**Assignment Statements**

- \( a = 10 \) Stores 10 in \( a \)
- \( a = \text{solve} (f(x) - f(z), z) \) solves the equation for \( z \) and stores it in \( a \)
- \( a = \text{single}(143/91) \) returns a single precision decimal approximation to the fraction.
- \( a = [1:10] \) will create the vector \( \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \)
- \( a = [0:2:10] \) will create the vector \( \{0, 2, 4, 6, 8, 10\} \)
- \( >> a = [1:2; 2:3] \)

\[
\begin{align*}
\text{a} & = 1 & 2 \\
   & 2 & 3
\end{align*}
\]

**Creating Functions**

There are several ways to create functions.

1. **m-file functions** — click on the blank sheet in the upper left hand corner.
   
   ```matlab
   function fname = fx(x1)
   fname = x1^2;
   ``
   
   Calling this function \( a = \text{fx}(4) \) returns a value in \( a \) of 16

2. **Anonymous functions**
   
   This is a new way of creating a function without using a m-file. On the command line type
   
   ```matlab
   function = @(arguments) function
   
   fzR=@(R,r,omega) omega^2/(4*g)*(R^2-2*r^2);
   
   It is called as \( a = \text{fzR}(RR,rr,om) \)
   ```
   
   and returns the value of the function in \( a \)
## Using solve

`solve` can be used to solve equations symbolically or numerically. MATLAB provides faster ways to numerically solve equations. An example of a symbolic solution is shown below. The equation must be cast in the form of \( f(x) = 0 \), but it is not necessary to add the = 0

\[
\frac{x_1}{1 - x_1} = \frac{x_2 - x_1}{1 - x_2} \quad \text{If } x_1 \text{ is known, solve for } x_2
\]

```matlab
>> syms x1 x2
>> solve ('x1 / (1 - x1) - (x2 - x1) / (1 - x2)', x2)
```

\[\text{ans} = 2 \times x_1 - x_1^2\]

or

```matlab
>> syms x
>> solve ('x^n + 2 * x - 4', x)
```

\[\text{ans} = 5^{(1/2)} - 1\]

or

```matlab
>> f1 = @(x) x^2 + 2 * x - 4;
>> solve (f1 (x), x)
```

\[\text{ans} = 5^{(1/2)} - 1\]

\[-1 - 5^{(1/2)}\]
**TI-89 Calculator Solution for the first solve example**

```
F4 1
Define fc (x1, x2, n) = (x2 - x1) / (1 - x2) ^ n
F2 1
solve (fc (0, 0.96, 1) = fc (0.96, x2), x2)
x2 = .9984
```

### Using fzero

**fzero** is a numeric routine that can be used to find the roots of an equation. It is normally fast and efficient. There is also a function called **fsolve** that uses the same syntax that can be used for solving nonlinear equations. **fzero** also assumes that the equation is equal to zero. It uses the function name and an initial guess as arguments.

Solving the polynomial equation from a previous example

```
f1 = @(x) x^2 + 2 * x - 4;
>> fzero (f1, 0.2)
 ans = 1.2361
>> fzero (f1, -2)
 ans = -3.2361
>> xr = [-5 : 5]
xr = -5 - 4 - 3 - 2 - 1 0 1 2 3 4 5
>> f1 = @(x) x .^ 2 + 2 * x - 4; Note the . before the ^ -- this is necessary to insure that the function is evaluated at each value of xr.
>> plot (xr, f1 (xr))
>> grid on
```

This example shows the importance of plotting functions before looking for the roots.
Plotting

\[ \text{plot}(x, y, \ 'o') \]

Will plot the values of \( y \) versus \( x \) using \( o \) for the points.

Two sets of data can be plotted on the same plot using

\[ \text{plot}(x, y, \ 'o', x1, f1(x1)) \]

Will plot \((x,y)\) as points and \((x1,f1(x1))\) as a line.

See the previous example for plotting a function.

Curve Fitting

The function polyfit can be used to fit a set of \((x,y)\) data to a polynomial of degree \( n \). The syntax is

\[ \text{coef} = \text{polyfit}(x, y, n) \]

polyfit returns the coefficients of the polynomial in \( \text{coef} \) (or any name that you want to assign).

To evaluate the polynomial at any value of \( x \) use polyval. The syntax is

\[ \text{ycal} = \text{polyval}(\text{coef}, x) \]

In this case, polyval returns a single value for the function. To generate a table of \((x,\text{ycal})\) replace the single value of \( x \) with a vector containing the values of \( x \) that you want \( \text{ycal} \) values calculated. The following was created using the \( f1 \) function from above and adding some random error.

\[ \text{>> } p = \text{polyfit}(x, \text{ynew}, 2) \]

\[ p = \begin{bmatrix} 0.9180 \\ 2.8839 \\ -2.4174 \end{bmatrix} \]

\[ \text{>> } \text{plot}(x, \text{ynew}, \ 'o', x, \text{polyval}(p, x)) \]
Integration

Matlab uses a version of Maple for some of the symbolic and numerical integration. There is also a polynomial integration function built into Matlab. Going back to the section on curve fitting, if a series of (x,y) data is available it can sometimes be fit to a polynomial using polyfit

\[ \text{pfit} = \text{polyfit}(x,y,3) \]

Once the coefficients (pfit) are available, the integral of the polynomial can be found by using polyint

\[ \text{pint} = \text{polyint}(\text{pfit}) \]

To evaluate the integral between x-values use polyval

\[ \text{area} = \text{polyval}(\text{pint},x_2) - \text{polyval}(\text{pint},x_1) \]

A function can be integrated between limits using quad or quadl (quadl handles singularities better than quad)

\[ \text{ans} = \text{quad}(f(x),x_1,x_2) \]

Differentiation

A polynomial can be differentiated by using polyder

\[ \text{pder} = \text{polyder}(\text{pfit}) \]

To evaluate the derivative at some value of x use polyval

\[ \text{dx} = \text{polyval}(\text{pder},x) \quad \text{where x is a number} \]

The Matlab help files are useful, but you may have to try several search terms to find the information that you need.