CISC271
Fall 2005
Homework for week 5
in preparation for quiz 2
Solutions

This homework will give you some practice with reviewing some basic concepts of linear algebra and Gaussian elimination. It will also introduce using Gaussian elimination and LU decomposition in a computer implementations.

1. Recktenwald Chapter 7. questions 2, 3 and 7.

   Note: These are review questions. Solutions will not be posted for the questions from Chapter 7.

2. Recktenwald Chapter 8. questions 21 a) and b) and 23.

   Solution:

   8- 21 a) Here is the inverter I wrote using lutx.

```
function Ainv = InvDR(A)

[L, U, p] = lutx(A);
[n n] = size(A);
I = eye(n);
%permute I according to the permutation vector p.
I = I(p,:);
Ainv = eye(n); %initialize
for col = 1 : n
    y = L\I(:,col);
    Ainv(:,col) = U\y;
end
```

8- 21 b) The lu decomposition is $2n^3/3$ then we do n forward and n backward substitutions using n^2 flops each. Now add to get $8n^3/3$ in all.

8-23 Use a 3 by 3 array XY

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

And letting the vector $z = [z_1 z_2 z_3]^T$ and $a = [a_1 a_2 a_3]^T$, we have the system $XYa = z$, and we need to solve for a.

Here is my m-file

```
function a = PlaneDR(p1,p2,p3)

% Given three points with coordinates (x,y,z) return coefficients a
% such that a1 xi + a2 yi + a3 = zi

XY = eye(3);
XY(:,1) = [p1(1) p2(1) p3(1)]
XY(:,2) = [p1(2) p2(2) p3(2)];
XY(:,3) = [ 1 1 1]'
z = [p1(3) p2(3) p3(3)]'
a = XY\z
```

I plugged in the given values and obtained $a = [-1, -1, 1]^T$

For the given points the $z - coordinatevalue$ is 0.5 for the first 3 and 0 for the last one.

3. Moler Chapter 2. questions 2.1, 2.7, 2.8. and 2.11

Note: Question 2.11 is very similar to question 23 from chapter 8 of Recktenwald.

Solution:

2.1 I set things up so that:

```
A =

     3    12     1
    12     0     2
     0     2     3

 b =

    2.3600
    5.2600
    2.7700
```

Now solving $Ax = b$ I obtained the values apples are 0.29 bananas are 0.05 and cantaloupes are 0.89.

2.7 Here is my determinant m-file called detDR

```
function Det = detDR(A)
%detDR   returns determinant of square matrix A using algorithm
%  on page  86 Q 2.7 of Moler
%   Det = detDR(A)

[n,m] = size(A);
if n ~= m error('matrix is not square'); end

[L,U,p,sig] = lusigDR(A);

Det = sig*prod(diag(U));
```

2.8 I used tic and toc (Do help tic and/or toc to see how they work.) For luDR an n by n random array n=340 took 9.507497 seconds. For lutx I went up to 500 for 8.942710 seconds. For the built in lu function I tried a 2000 by 2000 matrix and that only took 6.087484 seconds to factor. Here is my luDR function.

```
function [L,U,p] = luDR(A)
%LUDR  Triangular factorization, DR version
%   [L,U] = luDR(A) produces a unit lower triangular matrix L,
%   an upper triangular matrix U,
%   so that L*U = A

[m,n] = size (A);
p = [1:n];
if m ~=n error('matrix not square'); end;
for k = 1:n-1

    % Find index of largest element below diagonal in k-th column
   [r,m] = max(abs(A(k:n,k)));
   m = m+k-1;

   % Skip elimination if column is zero
   if (A(m,k) ~= 0)

      % Swap pivot row
      if (m ~= k)
         A([k m],:) = A([m k],:);
         p([k m]) = p([m k]);
      end
      for i = k+1:n
```

```
        % Compute multipliers
        A(i,k) = A(i,k)/A(k,k);

        % Update the remainder of the matrix
        for j = k+1 : n
           A(i,j) = A(i,j) - A(i,k)*A(k,j);
        end
      end
   end
end

% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```

2.11 This is similar to 8.21 above, except I use forward and backsub instead of the backslash.

```
function Ainv = InvDR2(A)

%use forward and backward function instead of \

[L, U, p] = lutx(A);
[n n] = size(A);
I = eye(n);
%permute I according to the permutation vector p.
I = I(p,:);
Ainv = eye(n); %initialize
for col = 1 : n
    y = forward(L, I(:,col));
    Ainv(:,col) = backsubs(U,y);
end

% ------------------------------

function x = forward(L,x)
% FORWARD. Forward elimination.
% For lower triangular L, x = forward(L,b) solves L*x = b.
[n,n] = size(L);
x(1) = x(1)/L(1,1);
for k = 2:n
   j = 1:k-1;
   x(k) = (x(k) - L(k,j)*x(j))/L(k,k);
end
```

```
% ------------------------------

function x = backsubs(U,x)
% BACKSUBS.  Back substitution.
% For upper triangular U, x = backsubs(U,b) solves U*x = b.
[n,n] = size(U);
x(n) = x(n)/U(n,n);
for k = n-1:-1:1
   j = k+1:n;
   x(k) = (x(k) - U(k,j)*x(j))/U(k,k);
end
```