HP Forum Archive 18

[Return to Index | Top of Index]

Near, or exactly, rank-deficient matrices.

*Message #1 Posted by **Rodger Rosenbaum** on 26 Jan 2008, 10:30 p.m.*

Just to avoid the apparent confusion about the earlier thread, I'm not describing a problem I'm having and asking for help. This is more of a challenge, but it's not exactly that, either. I'm trying to show methods for dealing with these systems, and it will be more interesting if people play along with me.

But, in particular, I'm trying to show just how capable the HP48G and descendants are for linear algebra problems. Any of these techniques can be done on Matlab or Mathematica or Maple, but that's not what this forum is about.

I'm going to demonstrate techniques that only such a powerful calculator, our well beloved HP48G can do so easily. The lesser calculators won't be able to do some of these with only a few keystrokes, but with a lot of programming, in theory they could follow along.

It's a kind of mirabile dictu sort of thing. If you don't speak Latin, look it up on the web.

I gave a linear system in another thread.

Previously, we had A:

```
[[ 4. 2. 3. 3. ]
 [ 5. 2. 6. 6. ]
 [ 9. 2. 5. 3. ]
 [ 4. 9. 4. 9. ]]
```

and B:

```
[[ 22.1743265837 ]
 [ 33.8228271732 ]
 [ 34.6819580525 ]
 [ 51.0426400707 ]]
```

I asked the reader to solve the system A*X=B

This system has an infinite number of solutions, but there is one solution with the smallest Euclidean norm, which gives a measure of the "size" or "length" of the solution vector. I will use the Euclidean norm whenever I speak of the norm of a solution, the residual of the solution, or the distance between two matrices or vectors. The ABS function on the HP50 calculates this norm. It is the square root of the sum of the squares of the elements. If I use another norm, I will distinguish it.

The LSQ function on the HP50 can find this solution (with flag -54 cleared):

```
[[ 2.01952510743 ]
 [ 2.23362533599 ]
 [ 1.20500296581 ]
 [ 2.00465552820 ]]
```

Now, to move on, consider a slight perturbation of the A matrix. Change the (2,1) element from 5. to 5.00001. The system is no longer rank deficient (singular), and it now has a single exact solution. But it does have a high condition number, about 50 million or so, depending on how you calculate it. This means that a solution calculated with a standard linear solver (SLS), (which as I do it consists of putting B on level of the stack, A on level 1 and pressing the divide key), will incur errors sufficient to lose about 8 to 9 digits of accuracy in the solution.

My HP50 gets:

```
[[ 1.997894004E-5 ]
 [ 5.26288302879 ]
 [ 6.25376578705 ]
 [ -2.37093891693 ]]
```

This is very much different from the minimum norm solution for the unperturbed system.

The LSQ command gets essentially the same solution, with some differences due to the high condition number.

This doesn't seem desireable (for reasons I will give later), that there should be such drastic changes in the solution for such a small perturbation.

Let me emphasize that the mathematically exact solution is essentially the same as the one we got (shown above) for the perturbed system. I'm not saying that it isn't, or that it shouldn't be, this different. After all, the perturbed system is no longer rank deficient, and therefore *has* an exact solution; the exact solution is what it is.

But, perhaps there is an approximate solution with certain more favorable properties.

I have to go now, but I'll be back.

In the meantime, play with the numbers. In particular, see what happens if you add 1% random noise to the A matrix elements and re-solve the system. Test what happens to the residual norm when noise is added to the A matrix, and to the B matrix.

"Ask Marilyn!" (Rank-deficient matrices and the HP-15C)

*Message #2 Posted by **Karl Schneider** on 28 Jan 2008, 3:11 a.m.,*
in response to message #1 by Rodger Rosenbaum

Rodger --

I'm not quite sure where you're going with this -- that is, assuming that your objective is to reveal something that Valentin has not already shown. If you have something "up your sleeve", so to speak, I'd prefer that you just tell us, for I don't have any other ideas.

The solutions of your system lie on an infintely-long line in four-dimensional space. The least-squares solution was the one point on the line having the minimum-magntitude characteristic. Perturbing one or more elements of the system matrix by a small amount will render the matrix non-singular (full-rank), and the unique solution will lie *close to*, but not on, that line. Perturbing the constant vector would have little effect, I'd guess, but the system will be "squirrely", with error from LU decomposition playing a role, so maybe not.

Readers --

In the vein of what we've been discussing, a modest logic/algebra problem was given in the "Ask Marilyn" column of today's (January 27, 2008) *Parade* magazine, which is a Sunday-newspaper insert in the US. A reader submitted the following mild "brain teaser":

> Quote:
>
> *A number has five different digits, none of which is 0: (a) The first plus the second equal the third digit; (b) the third times 2, plus the second, equals the fifth; (c) the second times 2 equals the first; (d) the first times 4 equals the fourth; and (e) the fourth minus the second equals the fifth. What is the number?*

The answer is **21387**. It is best solved by some basic algebraic-equation setup and logic. Discussion and comment is available at http://www.parade.com/articles/editions/2008/edition_01-27-2008/Ask_Marilyn.

Now, if one *were* to set this up as a linear-algebra problem, one would obtain the system

```
       A         N      b


[ 1  1 -1  0  0][n1]   [0]
[ 0  1  2  0 -1][n2]   [0]
[-1  2  0  0  0][n3] = [0]
[ 4  0  0 -1  0][n4]   [0]
[ 0 -1  0  1 -1][n5]   [0]
```

A is singular, with a rank of 4. So, the set of all real-valued possible solutions lie on a line in five-dimensional space. (If A were nonsingular, the only solution would be "all digits zero", which contradicts the problem statement.) However, this is a constrained problem, in which all solution variables are unique integers between 1 and 9, inclusive. It turns out that there is only one valid solution, among the possible Perm(9,5) = 15,120 selections.

If we were given one more piece of useful information -- e.g., "the sum of the digits is 21", we could then substitute

```
[ 1  1  1  1  1] and [21]
```

in place of any row of the 5-equation system, and solve it directly.

Let's make it a bit more interesting, by adding the last equation as a *sixth* equation. It is quite plausbile that the "least-squares" solution to the overdetermined system will be the valid solution, because its elements are all single-digit integers having low magnitudes. Try it, if you like:

```
       A'        N       b'



[ 1  1 -1  0  0][n1]    [0]
[ 0  1  2  0 -1][n2]    [0]
[-1  2  0  0  0][n3] =  [0]
[ 4  0  0 -1  0][n4]    [0]
[ 0 -1  0  1 -1][n5]    [0]
[ 1  1  1  1  1]       [21]
```

Solve $(A'^{T}A')N = A'^{T}\mathbf{b'}$. This can be done on an HP-48/49/50 using "LSQ", or by manual methods on an HP-28, HP-42S, HP-15C, or -- with the right plug-in ROM -- on an HP-41 or HP-71B.

The HP-15C approach will require 60 free (of the 64 available) allocatable memory registers. Here's how to do it:

1. Store A' and **b'** (36 registers in use)
2. Compute and store $A'^{T}\mathbf{b'}$ using "MATRIX 5" to a different matrix (41 registers in use)
3. Clear **b'** (35 registers in use)
4. Compute and store $A'^{T}A'$ using "MATRIX 5" to a different matrix (60 registers in use, but A' is no longer needed)
5. Compute and store the system solution to the matrix holding $A'^{T}\mathbf{b'}$

Due to the HP-15C's storage limitations (expensive RAM in the early 1980's), "MATRIX 5" ($A^{T}B$ function) and *in-place* system-solution and matrix-inversion certainly came in handy for this application:

- Standard matrix multiplication would have been impossible, with A' requiring double storage (RCL MATRIX A, STO MATRIX B, MATRIX 4, RCL MATRIX B, RESULT C, *).
- To store the solution separately from the constant matrix would have required five more registers, when no more than four were available. The augmented system matrix A' would have had to be partially or totally deleted.
- Step 5 could also be solved by inverting $A'^{T}A'$ in place, clearing some or all of A', then multiplying by $A'^{T}\mathbf{b'}$.

Kudos to that stellar HP-15C team!

-- KS

Edited: 28 Jan 2008, 2:46 p.m.

Re: "Ask Marilyn!" (Rank-deficient matrices and the HP-15C)

*Message #3 Posted by **Rodger Rosenbaum** on 29 Jan 2008, 8:22 p.m.,*
in response to message #2 by Karl Schneider

Quote:

Readers --

In the vein of what we've been discussing, a modest logic/algebra problem was given in the "Ask Marilyn" column of today's (January 27, 2008) *Parade* magazine, which is a Sunday-newspaper insert in the US. A reader submitted the following mild "brain teaser":

[/italic]

The answer is **21387**. It is best solved by some basic algebraic-equation setup and logic. Discussion and comment is available at http://www.parade.com/articles/editions/2008/edition_01-27-2008/Ask_Marilyn.

Now, if one *were* to set this up as a linear-algebra problem, one would obtain the system

```
       A         N      b


[ 1  1 -1  0  0][n1]   [0]
[ 0  1  2  0 -1][n2]   [0]
[-1  2  0  0  0][n3] = [0]
```

```
[ 4  0  0 -1  0][n4]   [0]
[ 0 -1  0  1 -1][n5]   [0]
```

A is singular, with a rank of 4. So, the set of all real-valued possible solutions lie on a line in five-dimensional space. (If A were nonsingular, the only solution would be "all digits zero", which contradicts the problem statement.) However, this is a constrained problem, in which all solution variables are unique integers between 1 and 9, inclusive. It turns out that there is only one valid solution, among the possible Perm(9,5) = 15,120 selections.

There are a couple of other ways to solve this problem directly. Notice that even though there are an infinite number of solutions, all of them are scalar multiples of one another, so in a certain sense, there is only one solution. If we find one, we have found them all, to within a scalar multiplicative factor.

Using an HP48G/49G/50G and assuming the system matrix is stored in a variable A, do this:

Type A EGV to get the eigenvectors and eigenvalues. Examine the eigenvalues which are in a vector on stack level 1; one of them will be nearly zero. In this case, they are complex, but on my HP50, the 4th one is (1.005646484E-13,0), close enough to zero. This tells us we want the 4th eigenvector, which is in a matrix on stack level 2. Drop the vector on level 1 and enter the matrix editor to delete all the columns except the 4th. Exit the matrix editor and see on the stack:

```
[[ (.25, 0. ) ]
 [ (.125, 0. ) ]
 [ (.375, 0. ) ]
 [ ( 1. , 0. ) ]
 [ (.875, 0. ) ]]
```

The imaginary parts are zero, so convert it to real. This is one of the solutions to the problem, but we want an all integer solution. It looks like if we multiply by 8 we will get what we want, [2 1 3 8 7]T.

If we form A^TA first, we will get real eigenvectors and eigenvalues, with the same solution.

Another method is this:

Type A SVD DROP SWAP DROP TRN to get only the transpose of the V matrix on the stack. Enter the matrix editor and delete all but the last column. Exit the matrix editor and see on the stack:

```
[[ .177471301883 ]
 [ 8.87356509416E-2 ]
 [ .266206952825 ]
 [ .709885207533 ]
 [ .621149556591 ]]
```

This is again one of the infinite solutions; it is the solution of length 1. Now we need to convert to integers. I would start by dividing by the smallest element in the vector. Enter the matrix editor to extract the 2nd element; press NXT to get to the ->STK function. Exit the matrix editor with this number on the 2nd level of the stack and the vector still on level 1; press SWAP /. See:

```
[[ 2 ]
 [ 1 ]
 [ 3 ]
 [ 8 ]
 [ 7 ]]
```

We lucked out, and that one division gave us the integer solution we were looking for.

All done, without knowing that the sum of the digits is 21.

Eigenvalues and Singular-value decomposition

*Message #4 Posted by **Karl Schneider** on 30 Jan 2008, 1:15 a.m.,*
in response to message #3 by Rodger Rosenbaum

Rodger --

Very good!

Using eigenvalue analysis and singular-value decomposition (SVD) to obtain the answers was enlightening.

I, too, saw the connection to the eigenanalysis formulation, but knowing that the exact eigenvalue was zero, wasn't sure how much help it would be. It appears that the process of computation steered the eigenvector to the desired value.

SVD was introduced to me in a graduate-level linear systems course 11 years ago. The instructor provided supplemental materials for the topic, because it wasn't covered in the text.

As you indicated, these methods aren't available in the RPL-based HP-28 or the RPN-based calc's (of which only the HP-42S would have enough horsepower).

Matlab, however, computed the correct eigenvalues (including zero) and normalized eigenvectors adroitly.

Much of my point of that example was to show that such a calculation could be tackled using a quarter-century-old HP-15C. With its 64 allocatable storage locations, the following linear-algebra and matrix problems can be solved:

- Determinant and norms of an 8x8 real-valued matrix (64 registers)
- Solution of a 7th-order, real-valued, exactly-determined system *with* preservation of the constant vector (63 registers) or *without* preservation (56 registers)
- Closest solution of an overdetermined, inconsistent 5th order real-valued system in six equations, as I demonstrated (60 registers)
- Solution of a 4th-order, complex-valued, exactly-determined system by a special technique (64 registers)
- Solution of a 3rd-order, complex-valued, exactly-determined system by the standard method with preservation of the constant vector (48 registers)

Quite impressive for the era, I'd say...

-- KS

Edited: 1 Feb 2008, 11:26 p.m. after one or more responses were posted

Re: Eigenvalues and Singular-value decomposition

Message #5 Posted by ***Rodger Rosenbaum*** *on 30 Jan 2008, 3:14 a.m.,*
in response to message #4 by Karl Schneider

By the way, remember this from one of your posts?

> Quote:
>
> BTW, flag -54 was indeed clear (the default setting) on my HP-49G when I obtained the least-squares "LSQ" solution to your exactly-determined system with the singular (thus, rank-deficient) matrix:
>
> ```
> 2.01952510743
> 2.23362533599
> 1.20500296581
> 2.00465552820
> ```
>
> This solution, when multiplied by your matrix A, produced your vector B with an error of one ULP on two elements, and was exact on the other two.
>
> I got a very different result from "LSQ" with flag -54 set:

```
673.501526294
-1004.98937644
-1677.5
1456.88232477
```

> This solution, when multiplied by your matrix A, produced a residual from B of 10-10*[263 368 275 493]T.

For that last one I get 10^-10*[223 368 235 753]T on the HP50G.

I checked these on an HP48G and on my HP49G, and I got the same thing I get on the HP50G, not the slightly different results you get. Are you using an HP49G emulator, or a real calculator?

My HP49G has version 1.05 firmware.

I wonder what's causing the small differences.

HP-49G version

*Message #6 Posted by **Karl Schneider** on 1 Feb 2008, 2:18 a.m.,*
in response to message #5 by Rodger Rosenbaum

> Quote:
>
> ---
>
> Are you using an HP49G emulator, or a real calculator?
>
> My HP49G has version 1.05 firmware.
>
> ---

I'm using a real ACO blue HP-49G, puchased in 2002. A few years ago, I verified its ROM as the latest version available. "VER" gives 4.2000031 as the CAS version.

"VERSION" *(thanks, Giancarlo!)* gives

```
"Version HP-49C
Revision #1.18"
"Copyright HP 2000"
```

-- KS

Edited: 1 Feb 2008, 3:00 a.m. after one or more responses were posted

Re: HP-49G version

*Message #7 Posted by **Giancarlo (Italy)** on 1 Feb 2008, 2:25 a.m.,*
in response to message #6 by Karl Schneider

Hi Karl.

> Quote:
>
> ew years ago, I verified its ROM as the latest version available. "VER" gives 4.2000031 as the CAS version.

Wasn't it just the "extended version" :) of the "VER" one, i.e. "VERSION"?
Hope this helps.
Best regards.
Giancarlo

Thanks, Giancarlo! (HP-49G version)

*Message #8 Posted by **Karl Schneider** on 1 Feb 2008, 3:02 a.m.,*
in response to message #7 by Giancarlo (Italy)

Hi, Giancarlo --

Thanks! I have updated my post. My HP-49G is newer than Rodger's.

-- KS

Re: Thanks, Giancarlo! (HP-49G version)

*Message #9 Posted by **Giancarlo (Italy)** on 1 Feb 2008, 3:46 a.m.,*
in response to message #8 by Karl Schneider

Hi Karl.
You're welcome, even though I realize that my quoting from your post was quite meaningless (my fault for not carefully checking which part of the post I selected and what was pasted in my reply...)
However, glad I could be of some help :)

Best regards.
Giancarlo

Re: "Ask Marilyn!" (Rank-deficient matrices and the HP-15C)

*Message #10 Posted by **Rodger Rosenbaum** on 29 Jan 2008, 8:50 p.m.,*
in response to message #2 by Karl Schneider

> Quote:
>
> It is quite plausbile that the "least-squares" solution to the overdetermined system will be the valid solution, because its elements are all single-digit integers having low magnitudes.

It's more than just plausible; it's a must. A "least-squares" solution to an overdetermined system can be exact. Since the LSQ command finds the solution with the least error, if there is a solution with zero error (an exact solution, in other words), the LSQ command will find it. Since we were given that there is an exact solution in integers, and your addition of the last row is another bit of exact information about the solution, the system will have only one "least squares" solution, one where the residual is not just minimized, but actually zero.

Re: "Ask Marilyn!" (Rank-deficient matrices and the HP-15C)

*Message #11 Posted by **Karl Schneider** on 2 Feb 2008, 12:17 a.m.,*
in response to message #10 by Rodger Rosenbaum

Hi, Rodger --

I stated:

It is quite plausbile that the "least-squares" solution to the overdetermined system will be the valid solution, because its elements are all single-digit integers having low magnitudes.

You stated:

> Quote:
>
> It's more than just plausible; it's a must. A "least-squares" solution to an overdetermined system can be exact. Since the LSQ command finds the solution with the least error, if there is a solution with zero error (an exact solution, in other words), the LSQ command will find it.

My statement was made without any analysis of the the rank of the "augmented" system. An effectively-underdetermined system (rank-deficient matrix) would have an infinite number of solutions on a line, plane, or other space, so "LSQ" would return the one with minimum Euclidean/Frobenius norm (rms magnitude). Since the desired solution consisted of single-digit integers, it seemed a good bet that the desired solution would be returned in any case.

The original 5-equation system had a rank of 4, according to Matlab. Now, recall that I had previously calculated only the matrix' zero determinant, not its rank -- what if that had been 3? In fact, I had only *stated without proof* that substituting the "sum of digits" equation for a given equation would allow the system to be solved directly.

Adding the "sum of digits" equation increased the rank from 4 to 5. Augmenting that matrix by the "constants column" $[0\ 0\ 0\ 0\ 0\ 21]^T$ produced a 6x6 matrix still having a rank of 5, because the column was consistent with the matrix. So, the solution produced was the only one possible.

> Quote:
>
> Since we were given that there is an exact solution in integers,

But that is not represented in the system of equations...

> Quote:
>
> and your addition of the last row is another bit of exact information about the solution, the system will have only one "least squares" solution, one where the residual is not just minimized, but actually zero.

That is absolutely correct, but the computational exercise was still worthwhile, I think.

-- KS

Edited: 2 Feb 2008, 12:22 a.m.

Re: "Ask Marilyn!" (Rank-deficient matrices and the HP-15C)

*Message #12 Posted by **Rodger Rosenbaum** on 2 Feb 2008, 8:49 a.m.,*
in response to message #11 by Karl Schneider

I see confusion over terms trying to happen here.

I've consulted a couple of my trusted authorities, "Solving Least Squares Problems", by Lawson and Hanson, and

"Matrix Computations", by Golub and Van Loan.

Golub and Van Loan put it succinctly:

"When there are more equations than unknowns, we say that the system Ax = b is *overdetermined*."

"We say that a system Ax = b is *underdetermined* whenever m < n." (m is the number of rows in the A matrix and n is the number of columns.)

The HP48G User's Manual doesn't say it explicitly, but what they do say on page 14-14 could be interpreted to mean that a system with more equations than unknowns might be, in a sense, considered to be underdetermined under some conditions.

I'm going to reject that and not use the phrase "effectively underdetermined". I'm going to return to my previous long time usage agreeing with Golub and Van Loan. I will use "rank deficient" for the rank deficient case..

You said:

> Quote:
>
> *It is quite plausbile that the "least-squares" solution to the overdetermined system will be the valid solution, because its elements are all single-digit integers having low magnitudes.*

This statement was made after you added a sixth equation, so I assumed you knew that the system was overdetermined. Had adding the sixth row not increased the rank to 5, you would not have called the system overdetermined, right?

My comments:

> Quote:
>
> It's more than just plausible; it's a must. A "least-squares" solution to an overdetermined system can be exact. Since the LSQ command finds the solution with the least error, if there is a solution with zero error (an exact solution, in other words), the LSQ command will find it. Since we were given that there is an exact solution in integers, and your addition of the last row is another bit of exact information about the solution, the system will have only one "least squares" solution, one where the residual is not just minimized, but actually zero.

were not intended to be about linear systems in general; they were intended to apply to the system to which you had added the sixth row.

For an underdetermined system, there are only two possiblilties--no solution, or an infinite number of solutions. So my comments could only apply to an overdetermined system anyway, because an underdetermined system can't have a single exact solution.

You say in the post to which I'm responding:

> Quote:
>
> Now, recall that I had previously calculated only the matrix' zero determinant, not its rank -- what if that had been 3?

The first sentence in this paragraph from the original "Marilyn" post seems to indicate that you *had* calculated its rank.

> Quote:
>
> A is singular, with a rank of 4. So, the set of all real-valued possible solutions lie on a line in five-dimensional space. (If A were nonsingular, the only solution would be "all digits zero", which contradicts the problem statement.) However, this is a constrained problem, in which all solution variables are unique integers between 1 and 9, inclusive. It turns out that there is only one valid solution, among the possible Perm(9,5) = 15,120 selections.

> Quote:
>
> > Quote:
> >
> > Since we were given that there is an exact solution in integers,
>
> But that is not represented in the system of equations...

I don't know what it means to say that it's not "represented", but I trusted that you had set up the equations (I checked them and they looked good to me) so that even if they had multiple solutions, at least one of them would be the all integer solution to Marilyn's problem.

What you say about what if the rank of the original matrix had been 3 is quite true, but it wasn't, and you knew it and so did I.

Edited: 2 Feb 2008, 9:18 a.m.

Rank-deficiency and "determination"

*Message #13 Posted by **Karl Schneider** on 2 Feb 2008, 5:52 p.m.,*
in response to message #12 by Rodger Rosenbaum

Rodger --

Quote:

The first sentence in this paragraph from the original "Marilyn" post seems to indicate that you *had* calculated its rank.

(Karl:) A is singular, with a rank of 4. So, the set of all real-valued possible solutions lie on a line in five-dimensional space.

Touche'! "So long is the thread, I forgot what I said."

In actuality, my "rank of 4" statement was only an educated guess that turned out to be true, after subsequent verification by Matlab.

Quote:

What you say about what if the rank of the original matrix had been 3 is quite true, but it wasn't, and you knew it and so did I.

Please see above. :-)

Quote:

Golub and Van Loan put it succinctly:

"When there are more equations than unknowns, we say that the system Ax = b is overdetermined."

"We say that a system Ax = b is underdetermined whenever m < n." (m is the number of rows in the A matrix and n is the number of columns.)

The HP48G User's Manual doesn't say it explicitly, but what they do say on page 14-14 could be interpreted to mean that a system with more equations than unknowns might be, in a sense, considered to be underdetermined under some conditions.

These questions of terminology and definitions have convinced me to find and purchase another text on linear algebra and matrices, so that another reference with alternative explanations and discussions will be readily available to me. My sophomore-level text gives scanty treatment to the "determination" of linear systems, and my Google search for directly-useful on-line material was not particularly rewarding.

-- KS

Edited: 2 Feb 2008, 6:03 p.m.

Re: Rank-deficiency and "determination"

*Message #14 Posted by **Rodger Rosenbaum** on 2 Feb 2008, 8:12 p.m.,*
in response to message #13 by Karl Schneider

I would highly recommend "Matrix Computations" by Gene Golub and Charles Van Loan. It's not a textbook, but it's in its 3rd edition and it's the modern bible when it comes to how anything is done involving matrix computations. It's also a paperback and an incredible bargain at about $40.

For a text, my favorite is "Linear Algebra and Its Applications", by Gilbert Strang. It's also a modern text, in it's 4th revision I think. It isn't a high powered graduate level text, but it's modern and deals with the orthogonal decompositions that have become so important nowadays, with a very readable style.

Near, or exactly, rank-deficient matrices.

*Message #15 Posted by **Rodger Rosenbaum** on 29 Jan 2008, 10:48 p.m.,*
in response to message #2 by Karl Schneider

> Quote:
>
> Rodger --
>
> I'm not quite sure where you're going with this -- that is, assuming that your objective is to reveal something that Valentin has not already shown. If you have something "up your sleeve", so to speak, I'd prefer that you just tell us, for I don't have any other ideas.
>
> The solutions of your system lie on an infintely-long line in four-dimensional space. The least-squares solution was the one point on the line having the minimum-magntitude characteristic. Perturbing one or more elements of the system matrix by a small amount will render the matrix non-singular (full-rank), and the unique solution will lie close to, but not on, that line. Perturbing the constant vector would have little effect, I'd guess, but the system will be "squirrely", with error from LU decomposition playing a role, so maybe not.

Let me explain further what I'm getting at here. I'm using an HP50 in RPN mode.

Assume the perturbed A matrix (stored in a variable PA) is:

```
[[    4.    2. 3. 3. ]
 [ 5.00001  2. 6. 6. ]
 [    9.    2. 5. 3. ]
 [    4.    9. 4. 9. ]]
```

```
(the unperturbed A matrix has 5. instead of 5.00001 as the 2,1 element)
```

and is the result of some measurements and that we plan to take some measurements like those every day, and the numbers would be similar. We know that the subsequent measurements will be contaminated with noise (and, yes, I'm well aware that there will probably be other sources of error, but let's take one step at a time), and we would like to minimize the effects of that noise.

Use the small program:

Cond << SVL DUP 1. GET SWAP DUP SIZE 1. GET GET / >>

to compute the condition number rather than the built-in COND. Cond gives a more accurate value for the number of accurate digits lost to ill-conditioning.

The condition number of the perturbed matrix PA is about 45E6.

We've solved the perturbed system as it stands with the B matrix:

```
[[ 22.1743265837 ]
 [ 33.8228271732 ]
 [ 34.6819580525 ]
 [ 51.0426400707 ]]
```

and gotten a solution (save in a variable S):

```
[[ 1.997894004E-5 ]
 [ 5.26288302879 ]
 [ 6.25376578705 ]
 [ -2.37093891693 ]]
```

and if we calculate the residual on the HP50, we get [0 0 0 0]T.

Suppose our next set of data is contaminated by noise. On the HP50, type 3.14159265359 RDZ so that our random number generator starts at a repeatable place. Now recall A and type RANM 100 /. We get:

```
[[ .05 -.01 .02 -.08 ]
 [ -.01 .06 -.05 .08 ]
 [ 0. .01 -.09 .03 ]
 [ .05 .05 .06 -.07 ]]
```

Save this in a variable NM (noise matrix), and let the symbol NPA stand for the sum of PA and NM. Type NM PA + S * B - ABS to add the noise (error) matrix to PA, postmultiply by B, and subtract S, thus computing the norm of the residual. I get 1.04328+ on the HP50.

Before we perturbed A, we got a solution of (save it in a variable ULSS, unperturbed least squares solution):

```
[[ 2.01952510743 ]
 [ 2.23362533599 ]
 [ 1.20500296581 ]
 [ 2.00465552820 ]]
```

Calculate the residual norm using the perturbed A matrix *without* additive noise and the solution to the unperturbed system (ULSS) by typing PA ULSS * B - ABS and get .0000201952. This is a small number indicating that the solution to the unperturbed system (ULSS) is actually a pretty good solution to the noise free perturbed system, using residual as a criterion. Since what we're doing is taking new measurements every day and postmultiplying by some solution vector to predict a new B vector, a preferred criterion for goodness of solution is a small residual norm.

But now let's try using ULSS as a solution to the perturbed system to which noise has been added. Type NM PA + ULSS * B - ABS and get .265886+, which is about 4 times less noise than when we use the exact solution S to the perturbed system.

To see why this happens, decompose the process. We have a matrix NPA with additive noise. Show the noise explicitly like this:

```
    [[    4.    2. 3. 3. ]   [[  .05 -.01 .02 -.08 ]
NPA= [ 5.00001  2. 6. 6. ] +  [ -.01 .06 -.05 .08 ]
     [    9.    2. 5. 3. ]    [   0. .01 -.09 .03 ]
     [    4.    9. 4. 9. ]]   [  .05 .05 .06 -.07 ]]
```

In other words, NPA = NM + PA, and when we do NM PA + S * B - ABS, we're doing ((NM S *) + (PA S *)) B - ABS. We can see that the noise contribution is from the NM S * part. Because as we showed above, PA S * B - ABS is almost the same as PA ULSS * B - ABS, that part of the computation doesn't change. But the computation NM S * ABS (the value is 1.04328+) is quite different from NM ULSS * ABS (the value is .26587+); the former is about 4 times the latter.

The two solutions S and ULSS are about equally good from a residual norm point of view (the difference in residual norms is about .00002), but S magnifies any additive noise much more than ULSS.

To emphasize this point, look at the solution vector ULSS:

```
[[ 2.01952510743 ]
 [ 2.23362533599 ]
 [ 1.20500296581 ]
 [ 2.00465552820 ]]
```

and then the solution vector S:

```
[[ 1.997894004E-5 ]
 [ 5.26288302879 ]
```

```
 [ 6.25376578705 ]
 [ -2.37093891693 ]]
```

both give a low residual when postmultiplying PA. But when they multiply the noise matrix, the large elements of opposite sign don't have the balancing effect they do when multplying the system matrix. So when a solution matrix has large elements of opposite sign, as they usually do when exactly solving an ill-conditioned system, those large elements multiply the noise without cancellation, since the noise is addes RMS wise.

The perturbed system is used here to show that when the exact solution to a system with a high condition number is used to predict results (multiply A + noise * S to get a new result B'), the noise can be magnified a lot because of the ill-conditioning. This is a very simple system, made ill-conditioned by perturbing one element of a singular system. In the real world, the system matrix would probably not be ill-conditioned because of a single perturbation, but this simple example shows what's going on.

Here's a real world example of a highly ill-conditioned system that came from the Bureau of the Census:

http://www.itl.nist.gov/div898/strd/lls/data/Longley.shtml

I've just about reached my limit for time spent on this post, so finally, consider the solution vector ULSS. We found it by getting a least squares solution to the matrix A before we perturbed it. How did we know to do that? As it happens, it is better than the exact solution to the perturbed system if our criterion for "goodness" is low magnification of additive noise in the system matrix with low residual norm when applying the solution vector to the noise free perturbed system.

How would we apply Valentin's procedure to get a solution vector for the perturbed system that magnifies noise less that the exact solution?

Edited: 29 Jan 2008, 11:17 p.m.

Re: Near, or exactly, rank-deficient matrices.

*Message #16 Posted by **Antonio Maschio (Italy)** on 30 Jan 2008, 5:54 a.m.,*
in response to message #15 by Rodger Rosenbaum

I guess this is very useful, and that someone should make up an article from all this.

-- Antonio

Re: Near, or exactly, rank-deficient matrices.

*Message #17 Posted by **Rodger Rosenbaum** on 31 Jan 2008, 3:09 a.m.,*
in response to message #15 by Rodger Rosenbaum

To continue, I'll assume the reader has access to an HP48G/HP49/HP50 calculator or emulator, running in RPN mode.

I'll also assume the reader has created a directory to work examples in, containing variables A, B, A1, B1, PA, SS, ULSS, NM, U, V, AND S.

The reader should also review (and install the small programs found in the first reference):

http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv016.cgi?read=90272#90272

http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv016.cgi?read=90258#90258

Store the matrix A in its variable:

```
[[ 4. 2. 3. 3. ]
 [ 5. 2. 6. 6. ]
 [ 9. 2. 5. 3. ]
 [ 4. 9. 4. 9. ]]
```

the matrix B in its variable:

```
[[ 22.1743265837 ]
 [ 33.8228271732 ]
 [ 34.6819580525 ]
 [ 51.0426400707 ]]
```

the matrix PA in its variable:

```
[[    4.    2. 3. 3. ]
 [ 5.00001  2. 6. 6. ]
 [    9.    2. 5. 3. ]
 [    4.    9. 4. 9. ]]
```

the matrix ULSS in its variable:

```
[[ 2.01952510743 ]
 [ 2.23362533599 ]
 [ 1.20500296581 ]
 [ 2.00465552820 ]]
```

the matrix SS in its variable:

```
[[ 1.997894004E-5 ]
 [ 5.26288302879 ]
```

```
 [ 6.25376578705 ]
 [ -2.37093891693 ]]
```

I called this solution vector S in the previous post, but I'm changing it to avoid conflict with the SVD matrix S.

and save the matrix NM in its variable:

```
[[ .05 -.01 .02 -.08 ]
 [ -.01 .06 -.05 .08 ]
 [ 0. .01 -.09 .03 ]
 [ .05 .05 .06 -.07 ]]
```

An important mathematical property to be aware of is this:

If you have N real numbers, whose sum is always T, if the individual numbers vary, the RMS value of the numbers is minimum when the numbers are equal. As their various differences increase, their RMS value increases. Here this particularly applies to the elements of a vector (or matrix). If a vector has 4 elements whose sum is a constant (or nearly a constant, as a practical matter), for example, the norm (calculated with ABS; giving the RMS value of the elements) of the vector is a minimum is the elements are identical.

For instance, let A1 be:

```
[[ 1. 1. 1. 1. ]
 [ 2. 2. 2. 2. ]
 [ 3. 3. 3. 3. ]
 [ 4. 4. 4. 4. ]]
```

and let B1 be:

```
[[ 4. ]
 [ 8. ]
 [ 12.]
 [ 16.]]
```

One obvious exact solution vector for this system is [1. 1. 1. 1.]T. But another exact solution is [12. -10. 12. -10.]T. The sum of the elements in each solution vector is 4, but the norm of the first is 2 and the norm of the second is 22.09+.

If we postmultiply the noise matrix NM by each of these solution vectors and calculate the norm, we get .1319+ for the first and 3.467+ for the second. The second is about 26.3 times the first. It can be shown that the ratio of the noise multiplication factors is about equal to the ratios of the norms of the two solution vectors. The relation doesn't hold exactly except as an average when many numerical experiments with many different noise matrices is conducted, but it's of the right order of magnitude for almost any single noise matrix.

So the idea is to find approximate solutions to a linear system that have an acceptably low residual norm, and a substantially reduced noise multiplication factor. This is done by using the PSU program from the first reference above. First the SVD (Singular Value Decomposition) is computed for the system matrix. Then the singular values are examined and if there are any that are very small in comparison to the largest, they are candidates for removal.

For example, the SVD of the PA matrix gives the following singular values:

[19.769 7.2897 2.8365 4.4034E-7]T (truncated to save space)

Clearly the last (and smallest; the singular values are always in decreasing order in the vector) is a candidate for removal.

Do this in the following way. Put the matrix PA on the stack, then press ->SV giving the three SVD results on the stack (U, V, and S). Store them in the variables S, U, and V. Then type 1 PSU B * to see a solution vector (call it PAS4):

```
[[ 2.01952724285 ]
 [ 2.23362880284 ]
 [ 1.20499966532 ]
 [ 2.00465262032 ]]
```

Calculate the residual norm like this: PA PAS4 * B - ABS. I get 3.35E-6. It's not an exact solution but it's good enough for government work as we used to say in the military. It's probably much better than the precision to which your data are known.

Compare this result to ULSS, the solution of the unperturbed (with singular A matrix) system, found with LSQ. It's the same to the 5th place after the decimal point. The unperturbed A matrix has a non-empty null column space; that's why it's singular. The perturbed A matrix (PA) has a basis vector for its column space that's very close to defining a non-empty null space, and that's why it has such a high condition number. When we solve the unperturbed system with LSQ, it gets a solution that doesn't include the column space null vector. With the perturbed system, even LSQ includes the near null vector. The only way we can get rid of it is with the SVD method, and then we get a solution vector very close to the solution of the unperturbed system when its null column space vector is not included. This is what we want for minimum noise magnification.

And, by the way, Karl, even though the LSQ function was able to solve the unperturbed system, if you try to do it with the Moore-Penrose technique A^T* (AA^T)^-1*B , you will get an error. Does this mean that it's not really underdetermined?

We could try eliminating another singular value in the solution like this: 2 PSU B * and get (call it PAS3):

```
[[ 1.71082483493 ]
 [ 1.87920936888 ]
 [ 1.54600935358 ]
 [ 2.29523695332 ]]
```

If we calculate the residual norm like this: PA PAS3 * B - ABS, we get 1.8418569, much larger than we would like. This shows what we could have known from our examination of the singular values. The 4th was very small and a good candidate for removal; the 3rd was too large.

Let's go back to the 4 ammeter problem.

The A matrix is (store in variable A1):

```
[[ 0.95 1.09 1.01 1.09 ]
 [ 1.94 1.95 2.02 1.91 ]
 [ 2.92 3.00 2.93 2.98 ]
 [ 4.01 3.95 3.97 3.94 ]]
```

and the B matrix is (store in variable B1):

```
[[ 1 ]
 [ 2 ]
 [ 3 ]
 [ 4 ]]
```

Recall A1 to the stack and execute ->SV (from the first reference). Store the stack results to S, V and U (S is on level 1, V on level 2, U on level 3). Examine the singular values which are in variable S: [10.8409016914 .147018600028 7.31158831777E-2 9.49300838352E-3]T

We can definitely delete the effect of the 3rd and 4th, and probably even the 2nd.

Let's compare the exact solution and the 3 solutions we get when we delete the effect of the last 3 singular values, plus the residual in each case (truncated to save space). The singular values are dropped starting with the smallest first, then the last two, then the last three.

	Solution	Residual
Exact	[.8826 3.1449 -.48362 -2.5482]T	1E-12
4th SV dropped	[.29408 .16277 .47708 .07622]T	3.92E-2
3rd & 4th SV dropped	[.40200 .14421 .33214 .13238]T	4.16E-2
2nd, 3rd, 4th SV dropped	[.25209 .25349 .25286 .25197]T	5.4024E-2

Notice how the variance in the elements of the solutions becomes less and less as we drop more singular values.

Remember the minimum RMS property I mentioned above? The residual norms of all these solutions are reasonably low, but the noise magnification factor decreases as the values of the elements become nearly the same. The exact solution is especially bad because of the large values of the elements. Such large values are usually of opposite sign in order to give the correct noise free solution. When noise is present, those large, varying, values increase the norm of the noise in the predicted B vector.

Let's test the noise magnification of the 1st and last solution from above. Postmultiply NM (the noise matrix) by the exact solution above and calculate the norm (ABS function). I get .4072. Now postmultiply NM by the solution labelled "2nd, 3rd, 4th SV dropped", and then calculate the norm. I get 3.338E-2, about 12 times less.

It is a characteristic of these tradeoffs that as the residual norm increases, the noise magnification decreases, so let's calculate the residual norm with some other solution vectors.

Solution	Residual norm
[.25 .25 .25 .25]T	7.818E-2
[.252607010711 .252607010711 .252607010711 .252607010711]T	5.401E-2
[.253280216940 .252112563361 .253205061818 .251830755168]T	5.389E-2

The solution in the first table above, the one labeled "2nd, 3rd, 4th SV dropped", has the largest residual norm, hence the lowest noise magnification. There is no other solution vector with smaller noise magnification properties, which is also equally close to the exact solution.

Of course, these are very small differences. Almost any of the solution vectors with elements near .25 would be satisfactory as a practical matter. Such a solution can be obtained without high powered mathematics; visual inspection of the system is sufficient. But even with this unusually symmetric system, the SVD technique shows its virtue and gives the best solution.

With the first system, the PA system matrix, visual inspection won't help you much. Only the SVD can so easily guide you to the improved solution with reduced noise magnification.

In the old days of statistics, multiple regression analysis would often encounter a very ill-conditioned system. Ill-conditioning usually indicates that some of the columns contain redundant information. One of the techniques they used was to delete a column and perform the regression (least squares analysis), and evaluate the residual norm; then delete a different column or columns, and do it again. After trying all the possible combinations of dropped columns, they would select the configuration with the best residual norm (and with consideration of some other factors). The more modern technique is the SVD method, explained very nicely in Lawson and Hanson's classic book, "Solving Least Squares Problems".