Mathematics 307—October 11, 1995

Gauss elimination and row reduction

I recall that a matrix is said to be in row-echelon form if it looks like this:

$$\begin{bmatrix} \# & * & * & * \\ 0 & \# & * & * \\ 0 & 0 & 0 & \# \end{bmatrix}$$

where the # are all non-zero and the $*$ are arbitrary numbers. (**Reduced** row-echelon form means all the $\# = 1$.)

In other words it must saisfy the condition that the successive first non-zero entries in each row are indented.

The process of Gauss elimination applies a sequence of **elementary row operations** to a matrix to get a row-echelon matrix.

There are three types of elementary row operations: (1) swapping two rows; (2) adding a multiple of one row to another; (3) scaling a row. We shall not use type (3) here, at least at first. For example if we have a 2×2 matrix

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

then we can get

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad \begin{bmatrix} a\,u_1 \\ u_2 \end{bmatrix}, \quad \begin{bmatrix} u_1 \\ u_2 + b\,u_1 \end{bmatrix}$$

where $a \neq 0$.

The process works by looking at matrices of fewer and fewer rows. If we are facing an $m \times n$ matrix $M = (m_{i,j})$ then we do these steps:

(1) If the upper left hand entry is non-zero, we let things be for the moment. Otherwise we scan the first column to see if there are non-zero entries anywhere in it. If there are, we swap rows to get the upper left hand entry non-zero. If there are none—if all entries in the first column are 0—we skip the first column, and pass on to the next. In effect, we look at a matrix of a smaller number of columns. We continue until we either get some non-zero column or have ascertained that the whole matrix is zero, in which case we just stop altogether.

(2) We subtract off a multiple of the first row from all the rows beneath it to get 0 entries there.

At this point we are looking at a matrix

$$\begin{bmatrix} \# & * & * & * & * \\ 0 & m_{2,2} & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix}$$

We now pass to consideration of the $(m-1) \times (n-1)$ sub-matrix

$$\begin{bmatrix} m_{2,2} & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

and do these steps all over again as long as there are columns and rows left to look at.

The purpose of this process is usually to solve equations. If we are given a system of m equations in n unknowns

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots a_{1,n}x_n &= y_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots a_{2,n}x_n &= y_2 \\ &\cdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots a_{m,n}x_n &= y_m \end{aligned}$$

then we apply Gauss elimination to the matrix

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & y_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & y_2 \\ \cdots & & & & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & y_m \end{bmatrix}$$

but stopping just before we get to the last column. The point is that *applying an elementary row operations transforms one system of equations into an equivalent system. At the end, the system we have, with a row-echelon coefficient matrix, is relatively simple to solve.*

Applying elementary row operations to a matrix is equivalent to multiplying it on the left by a simple matrix of some kind. For example, the operations done above are performed by left multiplication by

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ b & 0 \end{bmatrix}$$

The way to tell which matrix corresponds to which operation is to apply that operation to the identity matrix.

A permutation matrix is one obtained from the identity matrix by juggling rows around. A row swap corresponds to one of these, and any combination of row swaps will correspond to one also.

A lower triangular matrix is a square matrix with only 0 above the diagonal. The product of any two of these is again one. Adding a multiple of a row above to a row corresponds to multiplication by one of these, and similarly any sequence of such operations.

The consequence of the remark about row operations and left multiplication is

Proposition. *If M is an $m \times n$ matrix we can find a permutation $m \times m$ matrix W and a lower triangular $m \times m$ matrix L such that $U = LWM$ has row-echelon form. The matrix L can be chosen to have all diagonal entries* 1.

If M is $1 \times n$ then this is a simple matter. Similarly if M is $m \times 1$. Suppose M has size $m \times n$ with $m > 1$, $n > 1$. Either the first column of M is all 0, or we can apply operations to get M to have a 1 in the first upper left entry. This will possibly involve a row swap, so if necessary we replace M by $\sigma_{1,j_1}M$, where σ_{1,j_1} swaps rows 1, j_1. Then we subtract off multiples of the first row, which amounts to left multiplication by a matrix like

$$L_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & 0 & 1 & 0 \\ * & 0 & 0 & 1 \end{bmatrix}$$

In the next stage, assuming we do not have a special case, we swap some rows 2, j_2 with $j_2 \geq 2$, multiplying by some σ_{2,j_2}, and left multiply by a matrix that looks like

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & * & 1 & 0 \\ 0 & * & 0 & 1 \end{bmatrix}$$

The total effect is to change the original M to

$$L_2\sigma_{2,j_2}L_1\sigma_{1,j_1}M$$

Now comes a trick. We can write this as

$$L_2\sigma_{2,j_2}L_1\sigma_{2,j_2}\sigma_{2,j_2}\sigma_{1,j_1}M$$

since the product of a swap matrix withitself is I. The matrix

$$\sigma_{2,j_2}L_1\sigma_{2,j_2}$$

turns out to be a lower triangular matrix of the same shape as L_2, since it just swaps elements in the first column of L_2. For example

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}\begin{bmatrix}1&0&0\\a&1&0\\b&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}=\begin{bmatrix}1&0&0\\b&1&0\\a&0&1\end{bmatrix}$$

It works because multiplying on the left by the swap matrix swaps rows 2 and 3, while multiplying on the right swaps columns 2 and 3. We get in order

$$\begin{bmatrix}1&0&0\\b&0&1\\a&1&0\end{bmatrix},\quad\begin{bmatrix}1&0&0\\b&1&0\\a&0&1\end{bmatrix}$$

The product

$$L_2^*=L_2\sigma_{2,j_2}L_1\sigma_{2,j_2}$$

looks like

$$\begin{bmatrix}1&0&0&0\\*&1&0&0\\*&*&1&0\\*&*&0&1\end{bmatrix}$$

Furthermore the product of the swap matrices is a permutation matrix, say W_2.

So the total effect of the first two steps is usually to have

$$L_2^*W_2M$$

look like

$$\begin{bmatrix}\#&*&*&*\\0&\#&*&*\\0&0&*&*\\0&0&*&*\end{bmatrix}$$

We can continue on in this way to get

$$L_m^*W_m^*M$$

in upper echelon form.

In a later section I will explain why this way of doing Gauss elimination is useful.

Programming it

The tricky part above is switching around the σ. In progarmming this becomes a matter of renaming rows rather than actually swapping entries around.

Here is a full program in in the course calculator language. It has one feature, using a **pivot**, which I will explain in a moment. Right now I can say that, roughly speaking, `pivot` returns the row in which there is a non-zero entry if it exists, or the row beyond the last if there is none. But it is fact a bit more complicated.

```
% X r c -> row in which the choice of pivot lies or rX if none

/pivot {
8 dict begin
/c exch def
/r exch def
/X exch def

/rX X length def
/p rX def
/max 0 def
r 1 rX 1 sub {
/cr exch def
/newmax X cr get c get abs def
newmax
max gt
{
/max newmax def
/p cr def
}
if
} for
p

end
} def

% ================================

% X

/gauss {
12 dict begin

/X exch def
/rX X length def
/cX X 0 get length def

% var records the permutation
% initialize it

/perm rX identity def
```

```
/L rX array def
0 1 rX 1 sub {
/i exch def
L i
[
0 1 rX 1 sub {
pop
0
} for
]
put
} for

% look at the upper left corners, [0][0] to start

/r 0 def
/c 0 def
% loop
{

% looking at [r][c]

% halt if r >= rX-1 or c >= cX
r rX 1 sub ge
c cX ge
or
{
exit
}
if

% find the pivot and swap rows of X
/p X r c pivot def % returns the pivot row
% if the pivot row is rX, all zeroes in this column

p rX eq
{
/c c 1 add def
} % if all zeroes, just advance the column
{
% pivot

% first swap rows
X  r p elementswap
perm r p elementswap
L  r p elementswap

% then subtract off

% for rows r+1 to rX-1 ...
r 1 add
```

```
1
rX 1 sub
{ % i
/i exch def

% (for ...  )  ==

X i get % X[i]
c get % X[i][c]
X r get c get div
/rho exch def
X r i rho  % X r i rho
rowsubscale %
% build L embedded in X
L i get r rho put
} for

/r r 1 add def
/c c 1 add def
}
ifelse

} loop
% end of [r][c] loop

perm

L
rX identity
matrixadd

X

end
} def
```

The routine starts with M on the stack, and replaces it by in order) W, L, U.

Choosing a pivot

There is some choice to be made in choosing a row with a non-zero entry in the current column. There may be several such rows. It turns out that it is not good enough to choose any row in which there exists a non-zero entry, but we should choose a row in which the entry is as non-zero—as large in absolute value—as possible. This what the routine `pivot` does. We shall see later why it is necessary.