Gauss's Algorithm Meets Floating Point Arithmetic

Consider solving the linear system:

$$\begin{aligned} .001x_1 + x_2 &= 1 \\ -x_1 + x_2 &= 0 \end{aligned} \tag{0.1}$$

using Gauss's algorithm. If we multiply the first equation by 1000 and add it to the second equation, we obtain the equivalent system:

$$\begin{aligned} .001x_1 + x_2 &= 1 \\ 1001x_2 &= 1000 \end{aligned} \tag{0.2}$$

from which we obtain

$$x_2 = \frac{1000}{1001} = .999000999000... \tag{0.3}$$

and

$$x_1 = \frac{1 - 1 \cdot x_2}{.001} = \frac{1 - \frac{1000}{1001}}{.001} = .999000999000... \tag{0.4}$$

Now, let's apply the same algorithm but do all of the arithmetic in three decimal digit rounding floating point arithmetic. (I will not specify overflow or underflow levels here to emphasize the fact that neither overflow nor underflow is relevant to the difficulties. Nevertheless, we need to make the modest assumption that .001 does not underflow and 1000 does not overflow.) When we calculate the coefficient of x_2 in the second equation when we are eliminating x_1 we get

$$fl(1 - \frac{-1}{.001} \cdot 1) = fl(1 + 1000) = 1000 \tag{0.5}$$

instead of 1001. When we calculate the right hand side value for the second equation, we get

$$fl(0 - \frac{-1}{.001} \cdot 1) = fl(0 + 1000) = 1000 \tag{0.6}$$

- thus no error at all.

Finally, applying back substitution in floating point, we obtain...

$$\bar{x}_2 = fl(\frac{1000}{1000}) = 1 \tag{0.7}$$

and

$$\bar{x}_1 = fl(\frac{1 - 1 \cdot \bar{x}_2}{.001}) = fl(\frac{1 - 1}{.001}) = 0 \tag{0.8}$$

Thus, $\bar{x}_2$ is calculated quite accurately but $\bar{x}_1$ has 100% error. Clearly the calculation of $\bar{x}_1$ suffered catastrophic cancellation error: the value of $\bar{x}_2$ of 1 is quite accurate but not perfect and the subtraction of this from one in the calculation of $\bar{x}_1$ reveals this slight inaccuracy.

One might pose the question "Yes, the computed solution – at least its first component – is inaccurate, but does the computed solution come close to satisfying the original equations?". We find that

$$\begin{aligned} .001\bar{x}_1 + \bar{x}_2 &= 1 \\ -\bar{x}_1 + \bar{x}_2 &= 1 \neq 0 \end{aligned} \tag{0.9}$$

so the first equation is satisfied but the second has a huge difference. Simply put, the calculated solution (0,1) stinks in every possible fashion.

If we view this in a backward error fashion, we recognize that the original linear system (0.1) has been replaced by a different system, namely:

$$\begin{aligned} .001x_1 + x_2 &= 1 \\ -x_1 + 0 \cdot x_2 &= 0 \end{aligned} \tag{0.10}$$

since the original coefficient of one in the 2,2 position was negligible with respect to 1000 when the first step of the algorithm was applied. The pair (0,1) **is** the exact solution of this new system (0.10) - but (0.10) is not close to the original system (0.1) – that is: we have **not** solved a close problem.

So where is the difficulty?

The difficulty is that the error made when we updated the $\langle 2,2 \rangle$ position in step (0.5) was small relative to the numbers involved (i.e., an error of 1 relative to 1001) but that error of 1 is **huge** compared to the **original** components of the matrix. In fact, 1 is the largest component of the original matrix in absolute value. So a change by 1 renders us with a very different problem to solve – namely (0,10). What caused this was the major increase in the size of numbers with which we were dealing following the use of the tiny value .001 for the pivot. Since the reciprocal of the pivot becomes a multiplier, tiny pivots lead to huge numbers being introduced into the transformed matrix. This suggests that we might avoid such problems by choosing larger pivots.

To see the effect of this idea, we will apply the Gaussian algorithm to the system in (1.1) in the same floating point environment – except that we will reverse the order of the two equations. Now the value of -1 will be the pivot instead of .001. The new system is:

$$\begin{aligned} -x_1 + x_2 &= 0 \\ .001x_1 + x_2 &= 1 \end{aligned} \tag{0.11}$$

When we calculate the coefficient of x_2 in the second equation when we are eliminating x_1 we get

$$fl(1 - \frac{.001}{-1} \cdot 1) = fl(1 + .001) = 1 \tag{0.12}$$

instead of 1.001. When we calculate the right hand side value for the second equation, we get

$$fl(1-\frac{.001}{-1}\cdot 0)=fl(1+0)=1 \tag{0.13}$$

- thus no error at all. Finally, applying back substitution, we obtain...

$$\bar{\bar{x}}_2 = fl(\frac{1}{1})=1 \tag{0.14}$$

and

$$\bar{\bar{x}}_1 = fl(\frac{0-1\cdot\bar{\bar{x}}_2}{-1})=fl(\frac{0-1}{-1})=1 \tag{0.15}$$

If we compare the computed solution $(1,1)$ with the exact solution $(\frac{1000}{1001},\frac{1000}{1001})$ we see that each component has a relative accuracy of about 10^{-3} - an excellent result for a computing environment with unit rounding precision of $5\cdot 10^{-3}$.

In a backwards error sense the second computed solution is the exact solution to the problem:

$$\begin{aligned} -x_1 + x_2 &= 0 \\ 0\cdot x_1 + x_2 &= 1 \end{aligned} \tag{0.16}$$

which is

$$\begin{aligned} 0\cdot x_1 + x_2 &= 1 \\ -x_1 + x_2 &= 0 \end{aligned} \tag{0.17}$$

with the rows reversed. If we compare this to the original system (0,1), we see that the difference is of size .001 in the $\langle 1,1\rangle$ element. Since the other entries of the matrix are of size one, this difference is tiny given the particular floating point environment. The new solution even does a good job of satisfying the original equations, since

$$\begin{aligned} .001\bar{\bar{x}}_1 + \bar{\bar{x}}_2 &= 1.001 \cong 1 \\ -\bar{\bar{x}}_1 + \bar{\bar{x}}_2 &= 0 \end{aligned} \tag{0.18}$$

So what changed?

In the second process the error that was made in updating the $\langle 2,2\rangle$ coefficient was .001 relative to 1.001 – exactly the same relative error as we had with the first process when we had an error of 1 in 1001. But relative to the other numbers in the matrix an error of .001 is tiny and an error of 1 is huge. As said above, the effect of the small pivot is to magnify the size of the numbers being used so that errors small relative to them are actually large with respect to the original coefficients. Large pivots reduce this effect of artificial magnification.

Lastly, let me pose three claims regarding inaccuracies caused by floating point implementations of the Gaussian algorithm:

1. The errors are associated with solving large systems.
2. The errors are associated with making a large number of floating point errors.
3. The errors are associated with solving linear systems with nearly singular matrices (i.e., having rows nearly linearly dependent).

As our example shows, all three are nonsense. We got 100% error in the largest component of the solution when the system was small (i.e., two equations in two unknowns), had a single floating point error (i.e., the update of the $\langle 2,2 \rangle$ component only), and was far being singular. All inaccuracies can be traced to the single cause: a small pivot.