Basic Matrix Operations

This example shows basic techniques and functions for working with matrices in the MATLAB® language.

```
% Copyright 1984-2012 The MathWorks, Inc.
% These are my first notes
% Slight modifications by HA (30.10.2016)
```

First, let's create a simple vector with 9 elements called a.

```
a = [1 2 3 4 6 4 3 4 5]
```

```
a =

     1     2     3     4     6     4     3     4     5
```

Now let's add 2 to each element of our vector, a, and store the result in a new vector.

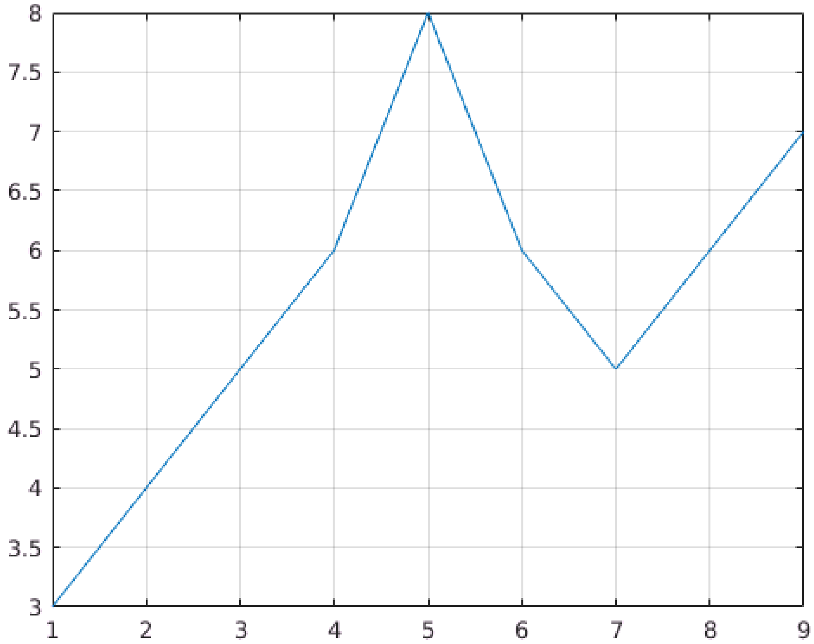
Notice how MATLAB requires no special handling of vector or matrix math.

```
b = a + 2
```

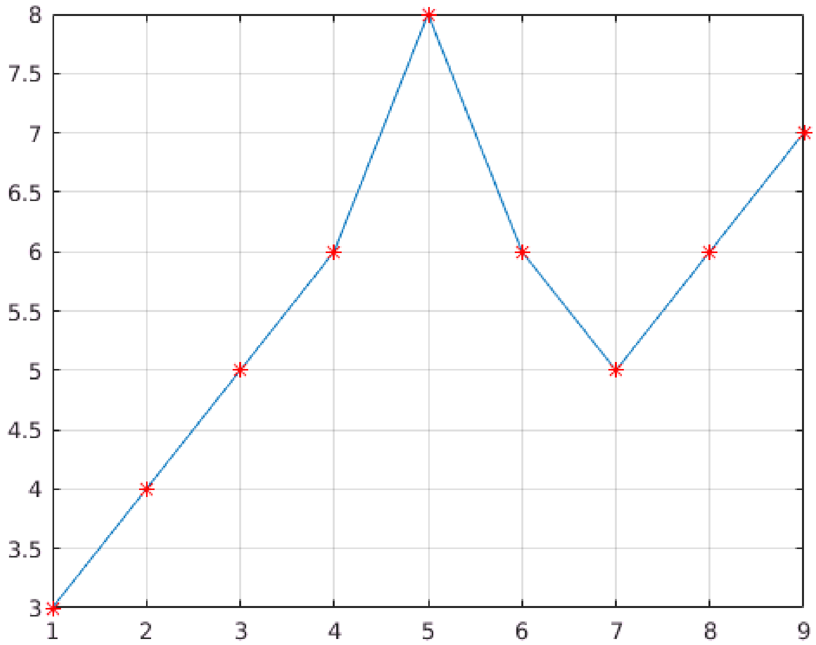
```
b =

     3     4     5     6     8     6     5     6     7
```

Creating graphs in MATLAB is as easy as one command. Let's plot the result of our vector addition with grid lines.

```
plot(b)
grid on
```

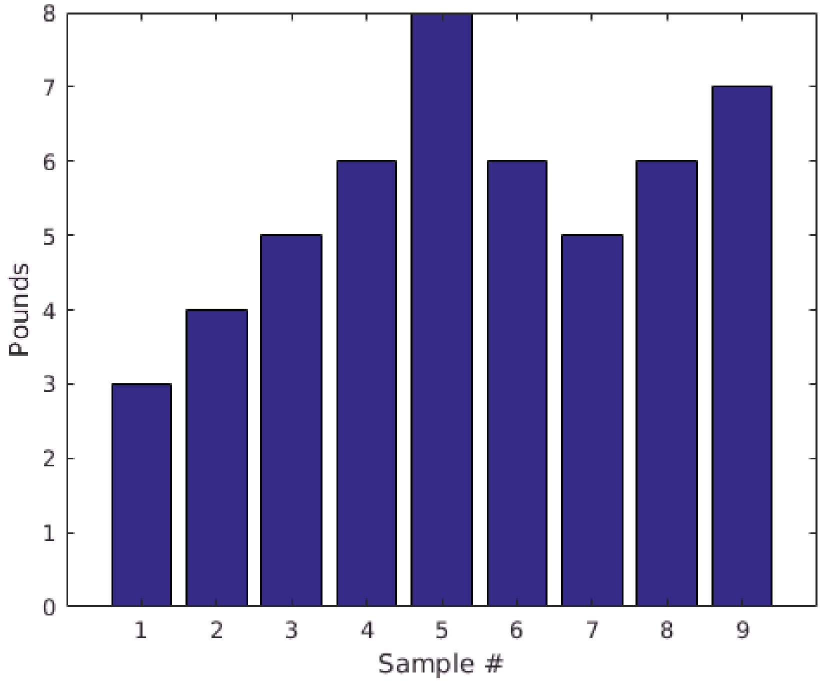



```
hold on
plot(b,'r*')
```

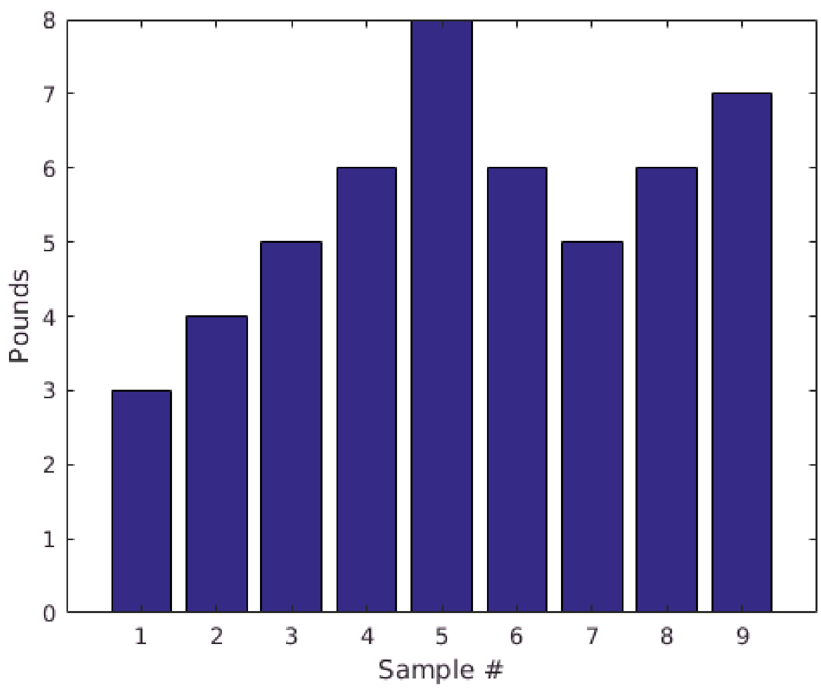


MATLAB can make other graph types as well, with axis labels.

```
hold off
```

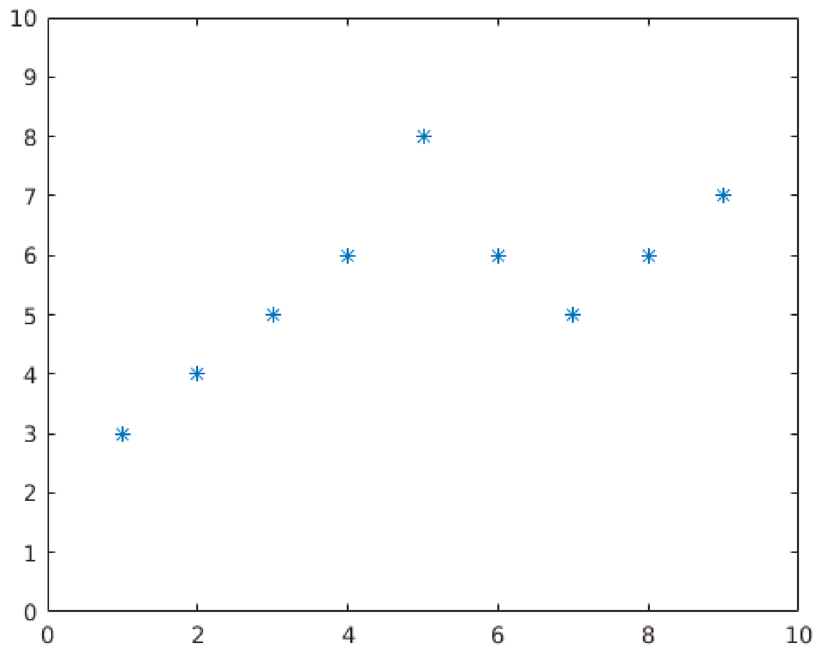



```
bar(b)
xlabel('Sample #')
ylabel('Pounds');shg
```



MATLAB can use symbols in plots as well. Here is an example using stars to mark the points. MATLAB offers a variety of other symbols and line types.

```
plot(b,'*')
axis([0 10 0 10]);
```




```
shg
```

One area in which MATLAB excels is matrix computation.

Creating a matrix is as easy as making a vector, using semicolons (;) to separate the rows of a matrix.

```
A = [1 2 0; 2 5 -1; 4 10 -1]
```

```
A =

     1     2     0
     2     5    -1
     4    10    -1
```

We can easily find the transpose of the matrix A.

```
B = A'
```

```
B =

     1     2     4
     2     5    10
     0    -1    -1
```

Now let's multiply these two matrices together.

Note again that MATLAB doesn't require you to deal with matrices as a collection of numbers. MATLAB knows when you are dealing with matrices and adjusts your calculations accordingly.

```
C = A * B
```

```
C =

     5    12    24
    12    30    59
    24    59   117
```

```
[A B C]
```

```
ans =

     1     2     0     1     2     4     5    12    24
     2     5    -1     2     5    10    12    30    59
     4    10    -1     0    -1    -1    24    59   117
```

Instead of doing a matrix multiply, we can multiply the corresponding elements of two matrices or vectors using the .* operator.

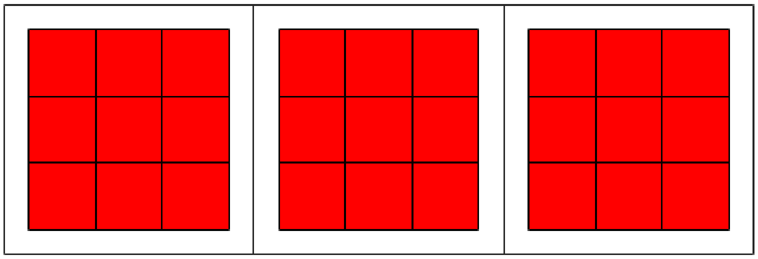
```
C = A .* B;
cells=([{A} {B} {C}])
```

```
cells =
    [3×3 double]    [3×3 double]    [3×3 double]
```

```
cells{1:3}
```

```
ans =

     1     2     0
     2     5    -1
     4    10    -1
ans =

     1     2     4
     2     5    10
     0    -1    -1
ans =

     1     4     0
     4    25   -10
     0   -10     1
```

```
cellplot(cells)
```



Let's use the matrix A to solve the equation, A*x = b. We do this by using the \ (backslash) operator.

```
b=[1;2;3];x = A\b
```

```
x =

     3
    -1
    -1
```

Now we can show that A*x is equal to b.

```
r = A*x - b
```

MATLAB has functions for nearly every type of common matrix calculation.

There are functions to obtain eigenvalues ...

```
[v,lam]=eig(A)
```

```
v =

   -0.2440   -0.9107    0.4472
   -0.3333    0.3333    0.0000
   -0.9107   -0.2440    0.8944
lam =

    3.7321         0         0
```

```
         0    0.2679         0
         0         0    1.0000
```

... as well as the singular values.

But let's stop here now !

```
svd(A)
```

The "poly" function generates a vector containing the coefficients of the characteristic polynomial.

The characteristic polynomial of a matrix A is

$$det(\lambda I - A)$$

```
p = round(poly(A))
```

We can easily find the roots of a polynomial using the `roots` function.

These are actually the eigenvalues of the original matrix.

```
roots(p)
```

Convolution

MATLAB has many applications beyond just matrix computation.

To convolve two vectors ...

```
q = conv(p,p)
```

Symbolic form: multiplication of polynomials

```
syms x
psym=x^3-5*x^2+5*x-1
qsym=psym*psym
expand(qsym)
q
% Agreement.
```

... or convolve again and plot the result.

```
r = conv(p,q)
plot(r);
```

At any time, we can get a listing of the variables we have stored in memory using the `who` or `whos` command.

```
whos
```

You can get the value of a particular variable by typing its name.

```
A
% disp(A)
```

You can have more than one statement on a single line by separating each statement with commas or semicolons.

If you don't assign a variable to store the result of an operation, the result is stored in a temporary variable called `ans`.

```
sqrt(-1)
```

As you can see, MATLAB easily deals with complex numbers in its calculations.

```
displayEndOfDemoMessage(mfilename)
```