Matrix Manipulation Functions

• We know that a specialty of MATLAB is working with matrices
• Early on in this course we investigated the colon operator as a basic tool for matrix manipulation
• In this section additional functions for matrix manipulation are be studied

Rotation

• A matrix can be rotated in the counter clockwise direction using the function rot90
  – \textit{rot90} (A) rotates A by $90^\circ$ counterclockwise
  – \textit{rot90} (A, n) rotates A by $n \cdot 90^\circ$ counterclockwise

  » A = [1 2; 3 4]
  A =
  1  2
  3  4
  » [rot90(A) rot90(A,2)]
  ans =
  2  4  4  3
  1  3  2  1

Flipping

• A matrix function that finds uses in signal processing is the ability to flip a matrix either from left-to-right, \textit{fliplr} (A), or from up-to-down, \textit{flipud}
\[
A = \begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]
\[
\text{flip}(A) \quad \text{flipud}(A)
\]
\[
\begin{bmatrix}
2 & 1 & 3 & 4 \\
4 & 3 & 1 & 2
\end{bmatrix}
\]

**Reshaping**

- The \(m \times n\) size of a matrix can be reshaped using \texttt{reshape(A,m,n)} so long as the product \(m \cdot n\) is held constant

\[
B = \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8
\end{bmatrix} \% 2 \times 4 = 8
\]
\[
\text{reshape}(B,4,2) \% 4 \times 2 = 8, \text{ taken in columns from } B
\]
\[
\begin{bmatrix}
1 & 3 \\
5 & 7 \\
2 & 4 \\
6 & 8
\end{bmatrix}
\]
\[
\text{reshape}(B,1,8) \% 1 \times 8 = 8, \text{ taken in columns from } B
\]
\[
\begin{bmatrix}
1 & 5 & 2 & 6 & 3 & 7 & 4 & 8
\end{bmatrix}
\]

**Extraction**

- Extraction of rows, columns, or sub matrices is easily accomplished using the colon operator
• To extract elements of other configurations we may use three additional functions defined in terms of the matrix main diagonal

\[
A = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

Diagonal \(k = 1\)

Main Diagonal, \(k = 0\)

Diagonal \(k = -1\)

\[
A_u = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  0 & a_{22} & a_{23} \\
  0 & 0 & a_{33}
\end{bmatrix}
\]

\[
A_l = \begin{bmatrix}
  a_{11} & 0 & 0 \\
  a_{21} & a_{22} & 0 \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

Upper Triangular Form

Lower Triangular Form

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diag(A), diag(A,k)</td>
<td>With (k = 0) or using (\text{diag}(A)) extracts a vector containing the main diagonal of (A). (k &gt; 0) extracts diagonal above main ((k = 0)) diagonal or for (k &lt; 0) extracts diagonal below main.</td>
</tr>
<tr>
<td>triu(A), triu(A,k)</td>
<td>With (k = 0) or using (\text{triu}(A)) extracts the upper triangular matrix of (A). (k &gt; 0) extracts above main ((k = 0)) diagonal or for (k &lt; 0) extracts diagonal below main.</td>
</tr>
<tr>
<td>tril(A), tril(A,k)</td>
<td>Similar to (\text{triu}(A,k)) except the corresponding lower triangular matrix is extracted.</td>
</tr>
</tbody>
</table>
Examples:

» C = [1 2 3; 4 5 6; 7 8 9]
C =
\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

» diag(C,0)
ans =
\[
\begin{bmatrix}
1 \\
5 \\
9
\end{bmatrix}
\]

» [diag(C,1) diag(C,-1)]
ans =
\[
\begin{bmatrix}
2 & 4 \\
6 & 8
\end{bmatrix}
\]

» [C triu(C) tril(C)]
ans =
\[
\begin{bmatrix}
1 & 2 & 3 & 1 & 2 & 3 & 1 & 0 & 0 \\
4 & 5 & 6 & 0 & 5 & 6 & 4 & 5 & 0 \\
7 & 8 & 9 & 0 & 0 & 9 & 7 & 8 & 9
\end{bmatrix}
\]

» [C triu(C,1) tril(C,1)]
ans =
\[
\begin{bmatrix}
1 & 2 & 3 & 0 & 2 & 3 & 1 & 2 & 0 \\
4 & 5 & 6 & 0 & 0 & 6 & 4 & 5 & 6 \\
7 & 8 & 9 & 0 & 0 & 0 & 7 & 8 & 9
\end{bmatrix}
\]

Example: Practice! p. 115 (4, 6, 10)

Determine the matrices generated by the following function references. Then check your answers using MATLAB. Assume that A and B are the following:
$A = \begin{bmatrix} 0 & -1 & 0 & 3 \\ 4 & 3 & 5 & 0 \\ 1 & 2 & 3 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 & 5 & 0 \\ 3 & 6 & 9 & 12 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$

\[
\text{flipud(fliplr}(B)) = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \\ 12 & 9 & 6 & 3 \\ 0 & 5 & 3 & 1 \end{bmatrix}
\]
6. `reshape(A,6,2)`

To begin with A is a $3 \times 4$ matrix. The reshape operation converts it to a $6 \times 2$ matrix (note the product is a constant as it should be). Since reshape works by using the columns of the original matrix to sequentially fill the new matrix, it follows that

$$\text{reshape}(A, 6, 2) = \begin{bmatrix}
0 & 0 \\
4 & 5 \\
1 & 3 \\
-1 & 3 \\
3 & 0 \\
2 & 0
\end{bmatrix}$$

```
>> reshape(A,6,2)
ans =
     0     0
     4     5
     1     3
    -1     3
     3     0
     2     0
```

10. `triu(B,-1)`

The upper triangular matrix extracted from B is everything above and including the diagonal $k = -1$, that is everything from the diagonal just below the main diagonal on up. Zeros are inserted elsewhere. Thus we expect to obtain
Looping Structures

• The ability to repeat a set of statements with certain parameters changing is a feature of all modern programming languages

• MATLAB provides the for loop and the while loop for performing such operations, with the caution that vectorizing algorithms is preferred since loops slow down program execution

The for Loop

• The basic structure of a for loop is the following

```matlab
for index=start_value:stop_value
    % ...
    % Code to be executed repeatedly
    % ...
end %This marks the end of the index for loop
```
• Often times we design a for loop to simply run over the index values of a vector we wish to operate on

**Example:** A for loop version of the rectangular pulse function

```matlab
function x = rect_loop(t)
    % RECT x = rect_loop(t): A function that is defined to be 1 on [-0.5,0.5] and 0 otherwise. This implementation uses a for loop which is inefficient.
    
    % Initialize a vector with zeros in it that is the same length as the input vetor t:
    x = zeros(size(t));
    for k=1:length(t)
        if abs(t(k)) <= 0.5
            x(k) = 1;
        end
    end
end
```

• Compare execution times for the two functions using tic and toc, making sure to run both functions at least twice so that the routines may be converted to fast binary form in the MATLAB workspace (the machine is a P120 running Win95)

```matlab
» t = -10:.001:10; %Second pass results are below
» length(t)
ans = 20001
» tic; x_vec = rect(t); toc;
elapsed_time = 0.060000000000000 % Time in seconds
» tic; x_loop = rect_loop(t); toc;
elapsed_time = 1.320000000000000 % Time in seconds

- **Note:** Once the two functions are ‘compiled’ to a fast binary form, the vector version of rect runs 1.32/0.06 = 22 times faster than the loop version!
```
The for loop can be used in different ways depending upon how the loop indexing is defined.

In general loop control is accomplished using

```
for index=expression
```

where `expression` may be a scalar, vector, or a matrix

- **scalar case**: Here the loop executes just once with `index` equal to the scalar value.
- **vector case**: This is the most common, with the vector often taking on integer values or of the form
  ```
  start_val:step_size:final_val
  ```
- **matrix case**: This is very non traditional as far as high-level programming languages is concerned, but if used `index` becomes a column vector iterating over the columns of the expression matrix.

The while Loop

Another useful looping structure is the while loop, which loops while some condition is true.

The while code block is of the form

```
while expression
    loop_statements
end
```

**Example:** Finding \( N = 2^y \geq x > 0 \), that is the nearest power of two is greater than or equal to some positive number.

```
function N = pow2up(x)
    % POW2UP N = pow2up(x) finds the power of 2
    % that is equal to or greater than x.
```
N = 1;
while x > N
    N = 2*N;
end

• Testing the function:
  » [pow2up(1) pow2up(15) pow2up(56) pow2up(1026)]
  ans =
        1      16      64    2048

• In most high-level languages loops must be used in order to solve problems

• In writing MATLAB programs we strive to eliminate loops if at all possible

• Sometimes loops cannot be avoided

Example: Practice! p. 117 (1,5)

Determine the number of times the following for loops will be executed. Check your answers using the MATLAB length function on the loop expression or simply place a counter in the loop that is initialized to zero before entering the for block.

1. for k = 3:20
   This for loop has k running from 3 to 20 in steps of 1, thus the number of iterations is \((20 - 3) + 1 = 18\)
   » length(3:20)
   ans = 18
   » count = 0;
   » for k = 3:20,
      count = count + 1;
   end
   » count
count = 18
5. for time = 10:5
   This for loop has time running from 10 down to 5 using the step size of 1, thus the number of iterations is 0, since there is no way make 10 = 5 by adding multiples of 1 to it.
   » length(10:5)
   ans = 0
   » count = 0;
   » for k = 10:5,
       count = count + 1;
   end
   » count
   count = 0