Solutions to Assignment #5

Problems 4.1, exercise 1: Can we swap rows by adding and subtracting rows, and multiplying by scalars $\lambda = \pm 1$ only? Fiddling around a bit was required for me to get this! I imagine there are other possibilities, but I propose:

$$R_i \leftrightarrow R_j \qquad \iff \qquad \begin{array}{ll} (1) & R_j \leftarrow R_j + R_i \\ (2) & R_i \leftarrow -R_i \\ (3) & R_i \leftarrow R_i + R_j \\ (4) & R_j \leftarrow R_j - R_i \end{array}$$

That is, to do the type I swap operation $R_i \leftrightarrow R_j$, we: do a type III operation, then a type II operation, and then two type III operations. In all cases $\lambda = \pm 1$.

To check my work I have to invent notation for the new rows. In particular, the operations can be re-written this way, denoting the successive new rows with tildes:

$$R_i \leftrightarrow R_j \qquad \iff \qquad \begin{array}{ll} (1) & \tilde{R}_j = R_j + R_i \\ (2) & \tilde{R}_i = -R_i \\ (3) & \tilde{\tilde{R}}_i = \tilde{R}_i + \tilde{R}_j \\ (4) & \tilde{\tilde{R}}_j = \tilde{R}_j - \tilde{\tilde{R}}_i \end{array}$$

Now we can simplify to see that the new jth row is the old ith row, and vice versa:

$$\tilde{\tilde{R}}_j = (R_j + R_i) - \left(\tilde{R}_i + \tilde{R}_j\right) = (R_j + R_i) - (-R_i + (R_j + R_i)) = (R_j + R_i) - (R_j) = R_i,$$

$$\tilde{\tilde{R}}_i = \tilde{R}_i + \tilde{R}_j = (-R_i) + (R_j + R_i) = R_j.$$

Problems 4.1, exercise 3: (*This fact was alluded-to in the lecture, when I said that the inverse of an elementary matrix is an elementary matrix of the same type.*) The inverse of an elementary matrix of type I is that same matrix. The inverse of an elementary matrix of type II has the same form but with "$1/\lambda$" replacing "λ" in the one diagonal entry which is not 1. Finally, the inverse of a type III elementary matrix is of the same form but has "$-\lambda$" replacing "λ" in the one off-diagonal entry which is not 0.

Problems 4.1, exercise 6: A monomial matrix can be built from a product of elementary matrices. Thus its inverse can be built the same way, and thus the inverse exists.

In this course a sufficiently-general example suffices, so here is one. If A is the monomial matrix

$$A = \begin{bmatrix} 0 & 0.1 & 0 \\ -2 & 0 & 0 \\ 0 & 0 & 345.67 \end{bmatrix}$$

then

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 345.67 \end{bmatrix}.$$

Thus A is the product of an elementary matrix of type I times three of type II. (*It is generally true that monomial matrices are products of type I and type II elementary matrices.*) Thus A is invertible, which is to say non-singular.

Computer Problems 4.1, exercise 2: **(b)** In my version of "`Prod`" below, I have chosen to have the procedure itself check the sizes, so that the user does not need supply m and n:

prodELB.m

```
function y = prodELB(A,x)
% PRODELB  Matrix-vector product.  Checks sizes before
%          computing product.  A primitive re-implementation
%          of built-in A*x.  See Kincaid & Cheney CP 4.1 #2b.
% Example: >> A = rand(4,6), x = rand(6,1), y = prodELB(A,x)

[m n] = size(A);
if n ~= size(x,1), error('A and x incompatible sizes'), end

y = zeros(m,1);
for i = 1:m
  for j = 1:n
    y(i) = y(i) + A(i,j) * x(j);
  end
end
```

(c) Equally straightforward:

multELB.m

```
function C = multELB(A,B)
% MULTELB  Matrix-matrix product.  Checks sizes before
%          computing product.  A primitive re-implementation
%          of built-in A*B.  See Kincaid & Cheney CP 4.1 #2c.
% Example:  >> A = rand(4,6), B = rand(6,2), C = multELB(A,B)

[k m] = size(A);
[M n] = size(B);
if m ~= M, error('A and B incompatible sizes'), end

C = zeros(k,n);
for i = 1:k
  for j = 1:n
    for k = 1:m
      C(i,j) = C(i,j) + A(i,k) * B(k,j);
    end
  end
end
```

Notes. We see clearly that a matrix-vector product requires mn multiplications and that a matrix-matrix product requires kmn multiplications, for the sizes here. (*Or is that really true?*)

Regarding the input arguments as listed in Kincaid & Cheney, namely "$\texttt{Prod}(m, n, A, x, y)$" and so on, it might help to know that for a language like Fortran77 or C one needs to supply the sizes as arguments. This fact is because an input "matrix" A to some procedure would actually be just an address to a location in memory, and a code needs to know how to set up the "for" loops which go through all the elements. Also, inputs like x and outputs like y, in "$\texttt{Prod}(m, n, A, x, y)$", can both be supplied as arguments to procedures. Modern languages fix up this situation in various ways. MATLAB/OCTAVE is designed to make numerical linear algebra easy, so things are convenient here.