MINV – N×N Matrix Inversion

© 2020 Valentín Albillo

Abstract

MINV is a program written in 1980 for the HP-41C programmable calculator and compatibles to quickly invert a real N×N matrix using an interchange method. One worked example included.

Keywords: matrix inversion, interchange method, RPN, programmable calculator, HP-41C, HP-41CV, HP-41CX

1. Introduction

MINV is a 170-step RPN program that I wrote in 1980 for the HP-41C programmable calculator (will also run as-is in the HP-41CV/CX) to compute the inverse of a real N×N matrix, where N ranges from 1 up to 16, using a non-gaussian interchange method. It has been optimized to be short (40 program registers) without loss of convenience, and fast (inverting a 16x16 matrix on the original, physical HP-41C takes about 36'). It does use synthetic programming1 to help accomplish both goals.

The program is written so that zero pivots will cause no trouble, they’re skipped and the following pivot is tested. The locations of all zero pivots are remembered and their corresponding interchanges are performed later, which avoids most problems when dealing with inconvenient matrices without having to manually rearrange them. There is one bad case, however, when all the pivots in the main diagonal are zero, in which case the program stops showing a program-generated error message. This is a rare case but can happen.

The method used is an interchange method: consider the system $A \cdot x = b$, which has the same matrix $A$ we’re trying to invert. The vectors $b$ and $x$ have $N$ components each. The method interchanges a component of $b$ with a component of $x$ at a time. After $N$ independent interchanges have been performed, the roles of $b$ and $x$ are reversed and the system becomes $A^T \cdot b = x$, where $A^T$ is the inverse of $A$. The algorithm in pseudo-code is:

```
FOR k = 1 to N
  LET a_kk = 1/a_kk
  LET a_ik = a_ik . a_kk , i = 1, 2, ..., N, i ≠ k
  LET a_kj = -a_kj . a_kk , j = 1, 2, ..., N, j ≠ k
  LET a_ijk = a_ijk - a_ik . a_kj . a_kk , i = 1, 2, ..., N, i ≠ k, j = 1, 2, ..., N, j ≠ k
```

A special procedure takes place if $a_{kk} = 0$: $k$ is incremented by 1 and flagged so that it will be remembered as a pending interchange to be performed later. After a successful interchange is accomplished a search takes place for the minimum $k$ which is still pending. If no such $k$ is found, the work is completed. If no interchange is successful an error condition is generated. That only happens if all remaining pivots in the diagonal are zero.

All inputs and outputs are labeled and there’s a warning to prevent a memory allocation error (SIZE).

1 Synthetic instructions using status registers M, N, O and d are used to save three storage registers (so that N×N matrices can be inverted on a bare-bones HP-41C) and to restore the status of all flags upon finishing. The program uses flags 00 through N-I but this is not apparent to the user because the status of all flags is saved at the beginning of program execution and restored afterwards before the program stops. This is accomplished by using synthetic instructions STO d and RCL d, and thus the user has all flags available except flag 19, which is used by the program to keep track of input/output but it’s not restored at the end of program execution. Specifically, the program uses the following synthetic instructions: STO/RCL d, STO/RCL M, STO N, ST+ N, ST- N, STO O, ST+ O, RCL IND N and RCL IND O.
2. Program Listing

```
01 LBL "MI" 35 "+" 69 RCL 05 103 * 137 RCL 01 -170 program steps
02 FIX 0 36 ARCL 03 70 * 104 ST- IND 06 138 RCL 03 (40 prog. regs.)
03 CF 29 37 "-" 71 + 105 LBL 06 139 X=Y?
04 CF 19 38 FIX 4 72 7 106 1 140 GOTO 00 ♦
05 "N=?" 39 FS? 19 73 STO 06 107 ST+ 06 141 RCL 04 - uses R00–R06 as
06 PROMPT 40 ARCL IND 04 74 STO 0 108 ST+ N 142 ST* IND 06 auxiliary registers,
07 STO 05 41 FC? 19 75 + 109 ISG 03 143 CHS plus the ones used
08 X↑2 42 "-?" 76 RCL IND X 110 GTO 04 ♦ 144 ST* IND 02 to store the matrix
09 6 43 PROMPT 77 X=0? 111 RCL 05 145 LBL 00 ♦
10 + 44 FC? 19 78 GTO 00 ♦ 112 ST- N 146 RCL 05 - uses status registers
11 SF 25 45 STO IND 04 79 1/X 113 LBL 02 147 ST+ 06 ♦
12 RCL IND X 46 ISG 04 80 STO IND Y 114 ST+ 0 148 ISG 02 - uses flags 00–19
13 FS?C 25 47 X<>Y 81 STO 04 115 RCL 00 149 X<>Y ♦
14 GTO 99 ♦ 48 ISG 03 82 X<>Y 116 STO 0 150 ISG 03 - sets FIX 4
15 1 49 GTO 10 ♦ 83 RCL 01 117 ISG 02 151 GTO 00 ♦
16 + 50 RCL 00 84 ST+ O 118 GTO 04 ♦ 152 SF IND 01
17 "SIZE " 51 STO 03 85 - 119 GTO 14 ♦ 153 RCL 00
18 ARCL X 52 ISG 02 86 RCL 00 120 LBL 07 154 STO 01
19 PROMPT 53 GTO 10 ♦ 87 STO 03 121 RCL 05 155 LBL 13
20 LBL 09 54 FS?C 19 88 STO 02 122 ST+ 06 156 FC? IND 01
21 7 55 RTN 89 + 123 GTO 02 ♦ 157 GTO 91 ♦
22 STO 04 56 1.001 90 STO N 124 LBL 14 158 ISG 01
23 RCL 05 57 - 91 LBL 04 125 STO 06 159 GTO 13 ♦
24 1 R3 58 STO 00 92 RCL 02 126 RCL 05 160 RCL M
25 / 59 STO 01 93 RCL 01 127 ST* 06 161 STO d
26 1 60 RCL d 94 X=Y? 128 RCL 01 162 SF 19
27 + 61 STO M 95 GTO 07 ♦ 129 ST+ 06 163 GTO 99 ♦
28 STO 00 62 0 96 RCL 03 130 * 164 LBL 90
29 STO 02 63 STO d 97 X=Y? 131 + 165 ISG 01
30 STO 03 64 LBL 91 98 GTO 06 ♦ 132 7 166 GTO 91 ♦
31 LBL 10 65 FS? IND 01 99 RCL IND N 133 ST+ 06 167 RCL M
32 FIX 0 66 GTO 90 ♦ 100 RCL IND O 134 + 168 STO d
33 "A" 67 RCL 01 101 * 135 STO 02 169 "ERROR"
34 ARCL 02 68 RCL 01 102 RCL 04 136 LBL 00 170 PROMPT
171 END.
```

The symbols ♦ and ♦ are purely cosmetic, to indicate branching: + is the Append alpha function.

2.1 Program characteristics

- This program is 170-step long (40 program registers if you let the final end of program memory .END. act as the end of this program) and requires SIZE N²+7 to invert an NxN matrix.

- This program is way faster than the "MATRIX" program in the MATH 1A ROM module. Comparitive times are:

```
<table>
<thead>
<tr>
<th></th>
<th>1x1</th>
<th>3x3</th>
<th>5x5</th>
<th>10x10</th>
<th>15x15</th>
<th>16x16</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATRIX</td>
<td>10&quot;</td>
<td>56&quot;</td>
<td>2'49&quot;</td>
<td>15’32&quot;</td>
<td>(45’)</td>
<td>(54’)</td>
</tr>
<tr>
<td>MINV</td>
<td>4&quot;</td>
<td>20&quot;</td>
<td>1'16&quot;</td>
<td>9'12&quot;</td>
<td>30’</td>
<td>36’</td>
</tr>
</tbody>
</table>
```

The times in parentheses are extrapolations, as the "MATRIX" program is limited to ≤14x14 matrices.

- The maximum matrix size depends on the number of RAM modules plugged in, as follows:

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NxN</td>
<td>up to 4x4</td>
<td>up to 8x8</td>
<td>up to 12x12</td>
<td>up to 14x14</td>
<td>up to 16x16</td>
<td></td>
</tr>
<tr>
<td># modules</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Usage Instructions
To find the inverse matrix of a real N\times N matrix \( A \), follow these instructions:

In RUN Mode, execute the program:

\[ \text{XEQ} \text{“MI”} \rightarrow N=? \quad \{ \text{asks for the size of the matrix} \} \]
\[ \text{N} \text{R/S} \rightarrow A_{i,j}=? \quad \{ \text{asks in turn for each of the matrix elements, left to right, top to bottom} \} \]
\[ a_{ij} \text{R/S} \rightarrow ... \]
\[ a_{NN} \text{R/S} \rightarrow ... \quad \{ \text{the program will now invert the matrix. The flag annunciators will turn on as the associated interchanges are performed. Once all interchanges are over the inverse matrix elements are output ...} \} \]
\[ \rightarrow A11=(\text{first element of the inverse matrix}) \]
\[ \text{R/S} \rightarrow A12=(\text{second element}) \]
\[ ... \]
\[ \text{R/S} \rightarrow A_{NN}=(\text{last element}) \]
\[ \text{R/S} \rightarrow \text{program ends} \]

To invert another matrix, repeat the procedure above; to invert back the just inverted matrix, press \( \text{R/S} \).

Notes:

- if after introducing \( N \) a message \textit{SIZE nnn} does show up this means the current register allocation is insufficient to run the program so simply execute \textit{SIZE nnn} as directed and then press \( \text{R/S} \) to resume. In general, if the matrix dimension is \( N \times N \), a minimum of \( N^2+7 \) registers are required.

- the inverted matrix replaces the original one in the storage registers; to reinvert the inverse matrix (to check accuracy, for instance) simply press \( \text{R/S} \) and once the inversion procedure is completed you should get the original matrix back (ignoring suitably small rounding errors); if not, the original matrix might be \textit{ill-conditioned} or nearly \textit{singular}. A singular matrix has determinant 0 and no inverse.

- if all pivots are zero along the main diagonal, the program stops with \textit{ERROR} in the display (after restoring all flags). This might indicate that the matrix is singular, but not necessarily. However, this happens very rarely.

- the program isn’t adapted to run with a printer attached, as it uses \textit{PROMPT} instead of \textit{AVIEW}, so pressing \( \text{R/S} \) is necessary in order to output the elements of the inverse. This may be easily changed in the program listing above if desired but remember that the printer slows down execution speed significantly while computing the inverse (let alone while actually printing.)

- flag \textit{19} is used (but \textit{not} restored) to control input/output so don’t turn off the calculator while I/O is taking place.

- The storage and status registers are used as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>...</th>
<th>(N^2+6)</th>
<th>(M)</th>
<th>(N)</th>
<th>(O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>0.00[(N\text{-1})]</td>
<td>(k)</td>
<td>(i)</td>
<td>(j)</td>
<td>pivot</td>
<td>(N)</td>
<td>aux.</td>
<td>(a_{11})</td>
<td>...</td>
<td>(a_{NN})</td>
<td>\textit{flags}</td>
<td>aux.</td>
<td>aux.</td>
</tr>
</tbody>
</table>
4. Examples

The following example can be useful to check that the program is correctly entered and to understand its usage.

4.1 Example

Invert the following 4x4 matrix.

\[
A = \begin{pmatrix}
2 & 2 & 3 & 2 \\
2 & 2 & 3 & 1 \\
11 & 5 & 4 & 6 \\
2 & 1 & 1 & -9
\end{pmatrix}
\]

Now the program starts to compute the inverse. Watch the flag annunciators: the 0 annunciator is on, as the first pivot is \(a_{11} = 2 \neq 0\), so the first interchange is done. However, the next annunciator that turns on is the 2 annunciator. This is because the next pivot, \(a_{22}\), is 0, so it’s skipped and the next pivot is considered, namely \(a_{33}\), which is not 0 and the interchange is performed.

After that, \(a_{22}\) is checked again but it’s still 0 so it’s skipped once more and next \(a_{44}\) is tried, which isn’t 0 and the interchange is performed.

Finally, \(a_{22}\) is checked for the third time and now it happens to be non-zero so the last pending interchange is performed and, as no pending interchanges remain, the work is done, the flags are restored and the elements of the inverse matrix are output:

\[
\begin{array}{c}
\text{...} \\
\text{R/S} & \rightarrow A1,1=70.0000 \\
\text{R/S} & \rightarrow A1,2=-71.0000 \\
\text{R/S} & \rightarrow A1,3=-1.0000 \\
\text{R/S} & \rightarrow A1,4=7.000000
\end{array}
\]

so (negligible rounding errors aside) the exact inverse is:

\[
A^{-1} = \begin{pmatrix}
70 & -71 & -1 & 7 \\
-252 & 255 & 4 & -25 \\
121 & -122 & -2 & 12 \\
1 & -1 & 0 & 0
\end{pmatrix}, \text{ running time was 41 seconds.}
\]

Notes

1. Just for fun, the running time in seconds to invert an N\(\times\)N matrix is approximately: \(t = 3.37 - 0.08 N + 0.33 N^2 + 0.52N^3\).

2. This program was published in PPC Technical Notes V1N2 pp4-7 (September 1980).

References


Copyrights

Copyright for this paper and its contents is retained by the author. Permission to use it for non-profit purposes is granted as long as the contents aren’t modified in any way and the copyright is acknowledged.