459, RCL B: -0.0329, RCL B:  0.0005,
RCL B: -0.0081, RCL B:  0.1066, RCL B: -0.0765, RCL B: -0.0271, RCL B:  0.0520,
RCL B: -0.0181, RCL B:  0.0712, RCL B:  0.0676, RCL B: -0.1183, RCL B:  0.0114
```

and so the *5x5* complex inverse matrix **M'** is:

```
       | -0.1197-0.0115i -0.0070-0.1044i  0.0635+0.0663i  0.0322+0.0940i  0.0285-0.1395i |
       |  0.0247-0.1038i -0.0206-0.0652i  0.0232-0.0403i -0.0363+0.0796i  0.0638+0.0945i |
M' =   |  0.0277+0.0430i -0.0416+0.0005i  0.0183-0.0459i -0.0348-0.0329i  0.0439+0.0005i |
       |  0.0397-0.0081i  0.0886+0.1066i -0.0975-0.0765i  0.0319-0.0271i -0.0036+0.0520i |
       |  0.0587-0.0181i  0.0388+0.0712i -0.0482+0.0676i  0.0509-0.1183i -0.0549+0.0114i |
```

Notes:

- If in doubt, a simple way to *check* the inverse's correction is to *reinvert* it once you've written down its elements, which should still be undisturbed in memory. Simply run the routine again:

  `C` (in **USER** mode) or **GSB** `C` (out of **USER** mode) -> `B` *N N*

  and you'll get the original matrix back (ignoring negligible rounding errors,) which can be verified by using **RCL** in **USER** mode, as we did in the ***Worked Example*** above.

- Once you're done with using the routine you might want to free memory by resizing all matrices **A**, **B** and the auxiliary matrix **E** to ***0x0*** as well as resetting the row/col indexes and reallocating the default numbered storage registers **R0**-**R.9**, like this:

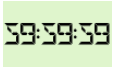
```
MATRIX 0, MATRIX 1, 19, DIM (i)
```

That's all, hope you enjoyed it and found it useful for your own purposes. Comments welcome.

V.

PM WWW FIND EDIT QUOTE REPORT

25th September, 2023, 16:52 **Post: #2**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Very nice, Valentin!
You make it increasingly harder for me to improve upon your code ;-)
I did manage to squeeze off a puny byte, a stack instruction, which will make little to no difference.
The one thing I miss in your extensive posts, however, is some explanation of the algorithm used.
So here goes, and it is based on my version of your code - which simply switches the roles of matrices E and B.

To invert a complex matrix A+iB = X+iY, you need to solve

```
A.X - B.Y = I
B.X + A.Y = 0
```

Add and subtract:
(This step is needed to make reasonably certain that the main matrix is invertible)

```
(A+B).X - (B-A).Y =  I
(B-A).X + (A+B).Y = -I
```

Let

```
E := B - A;
B := A + B;
```

then the equations become

```
B.X - E.Y =  I   (1)
E.X + B.Y = -I   (2)
```

multiply (2) by E.B^-1 and add to (1):

```
(B + E.B^-1.E).X =  I - E.B^-1
```

multiply (1) by E.B^-1 and subtract from (2):

```
(B + E.B^-1.E).Y = -I - E.B^-1
```

Then the algorithm becomes:
(with double inversion to keep the number of matrices needed down to 3)

```
E := B - A;
B := (A + B)^-1;
A := -E*B;
B := B^-1;
B := (B - A*E)^-1;
E := B*A;
A := E + B;
```

```
B := E - B;
```

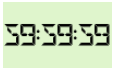
listing (30 lines, 31 bytes):

```
LBL C
RCL MATRIX B
RCL MATRIX A
RESULT E
-
RCL MATRIX A
RCL MATRIX B
RESULT B
+
1/x
RESULT A
x
CHS
RCL MATRIX B
RESULT B
1/x
X<>Y
RCL MATRIX E
MATRIX 6
1/x
RCL MATRIX A
RESULT E
x
RESULT A
+
RCL MATRIX E
RCL MATRIX B
RESULT B
-
RTN
```

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

26th September, 2023, 12:49 (This post was last modified: 27th September, 2023 17:36 by Werner.) **Post: #3**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Shamelessly ~~stealing~~ borrowing your idea, we can solve systems of equations the same way:

to solve (A+iB)*(X+iY) = C+iD, enter the matrices A,B,C and D, then (GSB) D.
The result will be in matrices C and D
48 lines, 52 bytes:

```
LBL D
RCL MATRIX D
RCL MATRIX C
RESULT E
-
RCL MATRIC C
RCL MATRIX D
RESULT C
+
RCL MATRIX B
RCL MATRIX A
RESULT D
-
RCL MATRIX A
RCL MATRIX B
RESULT B
+
1/x
RESULT A
x
CHS
RCL MATRIX B
RESULT B
```

```
1/x
X<>Y
RCL MATRIX D
MATRIX 6
1/x
RCL MATRIX C
STO MATRIX D
RCL MATRIX A
RCL MATRIX E
RESULT D
MATRIX 6
RCL MATRIX A
RCL MATRIX C
CHS
RESULT E
MATRIX 6
RCL MATRIX B
RCL MATRIX D
RESULT C
x
RCL MATRIX B
RCL MATRIX E
RESULT D
x
RTN
```

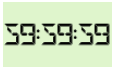
You can try out the above example with C the identity matrix and D all zeroes, effectively producing the inverse again.

This routine, incidentally, will allow solving a 4x4 complex system on an original 15C.
(and up to a 7x7 on the 15C2, or an 8x8 on the DM15L-M1B)
However, the original contents of A and B are lost so you can't solve a subsequent system.

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

27th September, 2023, 17:43 **Post: #4**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Not quite what you asked for, Valentin, but here's a short routine to multiply two split complex matrices.
Together with your inversion routine, it may be used to solve a system of equations as well.

This routine multiplies the complex matrices A+iB and C+iD, and places the result in C+iD.

```
C+iD := (A+iB)*(C+iD);
```

on condition that A is invertible. It leaves A intact, while B may suffer some accuracy loss.

```
(A+iB)*(C+iD) = (A.C - B.D) + i(A.D + B.C)
```

that is impossible to do with just one extra matrix, so we have to resort to a 'trick'

```
A.C - B.D = A*(C - A^-1.B.D)
A.D + B.C = A*(D + A^-1.B.C)
```

so we can do:

```
E := A;
B := E^-1*B;
E := C;
E := E - B*D;
D := D + B*C;
C := A*D;
D := C;
C := A*E;
E := A*B;
B := E;
```

If you leave off the last 5 lines, B is not restored, and it's 26 lines, 31 bytes.
31 lines, 37 bytes

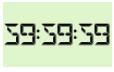
```
LBL D
```

```
RCL MATRIX A
STO MATRIX E
RCL MATRIX B
RCL MATRIX E
RESULT B
/
RCL MATRIX C
STO MATRIX E
RCL MATRIX B
RCL MATRIX D
RESULT E
MATRIX 6
RCL MATRIX A
RCL MATRIX B
RCL MATRIX C
CHS
RESULT D
MATRIX 6
RESULT C
x
STO MATRIX D
RCL MATRIX A
RCL MATRIX E
x
RCL MATRIX A
RCL MATRIX B
RESULT E
x
STO MATRIX B
RTN
```

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

29th September, 2023, 15:20 (This post was last modified: 3rd October, 2023 09:16 by Werner.) **Post: #5**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

[Update: shaved off 2 bytes ;-)]
Apologies for monopolizing this thread ;-)
Forget all that came before; well, save the previous post on multiplying two complex matrices, which may still be useful as is.
The following three small routines are all you need for complex matrix solving and inverting, in 'split' format.
Enter matrices A,B,C and D separately as before.
- to solve (A+iB)*(X+iY) = C+iD, press (GSB) C and (GSB) D
- to invert A+iB, do (GSB) C and (GSB) E
The GSB C part need only be done once - think of it as the LU-decomposition for regular matrices. So you can solve subsequent right-hand sides (C,D,D,..), and combine solving and inverting (C,D,D,..,E).
If you leave off LBL E, you have a routine of 48 bytes for solving complex equations, and subsequent ones, if needed. This routine thus replaces the one I previously posted, which couldn't handle subsequent solves - but it still fits in an original 15C and allows solving a 4x4 complex system. (the 15CE can solve a 5x5, the 15C-2 a 7x7 and the DM15L an 8x8 system)

The inversion routine alone needs 3.N^2 registers, the solve routine an additional 2N.M registers, where M is the number of right-hand side columns.
So, to solve a single 4x4 system, you'd need 3.4^2+2.4 = 56 registers. Since the program of 48 bytes uses 7 additional registers, it still fits in an original 15C.

Routine C (Factor) - 21 bytes
Routine D (Solve) - 27 bytes
Routine E (Invert) - 15 bytes
for a total of 63 bytes.

```
001 LBL C
002 RCL MATRIX B
003 RCL MATRIX A
004 RESULT E
005 -
006 RCL MATRIX A
007 RCL MATRIX B
008 RESULT A
009 +
010 1/x
```

```
011 RESULT B
012 x
013 CHS
014 RCL MATRIX A
015 RESULT A
016 1/x
017 X<>Y
018 RCL MATRIX E
019 MATRIX 6
020 RTN
021 LBL D
022 RCL MATRIX D
023 RCL MATRIX C
024 RESULT E
025 -
026 RCL MATRIX C
027 RCL MATRIX D
028 RESULT C
029 +
030 STO MATRIX D
031 RCL MATRIX B
032 RCL MATRIX E
033 MATRIX 6
034 RCL MATRIX A
035 /
036 RCL MATRIX B
037 RCL MATRIX D
038 CHS
039 RESULT E
040 MATRIX 6
041 RCL MATRIX A
042 RESULT D
043 /
044 RTN
045 LBL E
046 RCL MATRIX B
047 RCL MATRIX A
048 RESULT E
049 /
050 RCL MATRIX A
001 RESULT A
002 1/x
053 RESULT B
054 -
055 RCL MATRIX E
056 RCL MATRIX A
057 RESULT A
058 +
059 RTN
```

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

2nd October, 2023, 01:08 (This post was last modified: 3rd October, 2023 00:17 by Gerson W. Barbosa.) **Post: #6**

Gerson W. Barbosa
Senior Member

Posts: 1,522
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Werner Wrote:** (29th September, 2023 15:20)
> - to solve (A+iB)*(X+iY) = C+iD, press (GSB) C and (GSB) D

Thank you both, Werner and Valentín!
The matrix capabilities in the HP-15C were nice, but the methods in the manual for solving complex linear systems were somewhat complicated and not easy to remember. I would need to carry the manual along to use it for that task. Doing it programmatically is a great idea, I wonder why no one thought of this back then – at least I don't remember of any. These would have been handy when I got my 15C a few years later.

Here's a real-world example to illustrate the usage of your latest method. Perhaps not the best example as the main matrix has only real elements:

Three-phase voltage source (Wye), unbalanced resistive load (Δ), line voltage = 220V.

$R_1I_1 + 220\sqrt{3}/3\angle 120° = 220\sqrt{3}/3$

$R_2I_2 + 220\sqrt{3}/3 = 220\sqrt{3}/3\angle -120°$

$R_3I_3 + 220\sqrt{3}/3\angle -120° = 220\sqrt{3}/3\angle 120°$

$R_1 = R_2 = 12.1\ \Omega;\ R_3 = 24.2\ \Omega$

```
A =
| 12.1 0 0 |
| 0 12.1 0 |
| 0 0 24.2 |

B =
| 0 0 0 |
| 0 0 0 |
| 0 0 0 |

C =
| 190.5255888 |
|-190.5255888 |
| 0.000000000 |

D =
|-110 |
|-110 |
| 220 |

GSB C GSB D ->

X =
| I1 |
| I2 |
| I3 |

=

C =
| 15.74591643 |
|-15.74591643 |
| 0.000000000 |

D =
|-9.090909091 |
|-9.090909091 |
| 9.090909091 |

or
```

$I_1 = 18.18181818\angle -30°$

$I_2 = 18.18181828\angle -150°$

$I_3 = 9.090909091\angle 90°$

These are the load currents. The source currents are $I_1 - I_2$, $I_2 - I_3$ and $I_3 - I_1$.

Edited to add three missing negative signs (corrections in red).

EMAIL PM FIND QUOTE REPORT

Gerson W. Barbosa
Senior Member

Posts: 1,522
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Gerson W. Barbosa Wrote:** (2nd October, 2023 01:08)

> Perhaps not the best example as the main matrix has only real elements

Let's consider small inductive reactances in series with the voltage sources and get a set of equations that does require a complex system solver:

$(R_1 + XL_1)I_1 - XL_1I_2 + 220\sqrt{3}/3\angle 120° = 220\sqrt{3}/3$

$(R_2 + XL_2)I_2 - XL_2I_3 + 220\sqrt{3}/3 = 220\sqrt{3}/3\angle -120°$

$(R_3 + XL_3)I_3 - XL_3I_1 + 220\sqrt{3}/3\angle -120° = 220\sqrt{3}/3\angle 120°$

$R_1 = R_2 = 12.1\ \Omega;\ R_3 = 24.2\ \Omega;\ XL_1 = XL_2 = XL_3 = j0.1\ \Omega$

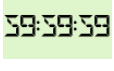
```
A =
| 12.1 0 0 |
| 0 12.1 0 |
| 0 0 24.2 |

B =
| 0.1 -0.1 0 |
| 0 0.1 -0.1 |
| -0.1 0 0.1 |

C =
| 190.5255888 |
|-190.5255888 |
| 0.000000000 |

D =
|-110 |
|-110 |
| 220 |

GSB C GSB D ->

X =
| I1 |
| I2 |
| I3 |

=

C =
| 15.74266458 |
|-15.89562348 |
| 0.07647945438 |

D =
|-9.352382547 |
|-8.958908239 |
| 9.155645392 |

or

I1 = 18.31115909∠-30.71364707°

I2 = 18.24644849∠-150.5940233°

I3 = 9.155964813∠89.52140479°
```

35 seconds on my old 15C (26 s + 9 s).

P.S.: When checking it upon the 48GX I noticed discrepancies in some results, starting in the third or fourth digits, plus one inconsistent angle (changed sign). It turns out that I had missed a negative sign in the beginning, when doing a polar to rectangular conversation. The element D(2.1) should be -110, not 110. Fixed.
This would cause less disturbance in the previous post example, only a few sign changes, which made the error harder to detect. A closer look to the angles values would have made the inconsistency evident, though. I will correct that as well.

EMAIL PM FIND QUOTE REPORT

3rd October, 2023, 09:47 **Post: #8**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Thank you, Gerson!
In the meantime I updated my post with a new, shorter and faster (for solving, at least) version.
It seems that every time I look at my code I find ways to improve it..

I also did some accuracy tests, with the example from pg 128 in the Advanced Functions Handbook, which you can now do on your original 15C:
(I multiplied all values by 3 - then they are all integers):

```
A 4x4:
  300    0    0    0
    0  3E6 -3E6    0
    0 -3E6  3E6    0
    0    0    0  3E5

B 4x4:
 -350  800    0    0
  800 -350    0    0
    0    0  442 -450
    0    0 -450  442

C 4x1:
 30
  0
  0
  0

D 4x1:
  0
  0
  0
  0
```

exact result, rounded to 10 digits:

```
 ( 1.995795141E-4 ,  4.096399085E-3)
 (-1.448833619E-3 , -3.563298308E-2)
 (-1.454083174E-3 , -3.563276083E-2)
 ( 5.344581171E-5 , -2.259868256E-6)
```

built-in result (on the 15CE, 8x8 matrix):

```
 ( 1.995820000E-4 ,  4.096401051E-3 )
 (-1.448812372E-3 , -3.563300015E-2 )
 (-1.454061929E-3 , -3.563277790E-2 )
 ( 5.344583732E-5 , -2.259863892E-6 )
```

split solve result:

```
 ( 1.995574520E-4 ,  4.096386294E-3 )
 (-1.448834528E-3 , -3.563298920E-2 )
 (-1.454084082E-3 , -3.563276695E-2 )
 ( 5.344583458E-5 , -2.259876193E-6 )
```

These results are comparable. The matrix is badly conditioned, so the results are only accurate to about 5 digits.
The split solve routine is slightly slower than the built-in one (without counting the necessary transformations to produce the 8x8 matrix, though!): 0.45 seconds vs. 0.36 seconds on the 15CE, but this latest version should be about 10% faster than the previous one.

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

3rd October, 2023, 16:28 (This post was last modified: 3rd October, 2023 16:41 by Gerson W. Barbosa.) **Post: #9**

Gerson W. Barbosa
Senior Member

Posts: 1,522
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Werner Wrote:** (3rd October, 2023 09:47)
> In the meantime I updated my post with a new, shorter and faster (for solving, at least) version.
> It seems that every time I look at my code I find ways to improve it..

Thanks again!

Now we have a really pocket calculator suitable for this kind of task, even the original HP-15C is fast enough to handle these.

It took about 30 seconds to solve my second example (20.5 s +9.5 s). Basically the same results.

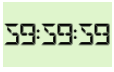
```
C =
| 15.7426645800 | [-1 ULP]
|-15.8956234800 | [+2 ULP]
| 0.07647945442 | [+8 ULP, previously 4 ULP]

D =
|-9.352382547 | [-5 ULP]
|-8.958908239 | [-7 ULP]
| 9.155645395 | [-4 ULP]

or

I₁ = 18.31115909∠-30.71364707° [-1, 0 ULP]

I₂ = 18.24644849∠-150.5940233° [-2, 0 ULP]

I₃ = 9.155964816∠89.52140479° [-3, 0 ULP, previously -6, 0 ULP]
```

ULP comparisons with the HP-GX results which are assumed to be more accurate, but not granted to be always exact to 12 digits.

Edited to fix formatting

EMAIL PM FIND QUOTE REPORT

5th October, 2023, 15:05 (This post was last modified: 6th October, 2023 13:28 by Werner.) **Post: #10**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Valentin Albillo Wrote:** (22nd September, 2023 22:22)
> The matrix **A+B** must be *invertible*, i.e. ***det(A+B) # 0***. For most real-life uses this will be the case but if this condition isn't met there are slight variations to this routine that would work Ok in those cases.

The problem is that the 15C *alters* a singular matrix slightly to produce a solution anyway, and you will probably not be aware that it did. I would've liked for this to be optional - but it has even found its way into the 42S as well...

Werner

EMAIL PM FIND QUOTE REPORT

6th October, 2023, 05:30 (This post was last modified: 6th October, 2023 06:45 by Valentin Albillo.) **Post: #11**

Valentin Albillo
Senior Member

Posts: 1,003
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hi all,

This is my ***1,001th*** post here so I'm celebrating it with a new, significant **improvement** **to the inversion routine** I included in my original post, but first a few ***comments*** are in order, as after a slow start this thread has been gathering momentum by the day. Let's see:

> **Werner Wrote:**
> You make it **increasingly harder** for me to **improve** upon **your code** ;-) I did manage to **squeeze off a punny byte** [...] The one thing I miss in your extensive posts, however, is some **explanation of the algorithm** used. [...] **Shamelessly ~~stealing~~ borrowing** your idea, we can solve systems of equations the same way:

Thanks for your interest, **Werner**, I hope you're enjoying your new **HP-15C CE** and do not regret your purchase at all. It certainly seems to have boosted your inspiration. As for the algorithms, as I said I leave that math-heavy discussion for the future article out of consideration for people who just want to use the code and don't care for its innards.

As for *squeezing off a puny byte*, well, that's your specialty. See if you can oblige in my new code below. And as for *stealing/~~borrowing~~* my idea, I appreciate your interest in my contributions and I don't resent you doing it at all, quite the contrary, thanks for it.

> **Werner Again Wrote:**
> **Not quite what you asked for**, Valentin, but here's a short routine to multiply two split complex matrices.

Very nice, and yes, I was expecting a simple piece of code that would use conventional *nested loops* to perform the multiplication by taking advantage of the *15C's native complex arithmetic*, that's why I left it as an easy exercise for the reader. As it wouldn't involve matrix *inversion*, it should run fast and accurately.

> **Werner Once More Wrote:**
> [Update: shaved off **2 bytes** ;-)] **Apologies** for **monopolizing** this thread ;-)] [...] to invert A+iB, do (GSB) **C** and (GSB) **E** [...] Routine **C** (Factor) - **21** bytes [...] Routine **E** (Invert) - **15** bytes

No need to apologize for monopolizing, I'm used to it by now. And congratulations for the *2-byte* savings, it's no mean feat and I know you enjoy doing it.

Per your figures, it seems you need to call *two* routines to *invert* a complex matrix, totalling *36 bytes* in all, which is nice taking into account that routine **C** is also used to solve a complex system if **C** is called first so **C** serves double duty. However, just *inversion* alone can be done by calling a *single 21-byte* routine, as I show below.

> **Gerson Wrote:**
> The matrix capabilities in the HP-15C were nice, but **the methods** in the manual for solving **complex** linear systems were somewhat **complicated** and **not easy to remember** [...] Doing it **programmatically** is a **great idea**, I wonder **why no one thought of this** back then [...]

Thanks a lot for your great *4x4 EE* examples for the original **HP-15C**, **Gerson**. The methods in the manual (both the ***OH*** and the ***AFH***) are wholly *inadequate*: extremely *cumbersome*, very *difficult* to remember when you needed them, outrageously *complicated* and requiring several *ad-hoc* pre- and post-transformations, but the worst of all was their enormous *inefficiency*, turning an ***NxN*** complex matrix into a ***2Nx2N*** real one, doubling the memory required and more than doubling the computing time. In short, a ***very poor*** performance by *HP* in the *OH*, and the *AFH* is no better in that regard.

And as for *"I wonder why no one thought of this back then"*, I **did**, but when the **HP-15C** was introduced *(1982)* I had alreay left ***PPC (***and ***PPC TN)*** so I had nowhere to submit any further articles, plus I didn't have the money for a *15C* or a *71B* or any other *HP* model with *complex/matrix* capabilities to implement my ideas, nor was I able to until 1985 and afterwards it took me several years to join the *MoHPC* forum and it's only now that the **15C CE** has appeared (with its *tremendous 180x* speed and *3x* the memory,) that I considered worthwhile and interesting to dust off my old notes, implement them and post the results here.

So here you are, the announced **improvement** ...

The improvement: Complex Matrix Inversion up to *8x8* in 21 bytes

The complex matrix inversion routine I included in my original post is nice and fast and all that jazz, and even quite short at **31 steps** *(32 bytes, 5 regs)*, but it's a bit conservative and thus somewhat ***longish*** and I can do better by selecting a different variant, as I'll explain right now.

The thing with *complex inversion* routines based on this specific *split*-matrix approach (**M** = **A** + i**B**) is that they require inverting some *real* matrices (say **A** or **B** or **A+B** or **A-B** or a *random* linear combination of **A** and **B**, etc.) and thus said matrices *must* be *invertible* (i.e. non-*singular* i.e. ***det#0***).

The good point is that we get to choose *which* matrix must be invertible, and if the complex matrix includes that component or combination and it happens to be singular, we can choose another that isn't, which would of course require a variant of the inversion routine.

Needless to say, all of this only applies if *complex* **M** *is* invertible even if *real* **A** or **B** (say) are *not*, but even if *both* are *singular* it might be the case that **M** is *not* and thus can be inverted using the particular routine which requires **A+B** (for instance) to be invertible, which is the ***31****-step* routine featured in my original post.

But that routine can be significantly *shortened* (by some **11 steps** no less, *33%* shorter) by assuming that **A** is invertible (which for real-life problems is usually the case and for random matrices is *always* the case,) i.e. ***det(A)#0***, and here's the resulting improved complex inversion routine (which, as detailed below, can *also* be used when **A** is *singular* but **B** is *not* by means of a simple trick):

Program listing

This ***20-step*** *(21-byte, 3-reg)* routine can be used to invert an ***NxN*** complex matrix **M** = **A** + i**B** when **A** is *invertible*, and also when **A** is *singular* but **B** is *invertible* using a simple trick, see details below.

```
LBL C           001- 42,21,13
RESULT A        002- 42,26,11
RCL MATRIX E    003- 45,16,15
RCL MATRIX B    004- 45,16,12
RCL MATRIX A    005- 45,16,11
1/x             006-       15
RESULT E        007- 42,26,15
 x              008-       20
RESULT A        009- 42,26,11
CHS             010-       16
RCL MATRIX B    011- 45,16,12
LASTX           012-    43 36
1/x             013-       15
R▼              014-       33
MATRIX 6        015- 42,16, 6
1/x             016-       15
RESULT B        017- 42,26,12
X<>Y            018-       34
 x              019-       20
RTN             020-    43 32
```

Changes in the documentation

I could have edited the docs in my original post to cater for this new routine but that's not my personal policy; unlike most people I only edit a post to correct *typos* or blatant *errors*, never to change, remove or add significant contents. Besides, the original documentation fully applies to my original **31-step** routine which is the one to use if *both* **A** and **B** are *singular* but **A+B** is not, so its docs will remain as they currently are.

What I'll do here is to specify the *changes* in the docs pertaining to this new **20-step** routine, including for context the abridged paragraphs where they appear, with the changed parts in **bold red**. You should still refer to the original docs for everything else (***Notes***, ***Requirements***, ***Worked Example***, etc.)

Changes in *Notes*:

- It runs *sequentially* from its first to its last step, executing each *just once*, which means it executes *exactly* ***20*** *user-code instructions in all* (not hundreds or thousands like other approaches,) so it runs very fast.

  Comment: As ***11*** *fewer* instructions are executed in the new routine *(33% less,)* it runs somewhat faster.

Changes in *Requirements*:

- The *maximum* size ***NxN*** complex matrix **M** you can invert depends on the memory available in your physical or virtual device, as per this table:

M	A,B,E	Regs	+Prog	Comments
1x1	1x1	3	**6**	-
2x2	2x2	12	**15**	-
3x3	3x3	27	**30**	-
4x4	4x4	48	**51**	Max. size w/ ***15C/64*** **but see (*)**
5x5	5x5	75	**78**	ditto ***CE/96***
6x6	6x6	108	**111**	-
7x7	7x7	147	**150**	ditto ***CE/192*** **but see (**)**
8x8	8x8	192	**195**	ditto ***DM15/M1B***

  so e.g. if you want to invert an *8x8* complex matrix you need **195** *regs* available, which includes matrices **A**, **B** (and **E**,) plus **3** *regs* to hold the routine itself.

  Comment: The new routine is only **21** bytes long, so it occupies *exactly* **3** *regs* of program memory. As the initial routine occupied **5** *regs*, this means that now **2** extra *regs* are available for additional data or up to **14** additional program steps.

- The matrix **A** must be *invertible*, i.e. ***det(A)#0***. For most real-life uses this will be the case but if this condition isn't met there are slight variations to this routine that would work Ok in those cases, namely for any of the following invertible matrices: **B**, **A+B** (dealt with in my original post in this thread,) **A-B**, etc.

Changes in *(*) Observation re the original HP-15C and 4x4 complex matrices:*

- The original **HP-15C** could invert a ***4x4*** complex matrix but it took all *64 regs* available and it was a complicated, completely *manual* process as there wasn't *any* memory left for program code. On the other hand, running my routine is a fast, *automated* process that leaves out all the drudgery (transformations, etc.) and still leaves **13** *regs* (up to **91** extra program steps) free for additional code or data. [...]

- Call my *complex inversion* routine. When it ends, the complex inverse is stored in **A**, **B** and there's still **13** *regs* free. [...]

- Get rid of auxiliary matrix **E** (redimension it to ***0x0***.) This frees another *16 regs*, so there's now **29** *regs* available for what follows.

  > **Comment**: There's now **29** *regs* still available in the *original* **HP-15C** after inverting a *4x4* complex matrix, which can be used to store further data (e.g. the *constant* matrix and the *solution* matrix, when *programmatically* solving a *4x4* system of complex equations (ignore the abstruse *manual* methods featured in the *OH* and *AFH*) or up to **203** program steps for additional code (e.g. your own program which uses the inversion routine,) or a combination thereof.

- Dimension both the *constant* matrix (say **C**) and the *solution* matrix (say **D**) to be *4x2* and populate the constant matrix. This will leave **13** *regs* (i.e. as many as **91** program steps) still available for the *matrix-multiplication* code, which is left as a fairly easy exercise for the reader (just a loop which multiplies each row of the *inverse* by the *constant* matrix using **the HP-15C's native** *complex* arithmetic.)

Changes in *(*) Observation re the HP-15C CE and 8x8 complex matrices*:

- Though there's not enough room in the **HP-15C CE** in *192-regs Mode* to invert an ***8x8*** complex matrix **M** by running the routine featured here (**195** *regs* would be needed; the routine itself wouldn't fit) there's just enough room for the split matrices **A**, **B** and the auxiliary matrix **E** (*192 regs* in all,) so the *8x8* complex matrix **can** be inverted in a pinch if the user executes the **18** program instructions *manually* in sequential order. The procedure would be like this: [...]

  - Carefully execute *manually* the **18** instructions from ***002* RESULT A** to ***019* x** in sequential order. Assuming you're reasonably proficient with the *HP-15C*, this should take **2 min** or less [...]

    > **Comment**: As now there's only **18** instructions to manually execute instead of the *29* required by the original routine, this makes the process all the more expeditious, less tiring to perform and much more likely to be completed without error.
    >
    > Also, it's a pity that the *index registers* **R0**, **R1** and **RI** are *permanent* and can't be allocated to the common pool. If it were possible, this inversion routine (which *doesn't* use the index registers,) could be stored in the *3 regs (21 bytes)* they would provide, thus allowing for inverting ***8x8*** complex matrices *programmatically*. Alas, no such luck, *HP* didn't see fit to allow for the possibility. 😕

Using this routine when A is *singular* but B is not

As I explained above, usually different variants of my *complex inversion* routine are needed depending on whether *(1)* **A** is invertible, or *(2)* **A** is singular but **B** is invertible, or *(3)* both **A** *and* **B** are singular but **A+B** (or **A-B**) is invertible.

My original **32**-byte routine somewhat conservatively addressed the *third* case, where **A** and **B** can *both* be singular as long as **A+B** isn't (a slightly-different, same-length variant would handle the invertible **A-B** case.)

Now, my new **21**-byte routine caters for the *first* case (**A** is invertible.) The *second* case (**A** is *singular* but **B** is invertible) can be dealt with using a near-symmetric same-length variant of the routine but actually there's no need to have *two* almost-identical routines in memory at once, as a simple trick will allow the present one to also handle the *second* case (**A** singular, **B** invertible), just proceed like this:

To compute using this routine the complex inverse of **M** = **A** + i**B** when **A** is singular but **B** is invertible:

1. Initialize and dimension matrices **A**, **B** as described in my *OP*'s ***Worked example***.

2. ***Swap*** *the roles* of **A** and **B**, i.e. store **M**'s *real* parts in **B** and the *imaginary* parts in **A**.

3. Compute the *complex inverse* of this *"swapped"* matrix: **GSB C**

4. Now ***swap back*** the contents of **A** and **B** while also changing the signs of their elements on the fly. From the keyboard, execute:

```
RCL MATRIX A        { swap back A and B using auxiliary matrix E ... }
CHS                 { ... while also changing their signs at the same time }
STO MATRIX E        { done; matrix E now holds the negated contents of A }

RCL MATRIX B        { now matrix B is recalled to the X stack register ... }
CHS                 { ... its contents are negated ... }
STO MATRIX A        { ... and stored back in A }

RCL MATRIX E        { finally, the negated contents of A held in matrix E ... }
STO MATRIX B        { ... are stored back in B and the swap is completed. }
```

   Now **A** contains the *real* parts of **M**'s true *complex inverse*, **M'**, and **B** contains the *imaginary* parts, as desired.

   > **Note**: Of course, all those 8 instructions executed manually can be included as program lines to be executed programmatically after calling my routine, in case **A** in **singular** but **B** is not. Using the *ad-hoc* variant of my routine for such case wouldn't need those 8 steps but then you'd have to

include said routine's 20 steps as well. Oh, and there's a trick to swap the contents of two *NxN* matrices *without* needing a third auxiliary one.

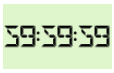
That's all. I've missed a few *usual suspects* posting here but that's life, no hard feelings. As always, hope you enjoyed it.

V.

Edit: a few typos.

PM WWW FIND EDIT QUOTE REPORT

6th October, 2023, 13:23 **Post: #12**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Valentin Albillo Wrote:** (6th October, 2023 05:30)
> (..) See if you can oblige in my new code below

I have to live up to my reputation, don't I?

18 lines, 19 bytes:

```
001 LBL C
002 RCL MATRIX E
003 RCL MATRIX B
004 RCL MATRIX B
005 RCL MATRIX A
006 RESULT E
007 /
008 CHS
009 RCL MATRIX A
010 RESULT A
011 1/x
012 1/x
013 Rv
014 MATRIX 6
015 1/x
016 RESULT B
017 x
018 RTN
```

Personally, however, I prefer the previous, slightly longer code. It offers more protection against degenerate systems, especially since, as I mentioned before, the 15C does not alert you when it encounters one, but alters it slightly so as to be able to come up with *a* solution - even if it is only one of infinitely many.

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

6th October, 2023, 16:51 (This post was last modified: 6th October, 2023 17:05 by J-F Garnier.) **Post: #13**

J-F Garnier
Senior Member

Posts: 848
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

I'm amazed by these little pieces of code posted by Valentin (congrats for your 1000 - now 1001 - posts !) and Werner.

I really appreciate short, powerful routines that expand the possibility of a machine.
And this is exactly the case here: the first thread from Valentin was providing a workaround for the 8x8 limit for real matrix inversion.

Then this second thread goes further by getting the utmost from the limited memory of the HP-15c in the case of complex matrices.
In the case of the 15c CE/192, this makes a difference and allows to manage 7x7 matrices instead of 6x6. An hypothetical 15c CE/224 could even handle 8x8 matrices.

J-F

EMAIL PM WWW FIND QUOTE REPORT

6th October, 2023, 19:37 **Post: #14**

Gerson W. Barbosa
Senior Member

Posts: 1,522
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Werner Wrote:** (29th September, 2023 15:20)
> So, to solve a single 4x4 system, you'd need 3.4^2+2.4 = 56 registers. Since the program of 48 bytes uses 7 additional registers, it still fits in an original 15C.

I've tried a 4x4 system on mine but it would stop with an Error 10 message on the display. MEM shows 1 15 9-0. Not a problem for the HP-15C CE (MEM: 19 25 09-9 after program completion).

Problem is both det(A) and det(B) in that particular system are zero. It does give a solution, but it looks meaningless to me:

```
A =
| 12.1 0.0 0.0 12.10 |
| 0.0 12.1 0.0 12.10 |
| 0.0 0.0 24.2 24.20 |
| 12.1 12.1 24.2 48.4|

B =
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |
| 0 0 0 0 |

C =
| 220 |
|-110 |
|-110 |
| 0.0 |

D =
| 0.000000000 |
| 190.5255888 |
|-190.5255888 |
| 0.000000000 |

=>

C =
| 5.425143454 |
| 13.47770233 |
|-24.62741542 |
|-5.425143453 |

D =
| 4.514866980 |
| 34.56742586 |
|-3.537691900 |
|-4.514866980 |

or

I₁ = 7.058059602∠39.76761789°

I₂ = 37.10195939∠68.69938684°

I₃ = 24.88021009∠-171.8254666°

I₄ = 7.058059601∠-140.2323821°
```

These may be a valid solution (I haven't checked them out), but they don't appear to be the solution for the problem I was trying to solve.

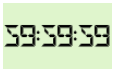
In an attempt to circumvent that I changed slightly the last element of matrix A, which managed to give results very close to the values I expected. I still have to doublecheck my equations for a possible mistake.

```
A =
| 12.1 0.0 0.0 12.100 |
| 0.0 12.1 0.0 12.100 |
| 0.0 0.0 24.2 24.200 |
```

```
| 12.1 12.1 24.2 48.39|


=>

C =
| 18.181816210000000 |
|-9.090911560000000 |
|-4.545456889000000 |
| 0.000002180000421 |

D =
|-0.000004626165054 |
| 15.745911680000000 |
|-7.872962425000000 |
| 0.000004820000930 |

or

I1 = 18.18181621∠-0.000014578°

I2 = 18.18181530∠120.0000142°

I3 = 9.090913908∠-119.9999995°

I4 = 0.00000529007∠65.66360913°

and

Ir1 = I1 + I4 = 18.18181839∠0.0000006111°

Ir2 = I2 + I4 = 18.18181838∠120.0000007°

Ir3 = I3 + I4 = 9.09090864376 ∠-120.00000279°
```

EMAIL PM FIND QUOTE REPORT

6th October, 2023, 21:38 **Post: #15**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hi Gerson,
To solve a 4x4 system on an original 15C, you have to leave off the inversion code LBL E. I see you used up 9 registers for programs, so you used the full 63 bytes and then there's not enough room left to solve the system. LBL C and D together only take 48 bytes so less than 7 registers, and then it will work (with 1 DIM (i)).
And yes, when A+B is singular you get a meaningless answer, unfortunately. The example you gave however, is rank-deficient anyway as B=0 and det(A)=0, so in this case there is no workaround, save a Least Squares solution, which is a different beast altogether.
Cheers, Werner

EMAIL PM FIND QUOTE REPORT

6th October, 2023, 22:20 (This post was last modified: 7th October, 2023 13:26 by Gerson W. Barbosa.) **Post: #16**

Gerson W. Barbosa
Senior Member

Posts: 1,522
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

> **Werner Wrote:** (6th October, 2023 21:38)
> And yes, when A+B is singular you get a meaningless answer, unfortunately. The example you gave however, is rank-deficient anyway as B=0 and det(A)=0, so in this case there is no workaround, save a Least Squares solution, which is a different beast altogether.

Hello, Werner,

Thanks for the enlightenment!
Anyway, I was able to get answers very close to the exact results, currents equal to 200/11 A and 100/11 A, shifted 120 degrees from each other. In this latest example instead of a wye I considered a delta three-phase voltage source. As a result, rather than a simple 3x3 system that could be solved by hand, I ended up with a more complicated 4x4 system. If not for anything else, at least another opportunity to use the new VW complex equation solver :-)

Best regards,

Gerson.

P.S.: I made indeed a mistake in the previous example by introducing an extra unnecessary equation.
Time for solving a 3x3 system on the original 15C is 32 seconds, not 30 as I said earlier; 56 second for a 4X4 system.
Instantly on the 15C CE.

EMAIL PM FIND QUOTE REPORT

11th October, 2023, 02:46 **Post: #17**

Valentin Albillo
Senior Member

Posts: 1,003
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hi, **all**,

This thread's got a number of new posts so some comments ...

> **First Werner "2-byte" Proudly Wrote:**
> I have to live up to my reputation, **don't I?**

Indeed you did, shaving off 2 bytes from my original code once again. That's the third or fourth time in a row so I think you've earned yourself the *"2-byte"* moniker.

Some remarks on your *18-step* code vs. my *20-step* routine:

- Your use of **[÷]** at step *007* plus **[1/x]** at steps *011* and *012* means that matrix **A** gets processed ***three*** times: *(1)* converting it to an *LU*-decomposed form, *(2)* then converting that *LU* form to **A**'s *inverse*, and *(3)* finally inverting **A**'s inverse to get back **A** (possibly with some negligible rounding errors) for further use.

  *However, I'm not sure if **[÷]** (which returns the result of $X^{-1}Y$,) does indeed compute the inverse (of **A**) before the multiplication or, as I suspect, it simply computes **A**'s **LU** form and then uses special multiplication rules as in the case of **MATRIX 5**, which computes Y^TX without actually transposing **Y**.*

- On the other hand, my original code only subjects matrix **A** to ***two*** inversion procedures instead of your three procedures detailed above, so I feel that your code's accuracy might be slightly *degraded* (whether the matrix is degenerate or not,) and I also wonder if your routine's speed is negatively affected. **Have you checked any of this ?**

Well, I have ! These are the results:

- As for **speed**, using **J-F**'s *5x5* complex matrix featured in my *OP (Worked example)* and calling our respective inversion routines *50* times in a loop I get these average timings:

  - my 20-step routine : **0.89"** per call
  - your 18-step routine: **0.91"** per call

  so mine is negligibly faster *(~2.2%)*.

- And as for **accuracy**, I've checked it (again using **J-F**'s handy *5x5* example) by inverting the complex matrix *twice* in succession so that the original matrix ought to be returned if the inversion procedures were exact, then comparing each returned matrix **A'**, **B'** vs. their respective originals **A**, **B** (conveniently saved to **C**, **D** because the inversion is done *in-place* and replaces the originals.)

  The comparison consists simply in subtracting the returned matrices from the saved originals (which if the inversions were error-free would result in the *zero* matrix,) and then computing the ***Row Norm*** **(MATRIX 7)** and the ***Frobenius Norm*** **(MATRIX 8)** of the respective **A' - A *(C)*** and **B' - B *(D)*** subtractions. The results are as follows:

GSB C, GSBC, RESULT **E**,	**Mine**	**Yours**
RCL MATRIX **A**, RCL MATRIX **C**, -, MATRIX 7 ->	*3.8753E-8*	*2.9381E-8*
LASTX, MATRIX 8 ->	*3.0716E-8*	*2.2004E-8*
RCL MATRIX **B**, RCL MATRIX **D**, -, MATRIX 7 ->	*4.3000E-8*	*4.4401E-8*
LASTX, MATRIX 8 ->	*3.1621E-8*	*3.6484E-8*

  so you see, the differences are again very minor. Your errors are slightly better for matrix **A** while mine are slightly better for matrix **B**, no big deal. And remember, these are the errors after inverting the complex matrix *twice*, the errors for a single inversion should be significantly smaller.

Then J-F Garnier Wrote:

I'm amazed by these little pieces of code posted by Valentin (**congrats for your 1000 - now 1001 - posts !**) and Werner.

Thanks, J-F. I was missing your comments.

Quote:

I really appreciate **short, powerful routines** that expand the possibility of a machine. And this is exactly the case here: the first thread from Valentin was providing a workaround for the **8x8 limit** for real matrix inversion.

With the **HP-15C** original and *CE* editions not having any sort of mass storage, it's vital that the programs are *as short as possible* lest no one will bother to key them in, not even to just try them out, as it would be such a lengthy, error-prone chore. And something had to be done about that annoying *8x8 limit*, so *partitioned matrices* immediately sprang to mind.

Quote:

Then this second thread goes further by getting the utmost from the limited memory of the HP-15c in the case of complex matrices. In the case of the 15c CE/192, this makes a difference and **allows to manage 7x7 matrices** instead of 6x6. An **hypothetical** 15c CE/224 could even handle 8x8 matrices.

On the other hand, a *not-so-hypothetical-but-actually-extant* **DM15** *clone* with firmware *M1B (229 regs)* can handle *8x8* complex matrices (and perhaps even *systems* of 8 complex equations in as many complex unknowns.) And if the *CE/192* user is not afraid of manually *typing in* just *16* or *18* instructions, he/she can invert an *8x8* complex matrix in the *CE/192* as well, it'll take just a little care and likely less than *2 min*.

BTW, it's a real pity *HP* didn't make the index registers **R0**, **R1** and **RI** *allocatable* because that would've left just enough program space for the complex inversion routine itself so no need for manually typing in anything.

It's also a pity that ***Moravia*** didn't allow for the max. *229 registers* in the *CE*, as ***SwissMicros*** did. The *37* extra regs might come in handy for everything, *8x8* complex matrix handling in particular.

Perhaps this capability was intentionally left for a future **HP-15C CE 2.0** version including

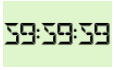
- that many registers *(229, or at least 224,)*
- a cable and a *Connectivity Kit* of sorts for mass storage, backups, outputting results, easy and convenient program/data editing, sharing progs/data with an ...
- ... also-included software emulator for *Windows/iOS/Android*,
- a patched firmware which corrects the bug in the "nut" emulation layer that fatally broke the embedded *HP-16C* and might also affect the *HP-15C ROM* code in as-yet-unknown ways,
- a printed color copy of the *"Advanced Functions Handbook"*,
- oh, and a *"**Thank you** again !"* card.

That way they'd easily milk yet another 130 € apiece from each of us ... 😀

Best regards.
V.

PM WWW FIND EDIT QUOTE REPORT

11th October, 2023, 17:11 **Post: #18**



Werner
Senior Member

Posts: 813
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hi Valentin.

Valentin Albillo Wrote: (11th October, 2023 02:46)

[..] my original code only subjects matrix **A** to ***two*** inversion procedures instead of your three procedures detailed above, so I feel that your code's accuracy might be slightly *degraded* (whether the matrix is degenerate or not,) and I also wonder if your routine's speed is negatively affected. **Have you checked any of this ?**

I didn't run simulations because I knew that the results would be comparable in accuracy and timing.

- timing:

  the operations count is actually similar, if not the same, thanks to the 15C's ability to use a matrix' LU-decomposition form in subsequent solve and inverse operations (which the 42S lacks btw). So while I do three operations (solve, and two inversions), they take the same time as two inversions and a multiply:
  WH:
  . solve(A*E=B) = LU + n*solve(1). LU-decomp has an operations count of n^3/3 + O(n^2), 1 solve takes n^2 operations, so n solves take n^3 operations
  . inversion of an LU-decomposed matrix has an operation count of 2/3*n^3 + O(n^2)

. full inversion is n^3 + O(n^2) operations
total operation count is thus 3*n^3 + O(n^2)
VA: 2 full inversions of n^3 each, one matrix-matrix multiplication, also n^3, so total is also 3*n^3 + O(n^2) ...
Operation count however does not tell the whole story, and it is true that a matrix multiply is more efficient than n solves - even if both have the
same operation count. That's probably where the 2% difference comes from?
I ran a test inverting the matrix 100 times, and could hardly see a second's difference in timing (and quite a bit of fluctuation btw)

Additional remarks:
- things are different when solving a single right-hand side of course. Then solving is about 3 times as fast as multiplying by the inverse
- I dimly remember LINPACK switching to multiplying by the inverse when the number of right-hand sides grew very large; I can't find it in today's LAPACK any more though. Both are to be taken with a grain of salt btw: it's not because I *seem* to remember it that it was so, and it's not because I don't find it that it is not there..

- accuracy

  Solving is more accurate than pre-multiplying by the inverse. Somewhere in the LAPACK Working Notes is an example where the solve routine has an error 5 orders of magnitude smaller than multiplying by the inverse. They also admit these cases are hard to find, but in general solving is somewhat more accurate.
  I compared solving A*E=B to pre-multiplying B by inv(A). FNRM of the difference between the calculated result and the exact result was almost twice as large for the latter. However, the split inversion routine inverts A twice (both mine and yours), before adding it to the matrix obtained by either solving or pre-multiplying by the inverse - so possible accuracy gains made by solving i.o. inverting will be completely lost anyway.
- solving

> **Quote:**
>
> *However, I'm not sure if **[÷]** (which returns the result of **$X^{-1}Y$**,) does indeed compute the inverse (of **A**) before the multiplication or, as I suspect, it simply computes **A**'s **LU** form and then uses special multiplication rules as in the case of **MATRIX 5**, which computes **Y^TX** without actually transposing **Y**.*

Those 'special multiplication rules' are called elimination and substitution.
(omitting the permutations) LU decomposition factors A into L*U, with L a lower triangular matrix with 1's on its diagonal, and an upper triangular matrix U.
A*x=b is then solved in two steps as
L*y=b and
U*x=y

e.g. the example of a 3x3 upper triangular matrix:

```
u11 u12 u13   x1   y1
    u22 u23 * x2 = y2
        u33   x3   y3
```

is solved easily as follows, in order of operations:

```
x3 := y3/u33;
x2 := (y2 - u23*x3)/u22;
x1 := (y1 - u12*x2 - u13*x3)/u11;
```

this is called 'back substitution'.
The lower unit triangular system is solved in a similar manner, and is called 'elimination' because the process is identical to the one used to create the LU-decomp in the first place.
(Have a look at my 'classical solve' here, which performs these operations, the same (albeit in a different order) as the 15C does internally, but without the benefit of 12-digit extended precision.
I should update that post, as the current version is more than 30 bytes shorter btw)

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

11th October, 2023, 18:54 **Post: #19**

Maximilian Hohmann
Senior Member

Posts: 1,151
Joined: Dec 2013

RE: [VA] SRC #015b - HP-15C & clones: COMPLEX Matrix Inverse up to 8x8

Hello!

> **Valentin Albillo Wrote:** (11th October, 2023 02:46)
>
> That way they'd easily milk yet another 130 € apiece from each of us ...

I can guarantee that the next one is going to cost more than 130 Euros, especially if it comes with a printed colour book! Books like that now tend to cost in excess of 30 Euros, especially if they are made in small numbers. And they are not getting cheaper.

But most important, thanks for the effort of sharing your program(s) with us - and Werner of course as well! This really makes a lot more sense than the method from the original manual. If anything, I would want these routines hardwired into the ROM of the next version of the 15C!

Regards
Max

Forum software: MyBB, © 2002-2023 MyBB Group.