Reflections on Gauss Elimination

George Cain

1. Let’s illustrate some of the computational phenomena associated with the solution of linear systems by Gauss elimination. I have created some Matlab functions to implement Gauss elimination—normally this would be a silly thing to do, as Matlab knows how to solve linear systems using Gauss elimination quite well without any help from me. The functions I have created will, however, allow us to calculate with or without a pivoting strategy, and will let us specify the number of decimal digits to be used in the calculations. First, let’s take a look at these special programs. (As you study these, you will see that I am not an expert programmer.)

The first one of these simply examines a matrix \( a \) and finds a nonzero pivot and makes the pivot row the first row of the matrix:

```matlab
function b=pivot2(a)
    j=1; n=length(a);
    while abs(a(j,1))<=0
        j=j+1;
    end
    q=a(1,1:n);a(1,1:n)=a(j,1:n);
    a(j,1:n)=q;
    b=a;
```

The next is an alternative pivoting program in which the pivot of largest magnitude is found, etc.

```matlab
function c = pivot1(a)
    n=length(a); j=1;
    while j<=n-1
        if abs(a(j,1))-abs(a(1,1))>0
            q=a(1,1:n); a(1,1:n)=a(j,1:n);
            a(j,1:n)=q;
        end
        j=j+1;
    end
    c=a;
```

The next function takes as input a vector \( x \) and a positive integer \( k \) and returns a vector \( y \) in which the entries are the first \( k \) decimal digits of corresponding entries of \( x \). This is used to allow us to specify the number of decimal digits used in the calculations.

```matlab
function y=tr(x,k)
    m=length(x); j=1; k=k-1;
    while j<=m
        ```
s = sign(x(j));
if abs(s)<1
  b(j)=0;
else
  n=floor(log10(abs(x(j))));
  b(j)=s.*floor(abs(x(j)))*10.^(-n+k)*10.^(n-k);
end
j=j+1;
end
y=b;

A couple of examples of what it does:

» x=[12.3456 0 -543.238]
x =
  12.3456   0  -543.2380
» tr(x,2)
ans =
  12   0  -540
» tr(x,4)
ans =
  12.3400   0  -543.2000

This next one finds a pivot row and makes all entries in the first column of the remaining rows zero. The calculations are done using pl decimal digit arithmetic:

function c =reducep(a,pl)
a = pivot1(a);
n=length(a); j=1;
while j<=n-1
  a(j,1:n)=tr(a(j,1:n),pl);
  j=j+1;
end
j=2;
while j<=n-1
  mult = a(j,1)/a(1,1);
  mult=tr(mult,pl);
  a(j,1:n)=a(j,1:n)-mult.*a(1,1:n);
  a(j,1:n)=tr(a(j,1:n),pl);
  j=j+1;
end
Note that the function \texttt{reducep} given uses the function \texttt{pivot1}. We can change our pivoting strategy simply by changing that line to use \texttt{pivot2}.

Next, we have a function takes as input $n \times (n + 1)$ and uses the previously given function \texttt{reducep} to return an upper triangular matrix:

\begin{verbatim}
function b=utriangp(a,pl)
    b=a;
    n=length(a); j=1;
    while j<=n-2
        b(j:n-1,j:n)=reducep(b(j:n-1,j:n),pl);
        j=j+1;
    end

Finally, we put them all together to solve the linear system with the function \texttt{solvp}.

\begin{verbatim}
function x=solvp(a,pl)
    n=length(a); j=1;
    while j<=n-1
        a(j,1:n)=tr(a(j,1:n),pl);
        j=j+1;
    end
    b=utriangp(a,pl);
    j=n-2;
    x(n-1)=b(n-1,n)/b(n-1,n-1);
    x(n-1)=tr(x(n-1),pl);
    while j>=1
        r=b(j,n); k= j+1;
        while k<=n-1
            r=r-b(j,k)*x(k);
            x=tr(x,pl);
            k=k+1;
        end
        x(j)=r/b(j,j);
        x(j)=tr(x(j),pl);
        j=j-1;
    end
\end{verbatim}

The program \texttt{solvp2} is exactly the same as \texttt{solvp} with \texttt{pivot1} replaced by \texttt{pivot2}. Thus \texttt{solvp2}
uses no pivoting strategy. Please note also that there is no check for singularity of the system. This would be a serious breach of decorum if this were to be a "real" program rather than just one to illustrate things with a few examples.

II. Now we are ready to take a look at some examples. First, a system chosen randomly by Matlab:

```matlab
» a=rand(6,7)
a =
 0.8420 0.2746 0.2399 0.6813 0.8456 0.4088 0.9611
 0.1598 0.0030 0.1809 0.3858 0.5901 0.1418 0.1260
 0.2128 0.4143 0.3175 0.3877 0.9554 0.5649 0.1998
 0.7147 0.0269 0.8870 0.4997 0.5561 0.2521 0.3192
 0.1304 0.7098 0.6521 0.1475 0.1482 0.4885 0.6293
 0.0910 0.9379 0.1503 0.5872 0.9833 0.4640 0.1267
```

First, solve using 4 decimal arithmetic and no pivoting:

```matlab
» solvp2(a,4)
ans =
-0.1520 -0.8566 -0.7267 3.1810 -2.3960 3.3180
```

Next, use partial pivoting:

```matlab
» solvp(a,4)
ans =
-0.1441 -0.8470 -0.7240 3.1610 -2.3800 3.2920
```

Quite a difference! Now, we’ll just unleash Matlab to see what the "actual" solution is:

```matlab
» (a(1:6,1:6)
```
```matlab
a(1:6,7))'
ans =
-0.1431 -0.8484 -0.7248 3.1645 -2.3816 3.2932
```

Now let’s look at a nastier example. For the coefficient matrix of our linear system, we shall use the celebrated Hilbert matrix. Here’s the system:

```matlab
a =
 1.0000 0.5000 0.3333 0.2500 0.2000 0.1667 0.1429 0.1000
 0.5000 0.3333 0.2500 0.2000 0.1667 0.1429 0.1250 0.1000
 0.3333 0.2500 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000
 0.2500 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000 0.1000
 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000 0.0909 0.1000
```

4
First, 4 digit arithmetic, no pivoting:

```matlab
» solv2(a,4)
ans =
-4.5420  43.5100  -41.8000  -133.4000  40.0000  357.8000  -262.1000
```

Now, partial pivoting:

```matlab
» solv(a,4)
ans =
-1.3320  12.4500  -10.2100  -38.7600  8.1000  92.5200  -61.1200
```

Wow! This difference should make believers of us all. But what is the “correct” answer? Let’s use more decimal places in our arithmetic. (We’ll use pivoting from now on!) First, just one more:

```matlab
» solv(a,5)
ans =
0.6234  -7.1269  7.3969  25.0140  27.9450  -172.5000  122.3700
```

How about that?! Try some more:

```matlab
» solv(a,8)
ans =
1.0e+03 *
0.0030  -0.1249  1.2620  -5.1280  9.8009  -8.8109  3.0045
```

```matlab
» solv(a,10)
ans =
1.0e+03 *
0.0007  -0.0342  0.3841  -1.7037  3.5085  -3.3640  1.2135
```

These results should make anyone nervous. Who knows what the correct solution might be!

**Question of the week**: What do you make of all this?