CS1112-202/206/207–Spring 2010

Lab 8 Solution

Friday, March 19

1 Subarrays

```
m= rand(6,5)
>> This creates a 6x5 matrix of random numbers between 0 and 1.

a= m(:,2) % What does the colon specify when used in place of an index? ____
>> This gives all the rows of the second column.  The colon says 'all.'

b= m(2:3,:) % ____________________________________________________________
>> This gives the second and third rows all columns, i.e., the whole
   second and third rows.

p= rand(6,5,3) % This is a 3-dimensional array
>> This creates a 6x5x3 3-d array of random numbers between 0 and 1.
   For simplicity, let's call the third dimension 'page.'
   That is, each page is a matrix of 6 rows and 5 columns.

c= p(:,:,2) % Is this a matrix (2-d) or a 3-d array? ________________________
>> This gives all rows and all columns of page 2 of c, which is a matrix,
   i.e., a 2-d array.

d= p(4,:,2) % Is this a vector, matrix, or 3-d array? ______________________
>> This gives the fourth row and all columns of page 2.  Since page 2 is
   a matrix, the fourth row of that is a vector (1-d array).
```

2 Two-Dimensional Interpolation

This function is easy to implement if we know how to access particular parts of a matrix efficiently. To recap, we first need to insert a column between any two adjacent columns of the original matrix. Since there is no way to literally "insert" a column or a row, we need to allocate a new matrix with the new number of columns. Then, we copy the values of the original matrix into the new one, leaving a blank column in between. In effect, we create a blank column between two adjacent columns of the original matrix. Now that we have blank columns in between, for each of them, we can take the average of the columns to its left and its right to obtain the value in that column itself. Finally, we do the same in the other orientation—insert a blank row between any two adjacent rows of the *intermediate* matrix, and then take the average of the rows just above and below to obtain the value of the new, blank rows.

2.1 Vectorized Version

One solution is posted on the course webpage. Another solution is attached below. Instead of creating an intermediate matrix, this version creates the final matrix right away and processes as if the matrix were the intermediate one when "calculating odd rows."

```
function newM = Interpolate2D(M)
% Perform 2-d interpolation on the real-valued data in nr-by-nc matrix M.
% The interpolated data are added between existing data points so newM is
% (2*nr-1)-by-(2*nc-1). Use the simple average as the interpolated value.

% get size of M
[nr,nc]=size(M);
% determine final size
nnr=2*nr-1;
nnc=2*nc-1;
newM=zeros(nnr,nnc);

% assign original values to odd rows, odd columns
newM(1:2:nnr,1:2:nnc)=M(:,:);

% calculate odd rows, even columns
newM(1:2:nnr,2:2:nnc-1)=(newM(1:2:nnr,1:2:nnc-2)+newM(1:2:nnr,3:2:nnc))/2;

% calculate even rows
newM(2:2:nnr-1,:)=(newM(1:2:nnr-2,:)+newM(3:2:nnr,:))/2;
```

2.2 Non-Vectorized Version

Once again, we present a solution that does not create an intermediate matrix.

```
function newM=Interpolate2D(M)
% Perform 2-d interpolation on the real-valued data in nr-by-nc matrix M.
% The interpolated data are added between existing data points so newM is
%  (2*nr-1)-by-(2*nc-1).  Use the simple average as the interpolated value.

[nr,nc]=size(M);
nnr=2*nr-1;
nnc=2*nc-1;
newM=zeros(nnr,nnc);

% assign original values to odd rows, odd columns
for i=1:nr
    for j=1:nc
        newM(2*i-1,2*j-1)=M(i,j);
    end
end
```

```
% calculate odd rows, even columns
for i=1:2:nnr
    for j=2:2:nnc-1
        newM(i,j)=(newM(i,j-1)+newM(i,j+1))/2;
    end
end

% calculate even rows
for i=2:2:nnr-1
    for j=1:nnc
        newM(i,j)=(newM(i-1,j)+newM(i+1,j))/2;
    end
end
```

Observe the resemblance between vectorized and non-vectorized versions. Remember that if we can write a vectorized code, we can always write one in a non-vectorized version.