Vectors/matrices

As suggested by its name (a contraction of "Matrix Laboratory"), MATLAB can create and manipulate arrays of 1 (vectors), 2 (matrices), or more dimensions. In the MATLAB vernacular, a vector refers to a one dimensional (*1×N* or *N×1*) matrix, commonly referred to as an array in other programming languages.

A matrix generally refers to a 2-dimensional array, i.e. an *m×n* array where m and n are greater than 1. Arrays with more than two dimensions are referred to as multidimensional arrays. Arrays are a fundamental type and many standard functions natively support array operations allowing work on arrays without explicit loops. Therefore the ***MATLAB*** language is also an example of array programming language.

A simple array is defined using the syntax: ***`init:increment:terminator`***.

For instance, the following code defines a variable named array (or assigns a new value to an existing variable with the name array) which is an array consisting of the values 1, 3, 5, 7, and 9.

```
>> array = 1:2:9
array =
 1  3  5  7  9
```

That is, the array starts at 1 (the ***init*** value), increments with each step from the previous value by 2 (the increment value), and stops once it reaches (or to avoid exceeding) 9 (the terminator value).

```
>> array = 1:3:9
array =
 1  4  7
```

the increment value can actually be left out of this syntax (along with one of the colons), to use a

default value of 1.

```
>> ari = 1:5
ari =
 1 2 3 4 5
```

assigns to the variable named ***ari*** an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the incrementer.

Indexing is “one-based”,which is the usual convention for matrices in mathematics, although not for some programming languages (the alternative being “zero-based”).

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row.
The list of elements should be surrounded by square brackets: [].
Parentheses: () are used to access elements and subarrays (they are also used to denote a function argument list).

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
 16  3  2 13
  5 10 11  8
  9  6  7 12
  4 15 14  1

>> A(2,3)
ans =
 11
```

Sets of indices can be specified by expressions such as "2:4", which evaluates to [2, 3, 4]. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>> A(2:4,3:4)
ans =
 11 8
 7  12
 14 1
```

A square identity matrix of size n can be generated using the function ***eye,*** and matrices of any

size with zeros or ones can be generated with the functions ***zeros*** and ***ones,*** respectively.

```
>> eye(3)
ans =
 1 0 0
 0 1 0
 0 0 1

>> zeros(2,3)
ans =
 0 0 0
 0 0 0

>> ones(2,3)
ans =
 1 1 1
 1 1 1
```

Most MATLAB functions can accept matrices and will apply themselves to each element. For example, ***mod(2*J,n)*** will multiply every element in "J" by 2, and then reduce each element modulo "n".

MATLAB does include standard "for" and "while" loops, but (as in other similar applications such as R), using the vectorized notation often produces code that is faster to execute.

This code, excerpted from the function ***magic.m***, creates a magic square M for odd values of n (MATLAB function ***meshgrid*** is used here to generate square matrices I and J containing 1:n).

```
[J,I] = meshgrid(1:n);
A = mod(I+J-(n+3)/2,n);
B = mod(I+2*J-2,n);
M = n*A + B + 1;
```

Matrix Multiplication

The transpose operator

Transposing a vector changes it from a row to a column vector and vice versa.
The transpose of a matrix interchanges the rows with the columns. Mathematically, the transpose of A is represented as A^T .

In Octave an apostrophe performs this operation:

```
>> A
A =
5  7 9
-1 3 -2

>> A'
ans =
5 -1
7 3
9 -2

>> A*x'
ans =
32
-7
```

Building composite matrices

It is often useful to be able to build matrices from smaller components, and this can easily be done using the basic matrix creation syntax:

```
>> comp = [eye(3) B;
A zeros(2,2)]
comp =
1  0  0  2  0
0  1  0  0 -1
0  0  1  1  0
5  7  9  0  0
-1  3 -2  0  0

>> comp = [eye(3) B;A zeros(2,2)]
comp =
1  0  0  2  0
0  1  0  0 -1
0  0  1  1  0
5  7  9  0  0
-1  3 -2  0  0
```

You just have to be careful that each sub-matrix is the right size and shape, so that the final composite matrix is rectangular.
Of course, Octave will tell you if any of them have the wrong number of row or columns.

Table of Basic matrix functions and decompositions

eye	Create an identity matrix
zeros	Create a matrix of zeros
ones	Create a matrix of ones
rand	Create a matrix filled with random numbers
diag	Create a diagonal matrix, or extract the diagonal of the given matrix
inv	Inverse of a matrix
det	Determinant of a matrix
trace	Trace of a matrix
eig	Calculate the eigenvectors and eigenvalues of a matrix
rank	Calculate an estimate of the rank of a matrix
null	Calculate a basis for the null space of a matrix
rref	Perform Gaussian elimination on an augmented matrix
lu	Calculate the LU decomposition of a matrix
qr	Calculate the QR decomposition of a matrix
svd	Calculate the SVD of a matrix
pinv	Calculate the pseudoinverse of a matrix