Resolving Stability Issues in Gauss-Jordan Equation Solutions

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2010 pub 09/10 as GuruGram #109
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Back in **GuruGram #77**, we looked at the **Gauss-Jordan Elimination** method of solving linear **nxn** equations. This method is especially computer friendly. It is based on simple and repetitive preprocessing of an equation array to the point where the solution can be observed by inspection. Our own uses of this technique have included our **Magic Sinewave Calculators** and various **Digital Filters**.

Useful and amazingly compact JavaScript solution utilities can be found **here**.

Unfortunately, **the Gauss-Jordan method can lead to severe stability issues for certain equations involving larger values of "n"**. In which the range of solution coefficients can increase exponentially and can easily surpass the quantized math limits of the algorithm in use.

The Problem...

The Gauss-Jordan method repeatedly subtracts two numbers to create a present **coefficient of interest**. It then uses this coefficient of interest to normalize or scale all coefficients that follow. Ideally, the coefficient of interest should be near unity. If it is too small, its normalization will cause extreme magnification of all following coefficients. And **repeated low coefficients of interest that normalize high might cause blowups beyond the quantization capability of the program**.

Surprisingly, **any coefficients of interest that are large compared to unity don't seem to be much of a problem**. Normalizing these creates small values. The next stage subtraction tends to "swallow" its smaller companion. At least in our **Magic Sinewaves**, nearly all of the coefficients often end up much larger than unity.

...and Some Solutions...

Some equations may have such a bad spread of coefficient values that **most any solution may be untenable**, Gauss-Jordan or not. And sometimes, rearranging columns can make a profound difference. But column rearrangement changes the basic equation structure and may create unweildy housekeeping details.

Instead, the best approach to improving stability is often…

> **Initial Gauss-Jordan equation rows may be interchanged in any order without affecting the final result**
>
> **Some sequences are much more likely to cause blowups.**

For instance, even way down at **n=16**, there are well over **10^13** possible ways to prearrange the equation rows. **At least a few of these arrangements should end up significantly less blowup prone**. Fortunately, by progressively finding the "best" row in sequence, the number of blowup avoidance tasks can be reduced to a few hundred or a few thousand.

Here are some tools and techniques that can minimize (but not necessarily eliminate) Gauss-Jordan blowups…

0. Trap out Div0

A missing variable or coefficient being normalized can cause a division by zero and the usual blowup. Very low values should be trapped out and reported. Sometimes, rearranging the column sequence can be of value.

1. Include Coefficient Reports

The JavaScript **alert** command can be extremely useful. Especially for reporting coefficients on entry to **Gauss-Jordan**, on completing the Gauss part, and on exiting the solution. You can use **view source** on **this calculator** for examples. Typical code might be…

> **If (eqns) is the array being manipulated by your Gauss-Jordan routine, use...**
>
> ```
> alert (eqns) ;
> ```
>
> **... as a temporary debugger. Or else something like...**
>
> ```
> form.ExportX.value = eqns  ;
> ```
>
> **... at the beginning, middle, and end to spot blowups.**

2. Avoid 32-bit Math

Floating point math routines of 32-bit resolution are normally limited to **six decimal digits** or so. While they can be extended somewhat using **these techniques**, some 32-bit blowup problems can reasonably be expected beyond **12x12** linear equations.

PostScript normally uses 32-bit math, while **JavaScript** offers full 64-bit calculations. Our **Magic Sinewave calculators** use JavaScript routines.

3. Consider 128-bit Math

This sledgehammer cure lets you **hunt with the big dawgs** instead of staying on the porch. But it takes some really fancy computing and equipment. A full 128-bit floating point package might not be needed. Only the ability to subtract, divide, and deal with 128-bit array values would really be required.

4. Try a Random Rearrangement

Sometimes a simple shuffling of the row sequences will allow a solution that is free of blowups. This is certainly worth a try and may be all that you need. Repeated randomizations can pick the lowest buildup values.

5. Try a Shuffling Alogrithm

Blowup issues were first encountered on our **Magic Sinewaves** around **20** pulses per quadrant, resulting in **40x40** linear equations needing solved. By going to a strange rule of "**alternate the remaining lowest and highest harmonic equation rows**", blowups were eliminated in our **ms calculator**.

Similarly, there might be obvious or strangely non-obvious rearrangements of the equation rows of other problems.

6. Pre-analyze and optimize row-by-row impact

Our final blowup workaround is very much data specific and requires some elaborate preprocessing…

> **If all else fails, test and re-organize the order of all rows for their least impact.**

This crude and unfinished **demo program** shows some of the techniques that may be involved. Candidate rows are selected one by one as the potential final top row. Each positional subtract-and-substitute candidate is then compared for its blowup impact. Any absolute result less than unity is suspect.

A blowup "figure of unmerit" can be calculated by using **abs 1 exch div dup 1 le {pop 0} if** This should produce a number above unity for each problem row, and zero for each well behaved row. Each candidate row can then be square-root-of-the-sums-of-the-squares processed for an overall figure of unmerit.

The lowest score candidate then becomes the new top row. The process gets repeated for as many rows as are needed.

Something like **1/2n^2** subtractions and normalizations may be needed, combined with **n** rms calculations. While not overly excessive, this is far more complex than the **Gauss-Jordan** calculations themselves.

For More Help

Actual working code can be generated on an as-needed or **custom consulting** basis. An expansion of out **magic sinewave calculators** is in the works. Blowup problems are anticipated beyond **n=32** or so with their **64x64** linear arrays needing solution.

Sourcecode for this **GuruGram** appears **here**.

Additional info on similar topics appear on our **Magic Sinewave** and **Math Stuff** library pages. You can also **email us** or call (928) 428-4073 for further help.