HP Forum Archive 18

The 7x7 Linear Equation Solver for the HP-67

Message #1 Posted by ***Palmer O. Hanson, Jr.*** *on 3 Apr 2008, 1:08 a.m.*

I have the Raymond Broeckx linear systems solver up and running on my HP-67. I have done both of his sample problems and get his results. The execution times are long as I expected they would be. For example, for the 3x3 sample problem the calculator runs about five seconds after the first equation is entered before it is ready for the second equation. The calculator runs about 25 seconds after the second equation is entered before it is ready for the third equation. After the third equation is entered the calculator then runs about 28 seconds to reach the solution.

For the 7x7 sample problem the run times after each equation are much longer:

```
        After Equation                    Run Time (sec)


              1                               12
              2                               64
              3                              105
              4                              134
              5                              147
              6                              140
              7                               96
```

I will report some results for other problems later this week.

In an earlier thread someone asked what the -x- command at step 184 of the program does. Page 172 of the *HP-67 Owner's Handbook and Programming Guide* explains:

> Quote:
>
> ... -x-, when encountered as an instruction by a running program, halts the program and displays the contents of the X-register for about 5 seconds, plenty of time to write down the answer, in most cases. So that you will know that the program has not stopped completely, the decimal point blinks eight times during the pause. When the pause is completed, the program resumes execution automatically with the next instruction in program memory. ...

I find that replacing the -x- with R/S is more to my liking.

Palmer

-x- statement ...

Message #2 Posted by ***Nenad (Croatia)*** *on 3 Apr 2008, 5:59 a.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

... was put into HP67 for compatibility with HP97 PRINT X statement.

Re: The 7x7 Linear Equation Solver for the HP-67

Message #3 Posted by ***Namir*** *on 3 Apr 2008, 12:05 p.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

Hi Palmer,

Can you please provide a link to the original source code?

Namir

An alternative 7x7 Linear Equation Solver for the HP-67

Message #4 Posted by ***Fernando del Rey*** *on 3 Apr 2008, 12:18 p.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

It's curious to read the recent threads about the ability of the HP-67 to solve systems of linear equations 7x7, given its limited storage capacity. It's rather amazing that 30-some years later, there can still be interest for such a topic.

I preserve copies of several fine programs by Mr. Raymond Broeckx, but not this one mentioned here. The program seems very similar to one I wrote back in 1977, which Valentin has mentioned on an earlier thread.

Just for the sake of adding to the historical background, I reproduce herein the listing of that program as I can read it from the old photocopies (I don't have access to a physical HP-67 to read and test the program, so I must apologize if any errors are introduced in the transcription). I have one listing hand-written by Valentin (only 30 years younger!) with brief usage instructions in Spanish, and a second listing written by myself on a User's Library submittal form.

This is what I can read from both (I have double checked, trying to avoid mistakes):

```
 1. LBL A
 2. 1
 3. STO + 0
 4. RDN
 5. LBL B
 6. STO E
 7. 2
 8. 5
 9. RCL 0
10. -
11. STO I
12. LBL 6
13. 2
14. 4
15. X=Y
16. GTO 5
17. R/S
18. STO (i)
19. ISZ
20. RCL I
21. GTO 6
22. LBL 5
23. GSB e
24. STO I
25. LBL 4
26. R/S
27. STO (i)
28. GSB d
29. RCL I
30. ISZ
31. X.ne.Y (X not equal to Y?)
32. GTO 4
33. RCL E
34. 1
35. STO E
```

```
36. RDN
37. LBL 1
38. X=0
39. GTO a
40. GSB e
41. STO I
42. RDN
43. GSB d
44. ENTER (**)
45. LBL 2
46. RDN
47. X<>Y
48. STO / I
49. X<>Y
50. RCL I
51. ISZ
52. X.ne.Y
53. GTO 2
54. RDN
55. RDN
56. 2
57. 4
58. RCL 0
59. -
60. RCL E
61. +
62. STO I
63. LBL 0
64. 2
65. 4
66. -
67. X=0
68. GTO 3
69. RDN
70. STO / I
```

```
 71. ISZ
 72. RCL I
 73. GTO 0
 74. LBL 3
 75. RCL 0
 76. RCL E
 77. X=Y
 78. GTO b
 79. 1
 80. -
 81. 9
 82. RCL 0
 83. -
 84. X
 85. STO I
 86. ISZ
 87. RCL D
 88. RCL (i)
 89. -
 90. STO D
 91. ISZ
 92. GSB d
 93. GSB e
 94. LBL c
 95. RCL (i)
 96. ISZ
 97. X<>Y
 98. X<>I
 99. X<>Y
100. STO - (i)
101. ISZ
102. RDN
103. X<>I
104. X.le.Y (X less or equal to Y?)
105. GTO c
```

```
106. LBL a
107. RCL E
108. 1
109. +
110. STO E
111. 2
112. 3
113. +
114. RCL 0
115. -
116. STO I
117. 2
118. 4
119. X=Y
120. GTO E
121. RCL (i)
122. GTO 1
123. LBL d
124. GSB e
125. 7
126. +
127. RCL 0
128. -
129. RTN
130. LBL e
131. 9
132. RCL 0
133. -
134. RCL 0
135. 1
136. -
137. X
138. 1
139. +
140. RTN
```

```
141. LBL b
142. 1
143. X=Y
144. RTN
145. STO I
146. GSB e
147. 2
148. -
149. STO D
150. 9
151. RCL 0
152. -
153. STO B
154. LBL 8
155. RCL (i)
156. STO E
157. ISZ
158. LBL 7
159. RCL I
160. RCL D
161. +
162. X<>I
163. RCL (i)
164. X<>Y
165. X<>I
166. X<>Y
167. RCL E
168. X
169. STO – (i)
170. RCL I
171. ISZ
172. RCL B
173. /
174. FRAC
175. X.ne.0 (X not equal to 0?)
```

```
176. GTO 7
177. RCL D
178. RCL B
179. -
180. STO D
181. X>0
182. GTO 8
183. RCL B
184. 1
185. STO I
186. STO D
187. -
188. STO B
189. LBL 9
190. RCL I
191. RCL D
192. +
193. X<>I
194. RCL (i)
195. X<>Y
196. X<>I
197. X<>Y
198. STO (i)
199. RCL I
200. ISZ
201. RCL B
202. /
203. FRAC
204. X.ne.0
205. GTO 9
206. LAST X
207. RCL 0
208. X=Y
209. RTN
210. 1
```

```
211. -
212. RCL D
213. X=Y
214. GTO 9
215. 1
216. +
217. STO D
218. GTO 9
219. LBL C
220. CL REG
221. RTN
```

(**) This line (line 044) appears to be different on both listings. On one it looks like the ENTER command (Key Code 41), while the other copy reads a strange (R)^ (could be the Roll Up stack command?). I have no means of verifying which is correct, but the Key Code 41 leads me to believe that ENTER is the correct line.

The program card (which I still preserve but don't have a physical 67 to run on) has written labels for keys A, B, and C, with the following wording:

```
A -> Start of line
B -> Restart of line
C -> Start
```

The Spanish instructions say sort-of the following:

To start: C

To enter equation lines:

a_{11} A, a_{12} R/S, a_{13} R/S, ... b_1 R/S
a_{21} A, a_{22} R/S, a_{23} R/S, ... b_2 R/S
..
a_{71} A, a_{72} R/S, a_{73} R/S, ... b_7 R/S

To retrieve solutions:

x_1 -> RCL 1
x_2 -> RCL 2

```
...
x7 -> RCL 7
```

If you get an error while entering a line and you know that the system is not singular, leave this line to be re-entered later. To introduce a new line after the error, start it by using key B instead of A.

Total execution time is 700 seconds (I read this from Valentin's hand-writing). This is nearly identical to Mr. Broeckx's program, which is not surprising since both must be executing the very same algorithm.

Just to complete the historical background, Valentin and I co-authored a similar program for the HP-41 which was published on the PPC Journal (V7 N5 P63, June 1980), and I wrote still another version but for equations with complex coefficients, published in the Australian PPC Technical Notes #12 (Jan-Feb 1982).

I am now working on a similar but more elegant program for the 35s which I was hoping to submit to the next issue of Datafile, but I'm afraid it's going to miss the deadline. I still hope to finish it soon, time permitting.

Re: An alternative 7x7 Linear Equation Solver for the HP-67

Message #5 Posted by ***Stefan Vorkoetter*** *on 3 Apr 2008, 3:28 p.m.,*
in response to message #4 by Fernando del Rey

> Quote:
>
> I am now working on a similar but more elegant program for the 35s which I was hoping to submit to the next issue of Datafile, but I'm afraid it's going to miss the deadline. I still hope to finish it soon, time permitting.

Note that I've already written such a program for the HP 35s, the Matrix Multi-Tool. In fact I just finished posting a revised version, which adds two new features:

1. User accessible matrix-vector multiplication.

2. Externally callable entry points for the equation solver and the matrix-vector multiplication, allowing the multi-tool to be used as a subroutine package for other programs.

The program is quite a bit longer than the 7x7 HP-67 program (445 steps, 1464 bytes), but it has a matrix and vector editor, prompting user interface, and of course can store the entire matrix, which means it can be a bit cleverer during the solution (pivoting). Also, it can handle up to 18x18 matrices.

Stefan

Re: An alternative 7x7 Linear Equation Solver for the HP-67

Message #6 Posted by ***Palmer O. Hanson, Jr.*** *on 3 Apr 2008, 10:35 p.m.,*
in response to message #4 by Fernando del Rey

> Quote:
>
> Just to complete the historical background, Valentin and I co-authored a similar program for the HP-41 which was published on the PPC Journal (V7 N5 P63, June 1980), and I wrote still another version but for equations with complex coefficients, published in the Australian PPC Technical Notes #12 (Jan-Feb 1982).

A similar program for the TI-59 which extended the capability of that machine from eighth order with the Master Library ML-02 program to sixteenth order but at a cost in accuracy was published in early 1981 in the Swedish newsletter *Programbiten*. The *Programbiten* program was the basis for my eighth order program for the HP-33s which appears as Article 678 in this site.

I have worked with one of Valentin's programs for the HP-41 from the *HP PPC Journal*. I haven't located my notes yet. They may be at my summer home.

Palmer

Re: An alternative 7x7 Linear Equation Solver for the HP-67

Message #7 Posted by ***Palmer O. Hanson, Jr.*** *on 10 Apr 2008, 11:41 p.m.,*
in response to message #4 by Fernando del Rey

> Quote:
>
> The program card (which I still preserve but don't have a physical 67 to run on) has written labels for keys A, B, and C, with the following wording:
>
> ```
> A -> Start of line
> B -> Restart of line
> C -> Start
> ```
>
> The Spanish instructions say sort-of the following:
>
> ```
> To start: C
>
>
> To enter equation lines:
> ```

```
a11 A, a12 R/S, a13 R/S, ... b1 R/S
a21 A, a22 R/S, a23 R/S, ... b2 R/S
..
a71 A, a72 R/S, a73 R/S, ... b7 R/S


To retrieve solutions:


x1 -> RCL 1
x2 -> RCL 2
...
x7 -> RCL 7
```

If you get an error while entering a line and you know that the system is not singular, leave this line to be re-entered later. To introduce a new line after the error, start it by using key B instead of A.

I have loaded the program but I don't have it working yet. I had hoped to test the program first with a system of lower order. What I don't see in the instructions is anything that allows me to set the order of the linear system. Did this program only work for seventh order?

Palmer

Re: An alternative 7x7 Linear Equation Solver for the HP-67

Message #8 Posted by ***Fernando del Rey*** *on 11 Apr 2008, 3:50 a.m.,*
in response to message #7 by Palmer O. Hanson, Jr.

Palmer,

> Quote:
>
> I have loaded the program but I don't have it working yet. I had hoped to test the program first with a system of lower order. What I don't see in the instructions is anything that allows me to set the order of the linear system. Did this program only work for seventh order?
>
> Palmer

Yes, I'm afraid the program only works for systems of 7x7. In my old collection of cards for the HP-67 I have a different card for systems of 6x6 and another one for 5x5, so I never got around to create a generalised version like the one from Mr. Broeckx. I vaguely remember creating these programs in an incremental way, to see how far I could reach, but I must have been happy just to have one version (in one card) for each size of system.

You can still solve systems of lower order by entering dummy coefficientes as fillers, but it is not very convenient I understand.

By the way, though I don't have an HP-67 to run the program, I do have a 41 with card reader. If you try some example and it does not work, let me know and I will run it on the 41.

Regards,

Fernando

Re: An alternative 7x7 Linear Equation Solver for the HP-67

Message #9 Posted by ***Palmer O. Hanson, Jr.*** *on 12 Apr 2008, 10:26 p.m.,*
in response to message #4 by Fernando del Rey

Fernando:

I finished loading your program into my HP-67 with ENTER at step 044. I have successfully run three 7x7 problems: the test case with the Broeckx program documentation, the modified sub-Hilbert and Valentin's problem from this thread. For the Broeckx test case the results from your program agree with the published results from his program to eight significant digits. The following table compares the results from your program with the exact answer and with thr results from the Broeckx program:

```
          Exact                          Broecks             Fernando


 1.554001554001554...E-04          1.268165871          2.023822812
-4.195801958041958...E-03         -3.552350670         -5.274249208
 3.496503496503496...E-02          3.037011615          4.278309948
-1.282051282051282...E-01         -1.134715293         -1.535762177
 2.307692307692307...E-01          2.072117992          2.717336358
-2E-01                            -1.816384763         -2.321899231
 6.666666666666666...E-02          6.110651499          7.648148148


RMS Relative Error                  1.27E-01             2.15E-01
```

where for whatever reason the magnitude of the Broeckx results are always smaller than that of the exact results and the magnitude of your results are always larger than that of the exact results. The first entry in the Broeckx results is different from that previously published to correct an error.

The following table compares results from the Broeckx program and from your program for Valentin's problem where the exact answer is a vector of seven ones.:

```
  Broeckx                    Fernando


-1.398589375               3.384562443
 1.000302400               0.9996994007
-1.138628146               3.126121577
-1.150846868               3.138268917
 0.998515899               1.001475407
 3.150813788              -1.138236042
 5.5                      -3.473684211
```

where the calculated RMS Relative Errors for the two programs are about 2.37 .

I have not yet tested the error recovery feature. I probably won't do that since I have already worked with a similar capability in Valentin's program for the HP-41.

The major deficiency of this program and the Broeckx progam is the absence of prompts for the individual entries. I do understand that there is a memory limitation; however, if you work with the TI-59 program or the HP-41 program which use a similar method of solution you will see how helpful the prompts are.

Palmer

Some results for a harder 7x7 problem

Message #10 Posted by ***Palmer O. Hanson, Jr.*** *on 4 Apr 2008, 5:05 a.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

Foir a harder problem I selected a matrix made up of a 7x7 sub-Hilbert multiplied by 360360 to make all of the elements integers and a vector of seven ones. I did the problem four ways: with the Broecks program on the HP-67, with the matrix routines in the Math Pac 1 on the HP-41, with the ML-02 program from the Master Library on the TI-59 and with a modified version of the *Programbiten* program on the TI-59.

```
      Exact                    HP-67          HP-41          TI-59 ML-02       TI-59 Program


 1.554001554001554...E-04     1.572819930     1.268165871     1.553996420296     1.552586721795
-4.195801958041958...E-03    -3.552350670    -4.242358580    -4.195792949492     1.552586721795
 3.496503496503496...E-02     3.037011615     3.532036790     3.496495633614     3.494150518513
-1.282051282051282...E-01    -1.134715293    -1.294026833    -1.282048804286    -1.281287947738
 2.307692307692307...E-01     2.072117992     2.327597187     2.307688403892     2.306460225949
```

```
-2E-01                        -1.816384763    -2.016005033    -1.999996995590    -1.999032135698
 6.666666666666666...E-02      6.110651499     6.716386939     6.666657669087     6.663716510582


Approximate Relative Error      1E-01          1E-02           1E-06                 5E-04
```

The "exact" results are repeating or terminating decimals since they are the quotients of integers. I included the mantissa and the exponent for the "exact" results. I dropped the exponents for the machine results.

I included the HP-41 Math Pac results to show what another ten digit machine would yield using Gaussian elimination with partial pivoting. For the problem at hand the program uses 72 seconds to find the determinant and 40 more seconds to find the solution. I would expect tha Advantage Pac for the HP-41 would yield better results. I don't have an Advantage Pac so someone else will have to provide those results.

I included the TI-59 programs since this thread is a followon to a thread which asked whether the HP-67 or the TI-58/59 was better. The ML-02 program uses 125 seconds to fiind the determinant and 45 more seconds to find the solution The *Programbiten* program operates in a mode similar to that of the Broecks program on the HP-67 in that some calculations are completed after each equation is entered. The calculation time after the first equation is entered is 4 seconds. The calculatioin times after the remaining six equations are entered are 8, 15, 15, 15, 14 and 14 seconds.

Coming next, results from my 8x8 solver for the HP-33s which is a translation from the Programbiten program and results from Stefan's matrix solver.

Palmer

Re: Some results for a harder 7x7 problem

Message #11 Posted by ***Valentin Albillo*** *on 4 Apr 2008, 5:22 a.m.,*
in response to message #10 by Palmer O. Hanson, Jr.

Hi, Palmer:

For a good laugh, try this one:

$$
\begin{aligned}
13 x_1 + 72 x_2 + 57 x_3 + 94 x_4 + 90 x_5 + 92 x_6 + 35 x_7 &= 453\\
40 x_1 + 93 x_2 + 90 x_3 + 99 x_4 + x_5 + 95 x_6 + 66 x_7 &= 484\\
48 x_1 + 91 x_2 + 71 x_3 + 48 x_4 + 93 x_5 + 32 x_6 + 67 x_7 &= 450\\
7 x_1 + 93 x_2 + 29 x_3 + 2 x_4 + 24 x_5 + 24 x_6 + 7 x_7 &= 186\\
41 x_1 + 84 x_2 + 44 x_3 + 40 x_4 + 82 x_5 + 27 x_6 + 49 x_7 &= 367\\
3 x_1 + 72 x_2 + 6 x_3 + 33 x_4 + 97 x_5 + 34 x_6 + 4 x_7 &= 249\\
43 x_1 + 82 x_2 + 66 x_3 + 43 x_4 + 83 x_5 + 29 x_6 + 61 x_7 &= 407
\end{aligned}
$$

which has the obvious, unique solution:

$x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = x_7 = 1$

and has nice, well-balanced, small integer elements.

Tell me what you get, I'd be *most* interested. Thanks in advance and

Best regards from V.

Re: Some results for a harder 7x7 problem

*Message #12 Posted by **John B. Smitherman** on 6 Apr 2008, 12:28 p.m.,*
in response to message #11 by Valentin Albillo

Hi Albillo. Not to divert attention away from the HP 67, the HP 50G does fine with this particular system of linear equations. However, the TI 83+ falls on it's face.

Regards,

John

Re: Some results for a harder 7x7 problem

*Message #13 Posted by **Gene Wright** on 6 Apr 2008, 3:58 p.m.,*
in response to message #12 by John B. Smitherman

The 50g cheats, depending on a flag setting. :-)

Re: Some results for a harder 7x7 problem

*Message #14 Posted by **Werner** on 8 Apr 2008, 1:37 a.m.,*
in response to message #13 by Gene Wright

? that's only to calculate the determinant of a matrix with only integer entries, which must be an integer as well. Here, he probably solved the system in exact mode.

Cheers, Werner

Re: Some results for a harder 7x7 problem

Message #15 Posted by ***Valentin Albillo*** *on 7 Apr 2008, 1:53 p.m.,*
in response to message #12 by John B. Smitherman

The HP-50g is indeed cheating.

Just divide all coefficients by 10 (i.e.: 1.3 x_1 instead of 13 x_1, etc) and the right-hand terms by 100 (i.e.: 4.53 instead of 453, etc), and see how it copes.

The solution should now be $x_1 = x_2 = ... = x_7 = 0.1$.

Best regards from V.

Re: Some results for a harder 7x7 problem

Message #16 Posted by ***John B. Smitherman*** *on 7 Apr 2008, 5:31 p.m.,*
in response to message #15 by Valentin Albillo

Hi Valentin. The 50G copes well with the new system you defined essentially finding a solution of 0.1 +/- 0.00x for each value of x.

Regards,

John

Re: Some results for a harder 7x7 problem

Message #17 Posted by ***J-F Garnier*** *on 8 Apr 2008, 2:53 p.m.,*
in response to message #15 by Valentin Albillo

The HP48G also finds the correct solution, even of this non-integer problem [0.1 0.1 0.1 0.1 0.1 0.1 0.1]. But not the HP48S.

And the HP-71B Math ROM also failed, to my great disappointment...

It would be interesting to track the changes in the matrix algorithms from the HP-71B to the HP48G: the results on the 71B, 48S and 48G are all different.

J-F

Addendum: the HP28S gives the same answer than the 48S.

Edited: 8 Apr 2008, 4:10 p.m.

Re: Some results for a harder 7x7 problem

*Message #18 Posted by **J-F Garnier** on 9 Apr 2008, 3:11 p.m.,*
in response to message #15 by Valentin Albillo

Hi Valentin,

Sorry, I didn't recognize at first glance one of your famous "Albillo Matrices" that you described in your "Mean Matrices" paper about 3 years ago...

J-F

3 years !? Indeed, time flies ! ... [NT]

*Message #19 Posted by **Valentin Albillo** on 9 Apr 2008, 3:26 p.m.,*
in response to message #18 by J-F Garnier

Best regards from V.

Re: Some results for a harder 7x7 problem

*Message #20 Posted by **Rodger Rosenbaum** on 9 Apr 2008, 3:56 a.m.,*
in response to message #12 by John B. Smitherman

Be sure you're in approximate mode (set flag -105) when you enter the numbers. If you entered them in exact mode, execute ->NUM with the matrix on the stack, and do the same for the column vector.

If you solve the system as Valentin gave it, the HP50 gets a solution of [1. 1. 1. 1. 1. 1. 1]T (using approximate mode). The HP50 is good, but it's not that good; it's an anomaly. The HP50 won't do that well most of the time.

To see a more typical result, transpose Valentin's matrix, and use a different right hand column vector:

[195 587 363 359 470 333 289]T.

The exact solution should be [1. 1. 1. 1. 1. 1. 1.]T as for the original system.

Working with Valentin's example

Message #21 Posted by ***Palmer O. Hanson, Jr.*** *on 9 Apr 2008, 11:26 p.m.,*
in response to message #11 by Valentin Albillo

I started my analysis with Valentin's example with the HP-41 Math Pac for no particular reason other than I would want to compare the HP-67 results with the Math Pac results since both machines are ten digit machines. The determinant was calculated as -86.30659184 . That didn't seem reasonable since Valentin has historically provided problems like this with well-behaved solutions, often yielding a determinant of 1. So, I tried it again and got the same result. I tried some other machines. My HP-28S calculated the determinant as 7.03238892937E-2 ! ML-02 on the TI-59 calculated the determinant as 1.024890772918 .My TI-86 calculated the determinant as zero.

I decided that I should find out what the exact determinant was so I reluctantly pulled out my TI-89. In a recent thread "design-nut" reported that he was not using his HP-50 very much because ..." I think it's a horrific amount of learning involved ..." I have similar feelings about the TI-89 -- too many menus, too many options for each menu, and too little patience on my part. Maybe that's because I'm pushing eighty. I did it any way and the TI-89 reported that the exact determinant was one.

I had another idea. I knew that if I reversed the columns in the matrix; i.e., entering the rows from right to left rather than right to left then the result (in a perfect world) should only be a change in sign. The following table presents the determinants for Valentin's matrix when entered in the forward and reverse direction:

```
                          Forward                 Reverse


HP-41 Math Pac       -86.30659184            +21.05471966


HP-28S                +7.03238892937E-2       -0.657461352048


HP-35s Stefan         +1.06684536599          -1.00834672918


TI-59 ML-02           +1.024890772918         -1.014431907782


TI-95 Math Module     +1.001715854872         -1.027663133267
```

I'm not sure that I understand much of this. I suspect that it says that Valentin's matrix is a really difficult matrix. I decided to continue on and solve the set of linear equations for the various machines using both the forward and reverse matrices. The solution vector (again, in a perfect world) should have the same values but in reverse order. The results for the HP-41 Math Pac program, for the HP-67 program and for B/A on the HP-28S are:

```
   HP-41           HP-41            HP-67            HP-67             HP-28S             HP-28S
  Forward         Reverse          Forward          Reverse           Forward            Reverse
```

```
1.303525198     3.242369937     -1.398589375      17.08182755      -2.94919254845      3.04627077473
0.999961733     2.071760046      1.000302400       8.686448254      1.00049786612      1.97803277687
1.270628887     0.999260468     -1.138628146       0.994696339     -2.521175647        0.999325143367
1.272175068    -0.071776533     -1.150846868      -6.686566476     -2.5412934091       2.19521805674E-2
1.000187803    -0.065687881      0.998515899      -6.642899805      0.997556494395     2.75083802477E-2
0.727829119     1.000150684      3.150813788       1.001080647      4.54123894334      1.00013750256
0.430555556    -0.195227704      5.5              -7.571933367      8.40909090909     -9.07029478458E-2
```

Results from Stefan's program for the HP-35s which follow include forward and reverse solutions for the Modified 7x7 sub-Hilbert problem and forward and reverse solutions for Valentin's problem.:

```
   sub-Hilbert           sub-Hilbert              Valentin              Valentin
     Forward               Reverse                Forward                Reverse


 1.55436746032E-4      6.66660207475E-2        6.80587353933E-1       -8.31889109179E-2
-4.19665984724E-3     -1.99997947504E-1        1.00004026765           4.79828583766E-1
 3.49713257772E-2      2.30766716511E-1        7.15205572516E-1        1.00035892572
-1.28225768337E-1     -1.28203642256E-1        7.13578437954E-1        1.52017941667
 2.30802850463E-1      3.49646032050E-2        0.999802368070          1.51722432517
 2.00026608729E-1     -4.19574896879E-3        1.28641715685           9.99926868643E-1
 6.66748283464E-2      1.55539797584E-4        1.59925093633           1.58009558596
```

where the correspondence between the forward and reverse solutions for the modified sub-Hilbert is evident, but the reverse solution is about an order of magnitude more accurate. There is no obvious correspondence between the forward and reverse solutions for the Valentin problem.

I would have liked to have the solution with the HP-41 Advantage module. I don't have one. Someone else will have to provide those results.

Palmer

Re: Working with Valentin's example

Message #22 Posted by ***Valentin Albillo*** *on 10 Apr 2008, 6:59 a.m.,*
in response to message #21 by Palmer O. Hanson, Jr.

Hi, Palmer:

> Thanks a lot for your magnificent efforts on this one, it sure must have taken ages and lots of eyestrain reading those many digits from the tiny displays and writing them down accurately for posting, very *much* appreciated.
>
> What it all means is that my matrix is indeed extremely troublesome, much more so than any Hilbert-related one and with the advantage of being much better balanced, having just small, 2-digit positive integer coefficients. You can't generate the coefficients on the fly as you can with

Hilbert's, but then again there's just 49 of them, 91 digits in all. Not too much typing at all, and once you've got them into a DATA statement or EM file, say, no need to type them again.

The troubles you're experiencing here can be alleviated by using exact arithmetic or carrying many more significant digits, say 20 (as may SHARP PC-1475 does) but nevertheless you'll lose 12-14 of them depending on rounding, algorithm, etc. And this is just for the 7x7 case, larger matrices (say 10x10) are much worse, nearly intractable, even using double precision. I took great care to make sure they are.

By the way, I didn't know you were about eighty, I think *"designnut"* is also. Though being some 30 years your junior I concur with your feelings about overcomplex calculators and handhelds in general: they may be über-powerful and have thousands of functions and possibilities, but unfortunately they combine that with either unfathomable, write-only programming paradigms or combinatorially-exploding menu trees or both, and in the end this just gets in the way and on the nerves.

Thanks again and best regards from V.

Re: Working with Valentin's example

Message #23 Posted by ***Rodger Rosenbaum*** *on 10 Apr 2008, 9:12 p.m.,*
in response to message #22 by Valentin Albillo

Valentin said:

> Quote:
>
> ...my matrix is indeed extremely troublesome, much more so than any Hilbert-related one...

I don't think this is quite true. The usual measure of how troublesome a matrix is the condition number. We can find "Hilbert-related" matrices with condition number similar to or greater than yours.

For example, your matrix has a condition number of about 3.17E12, while an ordinary Hilbert matrix of order 10 has a condition number of 1.6E13, and is clearly a "Hilbert-related" matrix. But, you might say, this is an order 10 matrix, while your matrix is order 7.

So, let us take an ordinary Hilbert matrix of order 12 and delete the first 5 rows and columns. Now we have an order 7, "Hilbert-related" matrix with a condition number of 8.3E12.

If we want to follow this procedure further, we can find 7th order "Hilbert-related" matrices that are much more troublesome than yours.

Sure (was: Re: Working with Valentin's example)

Message #24 Posted by ***Valentin Albillo*** *on 11 Apr 2008, 6:30 a.m.,*
in response to message #23 by Rodger Rosenbaum

Hi, Rodger:

Rodger posted:

> *"I don't think this is quite true. The usual measure of how troublesome a matrix is the condition number. We can find "Hilbert-related" matrices with condition number similar to or greater than yours."*
>
> > Your statement is vague. Please state exactly how you do compute the condition number and which matrix or matrices are you referring to when you say *"yours"*. Are you referring to the specific 7x7 matrix (which happens to be what I call AM#7) or some other matrices of various dimensions and characteristics belonging to my AM series ?
> >
> > It isn't fair to compare a whole generalized, NxN class of matrices you call *"Hilbert-related matrices"* to a *single*, *particular* instance of my AM suite and draw such conclusions as you do. It is neither fair nor useful if you're allowed to pickpoint *any of an infinite class* of matrices to compete against *one specific instance* of mine, you'd be proving nothing. That would smell fishy, don't you think ?
>
> *"For example, your matrix has a condition number of about 3.17E12, while an ordinary Hilbert matrix of order 10 has a condition number of 1.6E13, and is clearly a "Hilbert-related" matrix. But, you might say, this is an order 10 matrix, while your matrix is order 7."*
>
> > See ? Even *you* find your arguments 'fishy'. Comparing a 10x10 matrix against a 7x7 one !? Come on ! A 10x10 AM-series matrix can have a condition number *many* orders of magnitude higher than your *"ordinary Hilbert matrix"*. Please be serious.
>
> *"So, let us take an ordinary Hilbert matrix of order 12 and delete the first 5 rows and columns. Now we have an order 7, "Hilbert-related" matrix with a condition number of 8.3E12."*
>
> > I insist. If you want to make a meaningful, valid point, it's essential to play fair, which so far is eluding you. As I see it, you're taking a *specific*, particular, concrete 7x7 matrix I gave, which is a *fixed* target (more like a sitting duck in fact), and then you're allowed to arbitrarily choose from *infinite* NxN matrices of *larger* dimensions, of course, and do *arbitrary operations* to them, to concoct a matrix with unknown characteristics and unknown element-size (which you don't provide explicitly, just mention that it can be done) and that would beat my fixed one.

In other words, you're fighting against a sitting duck, a "straw man", and then you're defeating it (or so you say) using whatever techniques you can think of, with no limits, no specificactions, and saying that proves your point. Great strategy.

"If we want to follow this procedure further, we can find 7th order "Hilbert-related" matrices that are much more troublesome than yours."

No, Rodger, step down from your cloud. If you want to play this fairly, this is what you should do:

- Specify *which Hilbert-like matrix you're going to use* and which operations you'll do to it to arrive at a *previously specified size matrix* (say 7x7), suitably scaled to consist of *integer, positive elements* so all number-crunching can be done theoretically exactly, without decimals, fractions, or symbolic computations being required.
- Specify the *maximum size* of the elements, i.e., how large can they be: 2-digit ? 3-digit ? 6-digit ?
- Specify the exact method you use to compute the condition number, then exhibit a matrix of yours which conforms to the specifications above and state its condition number computed as specified in this point.
- We'll define such a matrix as *"winning"* over another similarly sized matrix with similarly sized elements if it has a greater condition number than it (both matrices are assumed to be non-singular, of course).

You do that, then I'll obey the exact same specifications and will produce a 7x7 AM (thus non-Hilbert-based) matrix, with integer positive elements of the maximum agreed size, and will also compute its condition number using your algorithm.

Wanna bet which matrix will be the *"winning"* one ? Would you bet an HP-15C ? Do you realistically think you have any chance now that the conditions are fully specified in advance and fair to both parties ?

I thought so.

Best regards from V.

Edited: 11 Apr 2008, 6:39 a.m.

Re: Sure (was: Re: Working with Valentin's example)

Message #25 Posted by ***Rodger Rosenbaum*** *on 11 Apr 2008, 7:40 p.m.,*
in response to message #24 by Valentin Albillo

The assertion of yours we are discussing here is:

> Quote:
>
> ...my matrix is indeed extremely troublesome, much more so than any Hilbert-related one...

Any vagueness in what I said is a direct consequence of the vagueness of *your* statement. You said "...my matrix..." without telling us just what you meant by that. When I said "...your matrix...", I was just restating what you had said, changing the first person "my" to the second person "your". If you want exactitude in my phrase "your matrix", then you must first make your own phrase "...my matrix..." exact.

You said that "...my matrix is indeed extremely troublesome..." without giving us the slightest idea of just what "troublesome" means mathematically, and then you have the nerve to chide me for not telling you how I calculated the condition number. At least I provided a mathematical concept for which a generally accepted calculation could be carried out, something not possible with "troublesome".

If you wanted to know how I compute the condition number you could have made some effort to look it up and see if there is a commonly accepted method. For example, http://mathworld.wolfram.com/ConditionNumber.html

But if you search for the word "troublesome" on Mathworld, there is no definition having to do with matrices. I can find no commonly accepted method of calculating whether a matrix is "troublesome".

You ask me:

> Quote:
>
> Are you referring to the specific 7x7 matrix (which happens to be what I call AM#7) or some other matrices of various dimensions and characteristics belonging to my AM series ?

I was referring to whatever you were referring to when you said "...my matrix...". Since your phrase "...my matrix..." was vague, I used as a specific example the matrix you had given in this thread, the one you call AM#7, guessing that might be the one you were referring to.

You say:

> Quote:

> It isn't fair to compare a whole generalized, NxN class of matrices you call "Hilbert-related matrices" to a single*, *particular* instance of my AM suite and draw such conclusions as you do.

If you didn't want to invite such comparisons, then you shouldn't have said "...my matrix is indeed extremely troublesome, much more so than **any** Hilbert-related one..."

In discussing mathematical concepts, words like "none, some, all, any" are often used. In that context I've never seen the word "any" used to mean anything other than "any"; that is, without restriction. You said "...**any** Hilbert-related matrices...". Since you didn't qualify the phrase, your readers are entitled to believe that it means just what it says:"...**any** Hilbert-related..."

The phrase "...**any** Hilbert-related matrices..." taken at face value, means: matrices strongly related to Hilbert matrices; weakly related to Hilbert matrices; having integer elements or not; and if having integer elements, restricted in some way or not; etc., etc. In other words, without restriction, except that they are somehow "Hilbert-related".

You say:

> Quote:
>
> See ? Even you find your arguments 'fishy'. Comparing a 10x10 matrix against a 7x7 one !? Come on ! A 10x10 AM-series matrix can have a condition number many orders of magnitude higher than your "ordinary Hilbert matrix" Please be serious.

Whatever "fishiness" there might be could only happen because the vagueness of your original statement allowed it. I was illustrating the first example that came to mind and that was allowed by your phrase " ...**any** Hilbert-related...".

What makes you think I wasn't serious? If you didn't want to invite such "fishy" comparisons, you shouldn't make statements that allow them.

You say:

> Quote:
>
> I insist. If you want to make a meaningful, valid point, it's essential to play fair, which so far is eluding you. As I see it, you're taking a specific, particular, concrete 7x7 matrix I gave, which is a fixed target (more like a sitting duck in fact), and then you're allowed to arbitrarily choose from infinite NxN matrices of larger dimensions, of course, and do arbitrary operations to them, to concoct a matrix with unknown characteristics and unknown element-size (which you don't provide explicitly, just mention that it can be done) and that would beat my fixed one.

In other words, you're fighting against a sitting duck, a "straw man", and then you're defeating it (or so you say) using whatever techniques you can think of, with no limits, no specificactions, and saying that proves your point. Great strategy.

Of course I'm allowed to "...arbitrarily choose from infinite NxN matrices of larger dimensions, of course, and do arbitrary operations to them, to concoct a matrix with unknown characteristics and unknown element-size...". Your phrase "...**any** Hilbert-related one" allows just that.

You said "...**any** Hilbert-related one". Do you want me to assume that when you say something in a mathematical context, you don't mean what you say? If you don't want me to "...arbitrarily choose...", then don't say **any**.

As far as "...(which you don't provide explicitly, just mention that it can be done)..." goes, you do this same thing. You provide no proof that "...my matrix is indeed extremely troublesome, much more so than any Hilbert-related one...", you just assert that it is true.

Finally, you say:

"No, Rodger, step down from your cloud. If you want to play this fairly, this is what you should do:"

And then give a list of particulars to make the comparison less vague. The burden isn't on me to do this. This is what you should have done in the first place. I won't deny that if you add some restrictions to your original statement (and it's up to you, not me) to particularize your original statement, then you can no doubt make it true (true meaning that not one counterexample exists). If you do so, I will be sure to read it.

Your original statement described a comparison with only one restriction, namely that "your" (unspecified) matrix was to be compared to **any** "Hilbert-related" (also not particularized) matrix. One counterexample is enough to falsify a mathematical statement, and I gave two which were within the domain of your vague statement, comparing with AM#7 which I guessed might be the "...my matrix..." you were referring to.

Re: Working with Valentin's example

Message #26 Posted by ***Palmer O. Hanson, Jr.*** *on 11 Apr 2008, 12:08 a.m.,*
in response to message #22 by Valentin Albillo

Quote:

... it sure must have taken ages and lots of eyestrain reading those many digits from the tiny displays and writing them down accurately for posting ...

There is more on the way. One interesting result shows that with the TI-95 a Cramer's rule solution using the determinants generated by the linear equation solver in the Math module yields a much better solution to your problem than the linear equation solver by itself.

```
    Solver               Cramer's Rule


0.6825693495438         0.9930691031...
1.000040017787          1.002724798...
0.7169727577414         0.983892769...
0.7153557197560         0.9756815714...
0.9998035943913         0.9995317689...
1.284639902366          1.025133491...
1.595532508975          1.052652573...
```

> Quote:
>
> By the way, I didn't know you were about eighty, I think *"designnut"* is also.

I turned 79 in February. In separate correspondence he told me he graduated in 1951. I graduated in 1950.

Palmer

Re: Working with Valentin's example

Message #27 Posted by ***Stefan Vorkoetter*** *on 10 Apr 2008, 2:52 p.m.,*
in response to message #21 by Palmer O. Hanson, Jr.

I think that the symmetry of the sub-Hilberts is responsible for making the reverse solution approximately the same as the reverse of the forward solution. Valentin's matrix has no such symmetry, and reversing it basically turns it into a totally different problem.

I'm a little surprised that my program is doing as well as it is, since it's not a particularly clever implementation or anything. It's just straightforward Gaussian elimination with partial pivoting (only swapping rows). If I had done full pivoting, I expect the results might be even better.

If I get a chance, I'll try the Advantage Pac program on my 41CX.

Stefan

Re: Working with Valentin's example

Message #28 Posted by ***Palmer O. Hanson, Jr.*** *on 11 Apr 2008, 8:19 a.m.,* *in response to message #27 by Stefan Vorkoetter*

> Quote:
>
> I think that the symmetry of the sub-Hilberts is responsible for making the reverse solution approximately the same as the reverse of the forward solution. Valentin's matrix has no such symmetry, and reversing it basically turns it into a totally different problem.

Stefan:

Try this: given the matrix

```
 6    -1    -3     1

-2     0     1     3

 2    -1     0     1

-3     2    -1     0
```

and a vector of four ones, the answer from your solver will be

```
Det = -1

X1   =   21.0000000002

X2   =   46.0000000003

X3   =   28.0000000002

X4   =    5.00000000002
```

Then, do the problem again, but with the elements of the rows of the matrix in reverse order, and the answer from your solver will be

```
Det   =   -1.00000000001

X1   =   4.99999999997
```

```
X2    =  27.9999999997


X3    =  45.9999999996


X4    =  20.9999999998
```

where the sign of the determinant doesn't change because the number of columns is even. When you think about it, column 1 contains the coefficients of the X1 element of the answer in the first case, but column 4 contains the same elements, but as the coefficients of the X4 element of the answer in the second case, etc.

Palmer

Re: Working with Valentin's example

Message #29 Posted by ***Stefan Vorkoetter*** *on 12 Apr 2008, 12:13 p.m.,*
in response to message #27 by Stefan Vorkoetter

> Quote:
>
> If I get a chance, I'll try the Advantage Pac program on my 41CX.
>
> Stefan

Here are the results from the Advantage Pac matrix program for Valentin's problem (from message #10):

Determinant: -2.916043996

Solution vector:

```
-0.442104267
 1.000181799
-0.285807768
-0.293154069
 0.999107721
 2.293134182
 3.705535763
```

And here are the results with the input matrix reversed left to right:

Determinant: 1.2964614995

Solution vector:

```
 3.272610135
 2.086213628
 0.999250497
-0.086230335
-0.080059560
 1.000152712
-0.211346320
```

By the way, it's very handy to have the card reader. After entering the matrix and vector the first time (which used up registers 00 through 57), I saved them to a pair of cards for later re-use. To create the reversed matrix, I just manually used the C<>C function three times (with arguments of 1.007, 2.006, and 3.005).

Stefan

Edited: 12 Apr 2008, 12:18 p.m.

Re: Working with Valentin's example

Message #30 Posted by ***Palmer O. Hanson, Jr.*** *on 13 Apr 2008, 9:05 p.m.,*
in response to message #29 by Stefan Vorkoetter

> Quote:
>
> Here are the results from the Advantage Pac matrix program for Valentin's problem

Could you provide the Advantage Pac results for the modified sub-Hilbert problem?

Palmer

Re: Working with Valentin's example

Message #31 Posted by ***Stefan Vorkoetter*** *on 14 Apr 2008, 11:36 a.m.,*
in response to message #30 by Palmer O. Hanson, Jr.

Will do, as soon as I get a chance. Just for interest's sake, I ran the Valentin example (forward only for now) through the Maple program that I wrote while designing my HP 35s program. (Basically, I wrote the algorithm in Maple, using Maple's matrices and vectors, and then gradually transformed it to work with a 1 dimensional array mimicking the HP 35s' indirect storage registers. When I was done that, and it was still working, I translated the result to RPN by hand.)

So, as I was saying, I tried the Valentin example in Maple, using floating point arithmetic, but specifying different precisions (Maple's "Digits" setting). Maple does floating point in decimal, with the specified number of digits, unless you specifically ask for hardware floating point (64-bit IEEE). Here are some results:

```
  Digits=10      Digits=11       Digits=12        Digits=13         Digits=15
-1.9316091830  -1.56357167680   0.680587353933   0.9951235256265   0.999703974877925
 1.0003695800   1.00032318460   1.000040267650   1.0000006147630   1.000000037319270
-1.6138788590  -1.28572956040   0.715205572516   0.9956520421084   0.999736058336531
-1.6288129030  -1.29878876640   0.713578437954   0.9956272006323   0.999734550341510
 0.9981861096   0.99841382686   0.999802368070   0.9999969827539   0.999999816838740
 3.6287724730   3.29875341010   1.286417156850   1.0043727321180   1.000265445575820
 6.5000000000   5.80952380950   1.599250936330   1.0091487669050   1.000555373538650
```

Notice that the result at 12 digits is exactly the same as the result that my program produces on the HP 35s. This suggests that the 35s uses 12 digits, and also that its overall floating point model (guard digits, rounding modes, etc.) is the same as Maple's at least as far as these calculations go.

Also notice that 13 digits is the first case where the answers are close to correct (they're all within 1% of 1.0).

I also found it interesting that 2nd and 5th rows are close to correct for all digits settings.

Stefan

Re: Working with Valentin's example

*Message #32 Posted by **Palmer O. Hanson, Jr.** on 14 Apr 2008, 8:55 p.m.,*
in response to message #31 by Stefan Vorkoetter

Stefan:

> Quote:
>
> I also found it interesting that 2nd and 5th rows are close to correct for all digits settings.

If you look at the solutions that I have published for other machines you will see that X2 and X5 are close to the correct digits for all of the solutions even though I used a wide range of machines. It is also true even though some of the solutions used Gaussian elimination with partial pivoting (TI-59 ML-02, HP-41 Math Pac, and your program) and others used an entirely different algorithm (Broeckx and Fernando on the HP-67 and Programbiten on the TI-59). It also holds true for some machine/algorithm pairs that I have not published here as on the TI CC-40 and two different programs on the Sharp PC-1261.

Furthermore, when one enters the equations in reverse order (i.e., entering Valentin's matrix from right to left) one finds that now the really accurate answers are for X3 and X6, which means that the really good results occur with the same column entries.

I have no idea why. I trust that Valentin will explain it.

Palmer

Re: Working with Valentin's example

Message #33 Posted by ***Rodger Rosenbaum*** *on 16 Apr 2008, 6:09 p.m.,*
in response to message #31 by Stefan Vorkoetter

> Quote:
>
> I also found it interesting that 2nd and 5th rows are close to correct for all digits settings.
>
> Stefan

If you calculate Transpose(Inverse(AM#7)) with exact arithmetic, you will get:

TIA =

```
[[    71082       -9      63378       63740       44      -63739      -133357    ]
 [   -507460      64     -452461     -455046     -314      455039      952047    ]
 [ -2128901626  268386 -1898169428 -1909014363 -1317227  1908985002  3994038146  ]
 [ -36543896     4607   -32583237   -32769397   -22611    32768893    68560103   ]
 [ 2658158513  -33428    236420404  237771160   164063   -237767503  -497464609  ]
 [   3554051    -448     3168860     3186965     2199     -3186916    -6667765   ]
 [ 2129774383  -268496  1898947595  1909796976  1317767  -1909767603 -3995675528 ]]
```

I may have made a typo or two, but it wouldn't change the essence of the post.

Given a non-singular square matrix A, the elements of the transposed inverse (call it TIA), Transpose(A^{-1}), have the following property:

The inverse of each element is just the quantity which, when subtracted from the corresponding element of A, will make A singular. That is, the additive perturbation to any particular element (j,k) of A, which will make it singular, is -1/TIA(j,k).

This also tells us that if some fixed perturbation, say .001 for example, is added to any element of A, it will have a perturbing effect proportional to the magnitude of the corresponding element of TIA.

In other words, if a tiny perturbation is added to an element of A, it will have a larger effect if the corresponding element of TIA is large (in absolute value), than if the corresponding element of TIA is small.

For example, if 1/9 is added to the (1,2) element of AM#7, making it become 72+1/9, the matrix will become singular (have a determinant of zero). But in order to make the matrix singular, the (3,1) element need only be increased by 1/2128901626. This means that the matrix is more sensitive to perturbations in the (3,1) element than in the (1,2) element.

The process of solving the system involving AM#7 introduces small perturbations because of rounding errors.

Notice that columns 2 and 5 of TIA are made up of numbers substantially smaller in absolute value than the other columns. Since the 2nd and 5th elements of the solution vector in your post are derived from the corresponding columns of AM#7, the rounding errors produced during the solution process have less perturbing effect on them.

Edited: 17 Apr 2008, 7:35 a.m. after one or more responses were posted

Re: Working with Valentin's example

*Message #34 Posted by **Valentin Albillo** on 17 Apr 2008, 6:16 a.m.,*
in response to message #33 by Rodger Rosenbaum

Hi all:

Rodger posted:

> *"The inverse of each element is just the quantity which, when subtracted from the corresponding element of A, will make A singular. That is, the additive perturbation to any particular element (j,k) of A, which will make it singular, is -1/TIA(j,k)"*

Yes, that's correct. This small (165 byte) HP-71B sample program clearly demonstrates the fact: it generates a 5x5 matrix containing random reals between 0 and 1, and then outputs its non-zero determinant (thus the original random matrix is not singular), and proceeds to perturb each of its 25 elements by the exact amount specified and output the new determinant. As you can see, all 25 perturbations result in the matrix becoming singular as seen by the 25 essentially zero determinants.

```
    10 DESTROY ALL @ OPTION BASE 1 @ DIM A(5,5),B(5,5),C(5,5) @ RANDOMIZE 1
    20 FOR I=1 TO 5 @ FOR J=1 TO 5 @ A(I,J)=RND @ NEXT J @ NEXT I @ DISP DET(A)
    30 MAT B=INV(A) @ MAT B=TRN(B) @ FOR I=1 TO 5 @ FOR J=1 TO 5
    40 MAT C=A @ C(I,J)=C(I,J)-1/B(I,J) @ DISP I;J,DET(C) @ NEXT J @ NEXT I


    >RUN


          6.38546961963E-3  {original random matrix determinant}


          I  J    determinant of perturbed matrix


          1  1      -9.86402690988E-14
          1  2      -4.43920925333E-14
          1  3       2.84377515270E-13
          1  4      -4.72742028783E-15
          1  5      -1.47596921949E-14
          2  1      -4.14531026724E-15
          2  2       4.31166588922E-16
          2  3      -1.11042227698E-13
          2  4       1.12998278784E-14
          2  5       6.34810062362E-14
          3  1      -6.18078629102E-14
          3  2      -1.53885813743E-16
          3  3      -1.68989107927E-13
          3  4      -5.32014948386E-14
          3  5      -1.26818714490E-14
          4  1      -5.33784984718E-14
          4  2       2.75132126351E-14
          4  3      -1.26818714490E-14
          4  4      -1.08906466681E-14
          4  5      -1.73390558794E-14
          5  1      -8.33447568541E-14
          5  2       6.69131342784E-14
          5  3      -3.10920884067E-13
          5  4      -1.67453389653E-14
          5  5      -1.13062108172E-13
```

Best regards from V.

Re: Working with Valentin's example

Message #35 Posted by ***Palmer O. Hanson, Jr.*** *on 17 Apr 2008, 12:02 a.m.,*
in response to message #29 by Stefan Vorkoetter

Stefan:

No sooner does Rodger explain one phenomena than I run into another. My tired, old 79 year old brain is running into overload.

I was working with Valentin's matrix and your program. I had the inverse of Valentin's matrix in A but had lost the solution in b. To save time and get the solution back in b I used the technique "Solving for Another Vector b" from your documentation, i.e., I put Valentin's vector back in b and ran the solver. The vector which appeared in b was

```
  22.
   6.619
-18.
   7.
   0.729
   1.
-10.
```

WOW! I remembered from some previous work somewhere that it was possible to get different sums depending upon the direction of the summing; i.e., the associative rule may not always apply with the digital arithmetic of computers and calculators. So I multiplied the first row of the inverse by the input vector which should yield the first element of the solution vector. When I did that in the forward direction I got 22. When I did that in the reverse direction I got 21.6684 . Neither of those answers is even close to the answer whichh appears if one does the complete problem from start.

I went back to work with the modified 7x7 sub-Hilbert problem by solving and getting the previously published results, and then re-entering the vector of all ones and solving again ("Solving for Another Vector b"). I got

```
 1.55439669E-4
-0.0041966601
 0.0349713332
-0.128225811
 0.230802894
-0.200026622
 0.066674831
```

where the SHOW command verifies that those are all the digits that are there. Furthermore, the results are somewhat less accurate than the full solution. If I work with a much simpler matrix, say the 4x4 matrix from Kahan's "Mathematics Written in Sand" and a vector on four ones the differences are much smaller:

I don't understand all of this. It seems to say that the first solution which includes calculating the determinant and the inverse doesn't use the "quick solve" routine starting at step 383 whhich is no more than multiplying the previously calculated inverse by the input vector. I can only conclude that using the "Solving for Another Vector b" approach is fraught with peril if one is working with sufficiently difficult problems, and that it is safer to go back and re-enter the input matrix.

Admittedly, going back and re-entering the input matrix is time-consuming but a solution is on the way. I have written a program which relies on your program for input and output, for solution of linear equations and for the subroutine which calculates the product of a matrix and a vector but which adds some of the matrix manipulation capabilites found in machines such as the HP-28S and TI-86. My program provides a third matrix location which can be used to save the input matrix together with store, recall and exchange routines for moving matrices between the three matrix locations. It offers calculation of the Frobenius norm, the row norm, the column norm, the dot product, the multiplication of two NxN matrices, the ability to generate an identity matrix and the ability to fill a matrix with a user-defined constant. I hope to have it posted as an article in this forum by this weekend.

Palmer

Re: Working with Valentin's example

Message #36 Posted by ***Stefan Vorkoetter*** *on 17 Apr 2008, 9:27 a.m.,*
in response to message #35 by Palmer O. Hanson, Jr.

The "Solving for Another Vector b" procedure, as you've determined, just multiplies the previously computed A^{-1} by the newly entered b, yielding a new x vector (now stored in b).

When solving a problem for the first time (having just entered A and b), the problem is solved by Gaussian elimination on A. However, in addition to A, there's a scratch matrix next to A in memory, and then the vector b next to that. The scratch matrix starts out as the NxN identity matrix. Basically, memory looks like this:

```
A|I|b
```

As the Gaussian elimination and back substitution proceeds, every operation that is carried out on A (swapping rows, adding rows, multiplying rows by a scalar) is simultaneously carried out on I and b. (Also, the determinant is simultaneously computed based on the row operations done.) At the end of this, memory looks like this:

$I|A^{-1}|x$

At that point, A^{-1} is copied into the location originally occupied by A, after which memory looks like this:

$^{-1}|^{-1}|x$

Since matrices like the 7x7 sub-Hilbert, and AM#7, are ill-conditioned, not only is the computed value of x in this procedure suspect, so is the computed value of A^{-1}.

When you solve for another vector by just multiplying by A^{-1}, it's not surprising that you get a different (and equally bogus) value.

You'll also notice that if you solve (from scratch) one of these ill-conditioned matrices, and then use my program's undo function to get the original back, what you get back won't look much like the original (much like the transporter accident in "Star Trek: The Motion Picture").

So yes, when the problem is ill-conditioned, it's safer to start from scratch.

Your program that interacts with mine sounds promising. Please note however that I've recently revised my program, and the entry points have changed (and also have been documented). If you're going to publish yours to work with mine, it might be best if it worked with the current release of mine.

On a related note, someone (this group perhaps) should develop an HP35s memory allocation library. I bring this up because I too am working on a program that works in conjunction with my Matrix Multi-tool, and it too makes use of additional indirect storage. Clearly my add-on program (which unlike yours, isn't about matrices) is going to conflict with yours.

Stefan Vorkoetter http://www.stefanv.com/calculators

[HP-15C] AM#7 7x7 linear system: results, methods

Message #37 Posted by ***Karl Schneider*** *on 13 Apr 2008, 5:33 a.m.,*
in response to message #11 by Valentin Albillo

Valentin (and all readers) --

I've done some of my own experimenting with this interesting problem, which is to solve an exactly-determined 7th-order linear system based on "Albillo Matrix #7" from Valentin's "Mean Matrices" paper from three years ago.

Quote:

```
13 x1 + 72 x2 + 57 x3 + 94 x4 + 90 x5 + 92 x6 + 35 x7 = 453
40 x1 + 93 x2 + 90 x3 + 99 x4 +    x5 + 95 x6 + 66 x7 = 484
48 x1 + 91 x2 + 71 x3 + 48 x4 + 93 x5 + 32 x6 + 67 x7 = 450
 7 x1 + 93 x2 + 29 x3 +  2 x4 + 24 x5 + 24 x6 +  7 x7 = 186
41 x1 + 84 x2 + 44 x3 + 40 x4 + 82 x5 + 27 x6 + 49 x7 = 367
 3 x1 + 72 x2 +  6 x3 + 33 x4 + 97 x5 + 34 x6 +  4 x7 = 249
43 x1 + 82 x2 + 66 x3 + 43 x4 + 83 x5 + 29 x6 + 61 x7 = 407
```

which has the obvious, unique solution:

```
x1 = x2 = x3 = x4 = x5 = x6 = x7 = 1
```

The determinant is exactly one, as indicated in Valentin's paper, which also noted that ill-conditioned matrices have "determinants that are far too small for their dimensions and element-size." A determinant of 1.00 might be suitable for solving an exactly-determined 2x2 system, but not necessarily a 7x7 system.

The HP-42S calculates the determinant as 0.0703238892937, and the "solution" as [-2.94919 1.00050 -2.52118 -2.54129 0.99756 4.54124 8.40909]^T. I do not know which specific methods it employs.

The HP-15C calculates the determinant as -2.956799886, and the "solution" as [-0.55584 1.00020 -0.38722 -0.39514 0.99904 2.39512 3.91892]^T.

As described in the *HP-15C Advanced Functions Handbook* (AFH): For calculation of determinants, inverses, and linear solutions, the HP-15C performs an LU decomposition of the original matrix A using the Doolittle method. For linear solutions, the LU decomposition will be in-place; for determinants and inverses, it can be output to a different matrix. The form is PA = LU, where P represents the row-transpositions, L is a lower-triangular matrix (normalized so that each of its main-diagonal elements is unity and other elements have magnitudes not exceeding unity), and U is an upper-triangular matrix. L and U are stored in the same space as the original matrix A (omitting the main diagonal of L), with normally-unused bits storing the row-interchange information.

Any "P^{-1}LU" decomposed matrix is indicated with "--" appended to its single-letter identifier in the HP-15C. This altered matrix -- *if accurately computed* -- will produce accurate determinants and linear solutions, much faster than if the original matrix were used.

It's instructive to know why HP implemented the pioneering matrix functionality of the HP-15C in this manner. (After all, why isn't it considered undesirable to alter the user's painstakingly-entered original matrix such that it can't be usefully edited, or to overwrite a different matrix that was stored in necessary RAM space?)

1. First and foremost: The decomposition *allows in-place matrix inversion*, as the L and U components can be created in-place, inverted separately, and multiplied in-place. This conserves scarce user RAM, which is only 64 registers. Inversion of 6x6, 7x7, and 8x8 matrices are possible within available RAM space using this approach. (Reference: pages 96-98 of the *AFH.*)

2. LU decomposition allows for faster linear solutions, because these can be done using simple back-substitution twice, first using U, then L. If repeated solutions are performed, the extra time spent on LU decomposition will be more than recovered by quicker multiple linear solutions.

3. The LU transformation is consistent with practical usage of the matrix functionality. The determinant of a matrix is usually calculated to assess how near-singular it might be. Once the LU decomposition is calculated, the determinant is simply the product of the main-diagonal elements of U (before possible negation by P operations). The LU representation gives the user a fast-performing matrix for subsequent linear solutions, if those are undertaken. However, if the user wants to *preserve* the entered original matrix, he is responsible for allowing space for it (if possible), and for selecting appropriate result-matrix identifiers as needed.

So, how accurately *does* the HP-15C compute the LU decomposition of Albillo Matrix #7?

The HP-15C's LU-transformation matrix (not showing the hidden row-interchange data) is as follows:

```
LU =


48.           91.             71.            48.             93.             32.           67.
 0.1458333333 79.72916667     18.64583334    -4.999999998    10.4375         19.33333333   -2.77083331
 0.8333333333  0.2153122551   26.81865691    60.07656128    -78.74732166     64.17062974   10.76326104
 0.0625        0.8317219754   -0.5199980516  65.39830469     41.55794805     49.28864425    7.7139377741
 0.2708333333  0.5939378103    0.9954401520   0.3695366092  121.6443367     -10.24144422    4.935104748
 0.8541666667  0.07865168535  -0.6753641545   0.6111292124   -0.631667889     4.893688725  -2.33890765
 0.8958333333  0.006009929486  0.08515613588 -0.07776712548   0.07860975206  -0.1245007524  0.00000000074


LU partitioned; U above and L below:


48.           91.             71.            48.             93.             32.           67.
------------- 79.72916667     18.64583334    -4.999999998    10.4375         19.33333333   -2.77083331
 0.1458333333 --------------  26.81865691    60.07656128    -78.74732166     64.17062974   10.76326104
 0.8333333333 0.2153122551  ---------------  65.39830469     41.55794805     49.28864425    7.7139377741
 0.0625       0.8317219754   -0.5199980516 ---------------  121.6443367     -10.24144422    4.935104748
 0.2708333333 0.5939378103    0.9954401520   0.3695366092 -----------------   4.893688725  -2.33890765
 0.8541666667 0.07865168535  -0.6753641545   0.6111292124    -0.631667889  ---------------- 0.00000000074
 0.8958333333 0.006009929486  0.08515613588 -0.07776712548    0.07860975206  -0.1245007524 ---------------
```

At the end of this post are the results from Matlab v6.5, including (same-format) LU decomposition, singular value decomposition (SVD), condition number, inversion, and linear solution.

A careful comparison of the HP-15C's elements of L and U with those of Matlab shows that the HP-15C calculates these results almost as accurately as permitted by its 10 external and 13 internal significant digits. However, for this ill-conditioned matrix, it's not good enough for accurate results. Even Matlab's results using double-precision floating-point mathematics aren't perfect, as a comparison with Valentin's results from Mathematica in the "Mean Matrices" paper will show.

I found only two significant differences between the LU results of HP-15C and Matlab exceeding 7 units in the last place (or, "ulp") of the HP-15C's 10 significant digits:

- Element (7,7) of the HP-15C's U matrix was larger in magnitude by a factor of about 300 (7.4E-10 versus -2.5E-12), and had a different sign. Also note that (7,7) is the only small-magnitude term of this particular U matrix. Only two significant digits were calculated by the HP-15C's for U(7,7); this also was probably a symptom of the near-singularity of this 7x7 "Albillo Matrix".
- Element (7,2) of L was 38 ulp's larger than that of Matlab's result.

-- KS

```
------------------------------------------------------------

A =


    13    72    57    94    90    92    35
    40    93    90    99     1    95    66
    48    91    71    48    93    32    67
     7    93    29     2    24    24     7
    41    84    44    40    82    27    49
     3    72     6    33    97    34     4
    43    82    66    43    83    29    61


b


b =


   453
   484
   450
   186
   367
   249
   407


[L,U,P]=lu(A)


L =
    1.0000         0         0         0         0         0         0
    0.1458    1.0000         0         0         0         0         0
    0.8333    0.2153    1.0000         0         0         0         0
    0.0625    0.8317   -0.5200    1.0000         0         0         0
    0.2708    0.5939    0.9954    0.3695    1.0000         0         0
    0.8542    0.0787   -0.6754    0.6111   -0.6317    1.0000         0
    0.8958    0.0060    0.0852   -0.0778    0.0786   -0.1245    1.0000


U =


   48.0000   91.0000   71.0000   48.0000   93.0000   32.0000   67.0000
         0   79.7292   18.6458   -5.0000   10.4375   19.3333   -2.7708
         0         0   26.8187   60.0766  -78.7473   64.1706   10.7633
         0         0         0   65.3983   41.5579   49.2886    7.7139
         0         0         0         0  121.6443  -10.2414    4.9351
         0         0         0         0         0    4.8937   -2.3390
         0         0         0         0         0         0   -0.0000


P =


     0     0     1     0     0     0     0
     0     0     0     1     0     0     0
     0     1     0     0     0     0     0
     0     0     0     0     0     1     0
     1     0     0     0     0     0     0
     0     0     0     0     1     0     0
     0     0     0     0     0     0     1


format long
[L,U,P]=lu(A)


L =


  Columns 1 through 5


   1.00000000000000                  0                  0                  0                  0
   0.14583333333333   1.00000000000000                  0                  0                  0
   0.83333333333333   0.21531225503005   1.00000000000000                  0                  0
   0.06250000000000   0.83172197543768  -0.51999805134701   1.00000000000000                  0
   0.27083333333333   0.59393781029527   0.99544015199493   0.36953660933039   1.00000000000000
   0.85416666666667   0.07865168539326  -0.67536415452818   0.61112921260013  -0.63166788876545
   0.89583333333333   0.00600992944865   0.08515613582111  -0.07776712546120   0.07860975199350


  Columns 6 through 7


                  0                  0
                  0                  0
                  0                  0
                  0                  0
                  0                  0
   1.00000000000000                  0
  -0.12450075225428   1.00000000000000


U =


  1.0e+002 *


  Columns 1 through 5


   0.48000000000000   0.91000000000000   0.71000000000000   0.48000000000000   0.93000000000000
                  0   0.79729166666667   0.18645833333333  -0.05000000000000   0.10437500000000
                  0                  0   0.26818656911419   0.60076561275150  -0.78747321661876
                  0                  0                  0   0.65398304671896   0.41557948068398
                  0                  0                  0                  0   1.21644336729411
                  0                  0                  0                  0                  0
                  0                  0                  0                  0                  0


  Columns 6 through 7


   0.32000000000000   0.67000000000000
   0.19333333333333  -0.02770833333333
   0.64170629736086   0.10763261039979
   0.49288644224680   0.07713937740537
  -0.10241444215755   0.04935104746538
   0.04893688732152  -0.02338980764163
                  0  -0.00000000000250


P =
     0     0     1     0     0     0     0
     0     0     0     1     0     0     0
     0     1     0     0     0     0     0
     0     0     0     0     0     1     0
     1     0     0     0     0     0     0
     0     0     0     0     1     0     0
     0     0     0     0     0     0     1


cond(A)


ans =


  3.1740e+012


svd(A)


ans =


  399.0161
  120.2434
   81.6069
   65.4718
   15.0734
    2.0586
    0.0000


format long
svd(A)


ans =


  1.0e+002 *


   3.99016086263324
   1.20243367935773
   0.81606914636306
   0.65471837097341
   0.15073408085272
   0.02058584440459
   0.00000000000126


format short
inv(A)


ans =


  1.0e+009 *


    0.0001   -0.0005   -2.1289   -0.0365    0.2652    0.0036    2.1298
   -0.0000    0.0000    0.0003    0.0000   -0.0000   -0.0000   -0.0003
    0.0001   -0.0005   -1.8982   -0.0326    0.2364    0.0032    1.8990
    0.0001   -0.0005   -1.9090   -0.0328    0.2378    0.0032    1.9098
    0.0000   -0.0000   -0.0013   -0.0000    0.0002    0.0000    0.0013
   -0.0001    0.0005    1.9090    0.0328   -0.2378   -0.0032   -1.9098
   -0.0001    0.0010    3.9941    0.0686   -0.4975   -0.0067   -3.9957


format long
inv(A)


ans =


  1.0e+009 *


  Columns 1 through 5


   0.00007108329001  -0.00050746920953  -2.12894026190704  -0.03654455920893   0.26516332517146
  -0.00000000900016   0.00000006400116   0.00026839087074   0.00000460708361  -0.00003342860666
   0.00006337915020  -0.00045246921139  -1.89820387651406  -0.03258382832977   0.23642469462416
   0.00006374115677  -0.00045505425830  -1.90904900833100  -0.03276999170826   0.23777547513807
   0.00000004400080  -0.00000031400570  -0.00131725090541  -0.00002261141035   0.00016406597746
  -0.00006374015676   0.00045504725818   1.90901964679815   0.03276948769911  -0.23777181807170
  -0.00013335942020   0.00095206427802   3.99411063093057   0.06856134724808  -0.49747363712801


  Columns 6 through 7


   0.00355411549992   2.12981303474609
  -0.00000044800813  -0.00026850087274
   0.00316891750936   1.89898205763645
   0.00318702283794   1.90983163553408
   0.00000219903991   0.00131779091521
  -0.00318697383705  -1.90980226200101
  -0.00666788600848  -3.99574804264624


x=inv(A)*b


x =


   1.00000000000000
   0.99999998509884
   1.00000000000000
   1.00024414062500
   1.00000005960464
   0.99987792968750
   1.00000000000000

-----------------------------------------------------------
```

Edited: 16 Apr 2008, 2:26 a.m.

Re: [HP-15C] AM#7 7x7 linear system: results, methods

Message #38 Posted by ***Rodger Rosenbaum*** *on 16 Apr 2008, 5:17 p.m.,*
in response to message #37 by Karl Schneider

> Quote:
>
> The determinant is exactly one, as indicated in Valentin's paper, which also noted that ill-conditioned matrices have "determinants that are far too small for their dimensions and element-size." A determinant of 1.00 might be suitable for solving an exactly-determined 2x2 system, but not necessarily a 7x7 system.

From the text above I can't be sure what mathematical procedure one might apply to determine if matrices have "...determinants that are far too small for their dimensions and element-size.". Some sort of ratio would seem to be involved, a ratio of some quantity embodying "...dimensions and element-size...", to the determinant.

The value of a (square) matrix's determinant is a well-defined number and procedures for calculating it can be found in any relevant text. But the "...dimensions and element-size..." measure alluded to above is not to be found so easily. But I can hazard a couple of possibilities.

One might simply calculate the average of the absolute values of the elements of the matrix times the order of the matrix. In the case of AM#7, that number is 370.857 and since the determinant is 1, the ratio of this measure to the determinant is simply 370.857.

Or, one might calculate the Frobenius norm of the matrix. In the case of AM#7, that value is 429.94, and the pertinent ratio is 429.94.

Since we already know that AM#7 is ill-conditioned, apparently these values 370.857 and 429.94 are of such magnitude as to indicate ill-conditioning. Since these numbers were derived as ratios with the determinant in the denominator, it is clear that a larger determinant would make these ratios smaller and presumably indicate a smaller degree of ill-conditioning.

Now let's perform a numerical experiment. Let's multiply the elements of AM#7 by 1/10. The determinant becomes 1E-7 and the two ratios become 370857142.9 and 429941856.5, apparently indicating a large increase in condition number.

Or, going the other way, and multiplying the elements of AM#7 by 10, the ratios become 3.70857E-4 and 4.299E-4, apparently indicating a significant reduction in ill-conditioning.

Unfortunately these calculations don't give results which are proportional to the condition number. It is well known that the condition number of a matrix is unchanged by multiplication of all the elements by a scalar.

In his excellent book, "Linear Algebra and its Applications", by Gilbert Strang, in section 7.2, "The Norm and Condition Number of a Matrix", Prof. Strang says:

"*...the determinant is a terrible measure of ill-conditioning.* It depends not only on the scaling but also on the order n."

Here's a good example of just how the comparison of the matrix "...dimensions and element-size..." with the determinant fails to give a good measure of ill-conditioning. The comparisons are done with the methods I proposed which seemed to me to embody the spirit of the quote at the beginning of this post. Perhaps there is some other method, not specified in the quote, involving a measure of matrix "...dimensions and element-size..." that might work better, but I don't know what it might be.

Consider this matrix:

EX1 =

```
[[ 66 19 63 81 71 64 61 ]
 [ 64 21 61 72 69 62 60 ]
 [ 56 18 53 64 59 54 52 ]
 [ 57 19 56 62 65 57 52 ]
 [ 39 13 36 44 39 37 37 ]
 [ 84 27 78 97 85 80 79 ]
 [ 74 24 72 83 83 73 68 ]]
```

This matrix is a 7x7 matrix with elements that are all positive integers between 1 and 99, inclusive, and it has a determinant of 1, all characteristics shared by AM#7. It has a condition number of 46857.55, nearly 8 orders of magnitude smaller than the 3.17E12 condition number of AM#7. Almost any calculator could solve a system involving EX1 with reasonable accuracy.

Two ratios were discussed above for AM#7, 370.857 and 429.94, as possible candidates for indicating ill-conditioning. The corresponding ratios for EX1 are 401.429 and 425.77, which are quite similar to those for AM#7. This similarity might lead one to think that the condition numbers of AM#7 and EX1 would also be similar, but in fact they are nearly 8 orders of magnitude apart.

Strang is right; the determinant, alone or in a ratio with some measure of matrix "...dimensions and element-size..." doesn't necessarily indicate ill-conditioning. It may seem to do so sometimes, but don't be fooled. The proper way to calculate matrix condition number is:

http://mathworld.wolfram.com/ConditionNumber.html

Re: [HP-15C] AM#7 7x7 linear system: results, methods

Message #39 Posted by ***Karl Schneider*** *on 17 Apr 2008, 2:21 a.m.,*
in response to message #38 by Rodger Rosenbaum

Rodger --

That was an informative response! It's probably true in most cases that a well-conditioned 7x7 matrix with elements considerably larger than unity has a determinant of large magnitude, but thank you for providing an illustrative counter-example.

> Quote:
>
> ---
>
> Consider this matrix:
>
> ```
> EX1 =
>
>
>
> [[ 66 19 63 81 71 64 61 ]
>  [ 64 21 61 72 69 62 60 ]
>  [ 56 18 53 64 59 54 52 ]
>  [ 57 19 56 62 65 57 52 ]
>  [ 39 13 36 44 39 37 37 ]
> ```

```
[ 84 27 78 97 85 80 79 ]
[ 74 24 72 83 83 73 68 ]]
```

This matrix is a 7x7 matrix with elements that are all positive integers between 1 and 99, inclusive, and it has a determinant of 1, all characteristics shared by AM#7. It has a condition number of 46857.55, nearly 8 orders of magnitude smaller than the 3.17E12 condition number of AM#7. Almost any calculator could solve a system involving EX1 with reasonable accuracy.

And, in fact, the HP-15C does. The determinant is calculated as 0.9999998788; the solution to the vector with each element being the sum of the corresponding row (as before) is calculated as

$[1.000000087\ 0.9999999582\ 0.9999999136\ 0.9999999844\ 1.000000028\ 1.000000004\ 0.9999999919]^T$.

Also telling is that U(7,7) of the HP-15C's LU decomposition is 0.1111111030, and not some minuscule value such as the 7.4E-10 from AM#7.

-- KS

Re: [HP-15C] AM#7 7x7 linear system: results, methods

Message #40 Posted by ***Rodger Rosenbaum*** *on 17 Apr 2008, 5:32 a.m.,*
in response to message #39 by Karl Schneider

Fortunately, most calculators that can do matrix arithmetic also provide for the calculation of one or more norms. The HP15 can calculate the row norm, column norm and Frobenius norm.

If one wants to know the condition number of a matrix, don't even bother with the determinant.

The most accurate computation of condition number is the ratio of largest to smallest singular value (the largest singular value is the spectral norm). But, in most cases, a good approximation can be had by using another norm. The Advanced Functions Handbook goes into this around page 98.

Given any norm applicable to a matrix A, the condition number, k(A), is approximated by: $k(A) = norm(A)^* norm(A^{-1})$

What does the HP15 give you for k(EX1) using the Frobenius norm and the column norm?

Rodger's 7x7 matrix -- calculations by HP-15C and Matlab

Message #41 Posted by ***Karl Schneider*** *on 17 Apr 2008, 11:18 p.m.,* *in response to message #40 by Rodger Rosenbaum*

The 7x7 matrix "A" provided by Rodger, with a small determinant of 1.00, but with a modest condition number:

```
A =


 66  19  63  81  71  64  61
 64  21  61  72  69  62  60
 56  18  53  64  59  54  52
 57  19  56  62  65  57  52
 39  13  36  44  39  37  37
 84  27  78  97  85  80  79
 74  24  72  83  83  73  68
```

Calculations using the HP-15C:

LU partitioned decomposition (L [omitting main diagonal] below; U above; row-swap information not indicated):

```
84.            27.               78.            97.            85.            80.            79.
-------------  -2.214285714      1.714285715    4.785714287    4.214285716    1.142857144   -1.071428570
 0.7857142857  --------------    3.596774191   -2.354838714    8.612903223    3.064516127   -1.935483872
 0.6785714286  -0.3064516126    --------------  0.2705530600   0.2615844552  -0.3064275046   0.1584454380
 0.8809523810  -0.09677419298    0.9596412551  --------------  0.2375690494  -0.04972374572  0.4696132595
 0.7619047619  -0.1935483872     0.5291479826   0.9889503094  -------------- -0.2093023290  -0.02325582557
 0.6666666667   4.064516130E-10  0.2780269051  -0.04419891673 -0.2093023009  --------------  0.1111111030
 0.4642857143  -0.2096774192     0.04035874408  0.2320441936   0.04651164080 -0.2222222123   -------------
```

Frobenius (Euclidian) norm is 425.7675422

Column norm is 503.

Determinant (based on LU decomposition) is 0.9999998788, as stated previously.

Inverting the LU decomposition of A and rounding each element to the nearest integer (using a short program) yielded:

```
A^-1 =


 -9  18  -3   6 -63  32 -13
  2 -13  -5  -8  28 -10  16
  9 -12  16  -2  56 -35   2
  1  -5  -2  -3  11  -4   6
 -3   4  -5   0 -18  11   0
  0  -1  -5   1  -1   3   1
  2   0   2   1   9  -6  -2
```

Frobenius (Euclidian) norm is 112.8671786

Column norm is 186.

Determinant is 1.000000034.

Product of Frobenius norms: 48,055.18123 -- quite consistent with Rodger's condition number of 46,857.55, based on the spectral norms from Singular Value Decomposition (SVD).

Product of column norms: 93,558.

And now, another good word for the HP-15C:

- Matrix inversion is *built-in* as "1/*x*"
- Determinant is *built-in* as "MATRIX 9"
- Frobenius (Euclidian) norm is *built-in* as "MATRIX 8"
- Row norm is *built-in* as "MATRIX 7"
- Column norm is the row norm of its transpose; matrix transposition is *built-in* as "MATRIX 4"
- LU decomposition is part of the process for determinant, inversion, and linear solutions.
- The simple program to round each element of matrix D to the nearest integer (for correction of small calculation errors) is as follows:

```
001  LBL A
002  RCL D (user mode off)
003  RND
004u STO D (user mode on)
005  GTO A
006  RTN
```

```
MATRIX 1
FIX 0
GSB A


-- The HP-15C is designed to skip step 005 of this program when the end of the matrix is reached.


-- The program size is exactly seven bytes (one register), because step 004 is a two-byte instruction.


-- (NOTE:  Such a program is unnecessary on the HP-42S, because "RND" will operate on a matrix.)
```

Thanks to the HP-15C's capability of *in-place* matrix inversion, the user is easily able to perform all the calculations listed in this example on a 7x7 real-valued matrix, along with linear solutions -- even though no more than 64 matrix elements can be stored. In fact, an 8x8 matrix could also be tackled, but without linear solutions.

Detailed, unmodified calculations by Matlab:

```
A = Rodger_7x7;


A


A =


    66    19    63    81    71    64    61
    64    21    61    72    69    62    60
    56    18    53    64    59    54    52
    57    19    56    62    65    57    52
    39    13    36    44    39    37    37
    84    27    78    97    85    80    79
    74    24    72    83    83    73    68


det(A)


ans =


     1
```

```
diary off
col=norm(A,1)


col =


   503


norm(A,2)


ans =


  425.6319


norm(A,'fro')


ans =


  425.7675


cond(A)


ans =


  4.6858e+004


svd(A)


ans =


  425.6319
    9.4255
    5.1109
    0.6869
    0.1898
    0.0412
    0.0091


format long


norm(A,'fro')


ans =


    4.257675422105354e+002


svd(A)


svd_A =


  1.0e+002 *


   4.25631916931461
   0.09425504902503
   0.05110917724593
   0.00686935662741
   0.00189754590938
   0.00041190717276
   0.00009083528627


cond(A)


ans =


    4.685755221285655e+004


B=inv(A)


B =
  Columns 1 through 5


  -8.99999999999717  17.99999999999679  -2.99999999999647   5.99999999999952 -62.99999999998308
   1.99999999999855 -12.99999999999895  -5.00000000000295  -7.99999999999998  27.99999999999182
   8.99999999999771 -11.99999999999670  15.99999999999852  -1.99999999999930  55.99999999998571
   0.99999999999944  -4.99999999999959  -2.00000000000112  -2.99999999999999  10.99999999999684
  -2.99999999999929   3.99999999999896  -4.99999999999960  -0.00000000000021 -17.99999999999557
   0.00000000000002  -1.00000000000022  -5.00000000000029   0.99999999999989  -0.99999999999967
   1.99999999999965   0.00000000000063   1.99999999999996   1.00000000000015   8.99999999999769


  Columns 6 through 7


  31.99999999998978 -12.99999999999935
  -9.99999999999439  16.00000000000055
 -34.99999999999216   1.99999999999839
  -3.99999999999784   6.00000000000020
  10.99999999999759   0.00000000000052
   2.99999999999999   1.00000000000031
  -5.99999999999884  -2.00000000000041


det(B)


ans =


   0.99999999999963


col=norm(B,1)


col =


    1.859999999999504e+002


norm(B,'fro')


ans =
    1.128671785772718e+002

svd(B)

ans =

  1.0e+002 *

   1.10089376169601
   0.24277314553411
   0.05269964721561
   0.01455740405164
   0.00195659576985
   0.00106095112182
   0.00002349447868

cond(B)

ans =

    4.685755221287628e+004

[L,U,P] = lu(A)

L =

  Columns 1 through 5

   1.00000000000000                  0                  0                  0                  0
   0.78571428571429   1.00000000000000                  0                  0                  0
   0.67857142857143  -0.30645161290323   1.00000000000000                  0                  0
   0.88095238095238  -0.09677419354839   0.95964125560538   1.00000000000000                  0
   0.76190476190476  -0.19354838709678   0.52914798206278   0.98895027624313   1.00000000000000
   0.66666666666667                  0   0.27802690582960  -0.04419889502760  -0.20930232558141
   0.46428571428571  -0.20967741935484   0.04035874439462   0.23204419889506   0.04651162790692
  Columns 6 through 7


                  0                  0
                  0                  0
                  0                  0
                  0                  0
                  0                  0
   1.00000000000000                  0
  -0.22222222222227   1.00000000000000


U =


  Columns 1 through 5


  84.00000000000000  27.00000000000000  78.00000000000000  97.00000000000000  85.00000000000000
                  0  -2.21428571428572   1.71428571428572   4.78571428571429   4.21428571428572
                  0                  0   3.59677419354839  -2.35483870967741   8.61290322580646
                  0                  0                  0   0.27055306427504   0.26158445440956
                  0                  0                  0                  0   0.23756906077347
                  0                  0                  0                  0                  0
                  0                  0                  0                  0                  0


  Columns 6 through 7


  80.00000000000000  79.00000000000000
   1.14285714285715  -1.07142857142857
   3.06451612903227  -1.93548387096774
  -0.30642750373693   0.15844544095666
  -0.04972375690605   0.46961325966850
  -0.20930232558138  -0.02325581395349
                  0   0.11111111111114


P =


     0     0     0     0     0     1     0
     1     0     0     0     0     0     0
     0     0     0     1     0     0     0
     0     0     0     0     0     0     1
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     0     1     0     0


format short
[L,U,P] = lu(A)


L =


    1.0000         0         0         0         0         0         0
    0.7857    1.0000         0         0         0         0         0
    0.6786   -0.3065    1.0000         0         0         0         0
    0.8810   -0.0968    0.9596    1.0000         0         0         0
    0.7619   -0.1935    0.5291    0.9890    1.0000         0         0
    0.6667         0    0.2780   -0.0442   -0.2093    1.0000         0
    0.4643   -0.2097    0.0404    0.2320    0.0465   -0.2222    1.0000


U =


   84.0000   27.0000   78.0000   97.0000   85.0000   80.0000   79.0000
         0   -2.2143    1.7143    4.7857    4.2143    1.1429   -1.0714
         0         0    3.5968   -2.3548    8.6129    3.0645   -1.9355
         0         0         0    0.2706    0.2616   -0.3064    0.1584
         0         0         0         0    0.2376   -0.0497    0.4696
         0         0         0         0         0   -0.2093   -0.0233
         0         0         0         0         0         0    0.1111


P =


     0     0     0     0     0     1     0
     1     0     0     0     0     0     0
     0     0     0     1     0     0     0
     0     0     0     0     0     0     1
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     0     1     0     0


>> [U_svd,S_svd,V_svd] = svd(A)


U_svd =
   -0.3937   -0.3715    0.7999   -0.0194   -0.2116   -0.0874   -0.1199
   -0.3762    0.1159   -0.1892   -0.8027   -0.1484   -0.3024    0.2267
   -0.3278   -0.0669   -0.0842    0.2276    0.6892   -0.5859   -0.1045
   -0.3383    0.6020   -0.0212    0.4624   -0.4787   -0.2728    0.0728
   -0.2250   -0.1937   -0.3886   -0.0354   -0.2523    0.0420   -0.8332
   -0.4879   -0.4846   -0.3802    0.2764   -0.0872    0.3071    0.4519
   -0.4392    0.4575    0.1458   -0.1099    0.3973    0.6222   -0.1400


S_svd =


  425.6319         0         0         0         0         0         0
         0    9.4255         0         0         0         0         0
         0         0    5.1109         0         0         0         0
         0         0         0    0.6869         0         0         0
         0         0         0         0    0.1898         0         0
         0         0         0         0         0    0.0412         0
         0         0         0         0         0         0    0.0091


V_svd =


   -0.3993   -0.1002   -0.3032    0.2209    0.4235   -0.2241    0.6784
   -0.1277    0.1045   -0.4919    0.0313    0.3898    0.6973   -0.3028
   -0.3803    0.2117    0.0094    0.2048    0.2998   -0.5413   -0.6211
   -0.4572   -0.6648    0.5050   -0.0018    0.1035    0.2628   -0.1188
   -0.4275    0.6393    0.3939   -0.4210    0.0336    0.1897    0.1976
   -0.3875    0.1665   -0.0881    0.6056   -0.6429    0.1837    0.0220
   -0.3711   -0.2363   -0.4981   -0.6035   -0.3919   -0.1783   -0.0936
```

-- KS

(Added HP-15C LU decomposition)

Edited: 3 May 2008, 10:44 a.m.

Some 12 digit results for the harder 7x7 problem

Message #42 Posted by ***Palmer O. Hanson, Jr.*** *on 4 Apr 2008, 11:13 p.m.,*
in response to message #10 by Palmer O. Hanson, Jr.

This is a continuation of the "harder problem" of a matrix made up of a 7x7 sub-Hilbert multiplied by 360360 to make all of the elements integers and a vector of seven ones in which I examine the results from some HP 12 digit machines . The solutions are presented for Stefan's Matrix Multi-tool program for the HP-35s which uses Gaussian elimination with partial pivoting, for my Eighth Order Linear Equation Solver for the HP-33s whiich uses a method similar to the Broecks program on the HP-67, for the HP-28S using B/A division, and for the HP-28S using B/A division with one iteration of refinement.

```
                             Stefan          Hanson          B/A            Iterated
      Exact                  HP-35s          HP-33s          HP-28S          HP-28S


 1.554001554001554E-04     1.55436746032    1.55486390400    1.55401587783    1.55400222387
-4.195801958041958E-03    -4.19665984724   -4.19785367189   -4.19583345636   -4.19580569077
 3.496503496503496E-02     3.49713257772    3.49802597119    3.49652289869    3.49650455915
-1.282051282051282E-01    -1.28225768337   -1.28255439566   -1.28205714591   -1.28205162195
 2.307692307692307E-01     2.30802850463    2.30851610810    2.30770123782    2.30769285047
-2.E-01                   -2.00026608729   -2.00065461741   -2.00000668114   -2.00000042281
 6.666666666666666E-02     6.66748283464    6.66868090366    6.66668620142    6.66666794681


Approximate Relative Error     2E-04            4E-04            5E-06           3E-07
```

Stefan's program runs for about 75 seconds.

With my program for the HP-33s some calculations are completed after each equation is entered. The calculation time after the first equation is entered is about one second. The calculation times after the remaining six equations are entered are 2, 3, 3, 3, 3, and 3 seconds..

The HP-28S completes the B/A division in about two seconds.

The HP-28S completes the B/A divisioin with one iteration of refinement in about four seconds.

Palmer .

Re: Some 12 digit results for the harder 7x7 problem

Message #43 Posted by ***Stefan Vorkoetter*** *on 5 Apr 2008, 1:05 p.m.,*
in response to message #42 by Palmer O. Hanson, Jr.

Palmer, can you clarify what you mean by a "sub-Hilbert" matrix. First I thought you just meant, for the 7x7 case, the Hilbert matrix multiplied by 360360. However, when I try that, I get totally different answers (which I've also verified using Maple). So perhaps you mean something different by "sub-Hilbert"?

Re: Some 12 digit results for the harder 7x7 problem

Message #44 Posted by ***Palmer O. Hanson, Jr.*** *on 5 Apr 2008, 2:54 p.m.,*
in response to message #43 by Stefan Vorkoetter

> Quote:
>
> Palmer, can you clarify what you mean by a "sub-Hilbert" matrix.

The elements of the sub-Hilberts are defined by $A_{i,j} = 1/(i + j)$

I use sub-Hilberts rather than Hilberts because sub-Hilberts were the test cases we used back in the days of the TI-59 vs HP-41 so-called "friendly competition".

Palmer

Re: Some 12 digit results for the harder 7x7 problem

Message #45 Posted by ***Rodger Rosenbaum*** *on 6 Apr 2008, 5:44 a.m.,*
in response to message #44 by Palmer O. Hanson, Jr.

I've wondered about that also, Palmer.

But why did you use sub-Hilberts back in the day?

Why not just use Hilberts?

Re: Some 12 digit results for the harder 7x7 problem

Message #46 Posted by ***Palmer O. Hanson, Jr.*** *on 7 Apr 2008, 10:55 p.m.,*
in response to message #45 by Rodger Rosenbaum

> Quote:
>
> Why not just use Hilberts?

In the early 1980's we were looking for a matrix problem that was difficult but which was also easy to enter. You have to remember that in those days memory was limited. If one wanted to examine the solution from an HP-67 or a TI-59 to a set of linear equations by multiplying the solution vector by the input matrix and comparing the result with the input vector then one typically had to reenter the input matrix either by hand or with a magnetic card. It got a little easier with the HP-41, but consider how easily we move matrices around in the modern graphic calculators. George Thomson was the "expert" on matrix problems in the TI community back in those days. He wrote

> Quote:
>
> ... The workhorse test marices are the Hilberts. ... ... Their inverses have horrendously huge integers and are available. ... ... The sub-Hilberts with the first row 1/2, 1/3, 1/4, ..., the second row 1/3, 1/4, 1/5, ..., and so on are even harder to invert correctly. I suggest as a guinea pig the 7 x 7 sub-ilbert, with ones on the right hand side: ...

I suspect that it helped a bit that the TI community perceived, rightly or wrongly, that the 7 x 7 sub-Hilbert problem would be difficult for the ten digit HP product line. That would have been consistent with the gamesmanship that was inherent in the so-called "friendly competition." For example, the HP community had proposed the calculation of 1000 digits of pi as a problem for the competition, knowing full well that the problem would be relatively easy for the HP-41 and difficult for the TI-59.

Personally, I had done a lot of polynomial fitting and I believed that the large variations in the Hilberts or the sub-Hilberts tested the capability that I was using.

Palmer

Re: Some 12 digit results for the harder 7x7 problem

Message #47 Posted by ***designnut*** *on 6 Apr 2008, 10:57 p.m.,*
in response to message #42 by Palmer O. Hanson, Jr.

Has anyone run this problem through a 50G? I'd be intrigued to see whether it is more accurate. Sam

Re: Some 12 digit results for the harder 7x7 problem

Message #48 Posted by ***John Keith*** *on 7 Apr 2008, 8:28 p.m.,*
in response to message #47 by designnut

I tried Palmer's sub-Hilbert matrix and vector of 1's as given above on the 50g with the following result (approximate mode):

```
1.5540015563E-4
-4.1958042203E-3
.034965034852
-.12820513102
.23076923964
```

```
-.19999999854
.066666667046
```

Average error: 2.E-9 Execution time: 0.52s

Though some may consider it "cheating", the 50g in exact mode gives the exact (rational) solution to the above system, though that takes about 10 times as long.
Note also that the 50g has a built-in command HILBERT which generates Hilbert matrices, and the following short RPL program will give the sub-Hilbert matrix that Palmer describes:

```
\<< 1 + DUP HILBERT SWAP ROW- DROP 1 COL- DROP \>>
```

According to the Wikipedia Hilbert matrix page, These matrices are "notoriously ill-conditioned", and thus are probably a good choice for testing matrix routines.

John

Re: Some results for a harder 7x7 problem

Message #49 Posted by ***Rodger Rosenbaum*** *on 10 Apr 2008, 9:41 p.m.,*
in response to message #10 by Palmer O. Hanson, Jr.

Try this, Palmer.

You are solving systems with the general form A*X=B.

Let A be your 7th order sub-Hilbert multiplied by 360360 to make its elements integers.

Let the B vector be:

```
[[ 619047. ]
 [ 478907. ]
 [ 394823. ]
 [ 337493. ]
 [ 295451. ]
 [ 263111. ]
 [ 237371. ]]
```

Solve this system; the solution should be [1 1 1 1 1 1 1]T.

Now peturb the 5th element of the B vector so that it becomes 295450, and solve the system. Look how much the solution changes due to a fairly small perturbation in the B vector.

Imagine that the elements of the B vector were physical measurements of some quantity. Getting 6 accurate, repeatable, digits for any physical measurement is difficult, and it's not unreasonable that there might be errors of one ULP (unit in the last place) or more, probably in all the elements, not just one.

Given that there are bound to be errors in measured quantities, it's foolish to try to find an exact solution for a system like this. What one ought to do is find an approximate solution that isn't so highly sensitive to perturbations in the measurements.

The challenge is to find a good way to do so.

The trouble with STO / i

Message #50 Posted by ***Palmer O. Hanson, Jr.*** *on 8 Apr 2008, 4:55 p.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

The STO / i commands at steps 011 and 089 will cause the HP-67 to stop with an "Error" message displayed if the content of the x register is a zero; i.e., a divide by zero will have occurred. Step 011 requires that A_{11} may not be zero. Zeroes at step 089 can be caused by specific combinations of coefficients; for example, if $A_{22} = A_{21} \times A_{12} / A_{11}$. Consider the following problem:

```
  2   -1   -3     1        1

 -2    1    1     3        1

  2   -1    0    -1        1

 -3    2   -1     0        1
```

where the solution 5.0, 8.25, 0.5, 0.75 can be readily obtained with the TI-59 ML-02 program, with the H-41 Math Pac and with others. If you attempt to solve the problem with this HP-67 program you will get a "Error" message during the calculations after the entry of the second equation. If you start the program over again and enter the first row and then the third row you will again get the "Error" message during the calculations after the entry of the second equation. In either case if you press any key the calculator will show that a zero is in the x register. Switching to the W/PRGM mode will show that the machine has stopped at step 089. If you start the program one more time and enter the first equation, followed by the fourth equation, the second equation, and the third equation in that order you will see the correct solution. Similar situations arise with the TI-59 program from *Programbiten* and its derivatives such as my eighth order linear equation solver for the hp-33s in Article 678 The point of all this is that with matrix programs of this kind the user may have to rearrange the equations and start data entry again from the beginning in order to obtain a solution.

Valentin Albillo published a program of this kind for the HP-41 in Richard Nelson's *HP PPC Journal*. His program has the same problems with divide by zero but with an important difference. In his program a restart from the beginning is not required. Rather, his program allows the user to substitute another equation for the one which caused a divide by zero and continue on. The documentation includes an example problem to illustrate the technique. I obtained a copy of that program from Gene Wright some time ago. I can't find it. It may be at my summer home. Perhaps Valentin or Gene can identify the location.

Is it possible that the HP-67 program has a similar capability buried in the code? Someone else will have to tell us.

Palmer

Link to PDF of PPC Journal program

Message #51 Posted by ***Gene Wright*** *on 8 Apr 2008, 9:15 p.m.,*
in response to message #50 by Palmer O. Hanson, Jr.

Link to HP41 program from PPC Journal V7N5

Here you go!

Re: The trouble with STO / i

Message #52 Posted by ***Fernando del Rey*** *on 9 Apr 2008, 6:42 a.m.,*
in response to message #50 by Palmer O. Hanson, Jr.

Palmer,

> Quote:
>
> Is it possible that the HP-67 program has a similar capability buried in the code? Someone else will have to tell us.

The "Alternative" 7x7 Equation Solver I have listed earlier in this thread has precisely the feature you mention. If you get a divide by zero after entering an equation line, you can restart the line with another equation and leave the one that caused the error to be entered later. You don't need to start all over again.

At the bottom of the program listing there are instructions on how to do this.

Fernando

Hi, Fernando !

Message #53 Posted by ***Valentin Albillo*** *on 9 Apr 2008, 7:33 a.m.,*
in response to message #52 by Fernando del Rey

So you have time to post 7x7 solvers but you don't have time to have a go at my S&SMC#20 challenges, though you can do them on your head ?

I'll have a word or two with you next time we talk ... :-)

Best regards from V.

7x7 Lin Eq Solver for the HP-67 converted to HP-41

Message #54 Posted by ***Bill (Smithville, NJ)*** *on 8 Apr 2008, 7:16 p.m.,*
in response to message #1 by Palmer O. Hanson, Jr.

I have run the 7x7 HP-67 program listing through a conversion program that outputs HP-41 source listing (with the card reader module installed). But whenever I load it into V41 emulator, it doesn't run correctly.

Palmer was nice enought to send me a card with the HP-67 program on it. Thanks. I have loaded it into my 41CX and printed the program listing and it exactly matches the output from my concersion program. The 7x7 does run correctly on a real HP-41cx. But it does not when loaded into V41. Not sure what is happening.

Are there any other 41C emulators that will also load the card reader module? I'd like to test the source on another one if possible to see if I get same results.

If anyone wants to try out the conversion program or take a look at the 7x7 program to see why it's not working in V41, just email me thru the HPMuseum.

I know the resulting program does work on a real HP-41CX, just can't figue out what's going on with the emulator.

Appreciate any help.

Thanks,

Bill

[Return to Index | Top of Index]

GTO Go back to the main exhibit hall