

# Defining and using functions

```
>> help sin
```

```
SIN      Sine of argument in radians.
```

```
    SIN(X) is the sine of the elements of X
```

```
>> pi
```

```
ans =
```

```
3.141592653589793
```

```
>> sin(pi/4)
```

```
ans =
```

```
0.707106781186547
```

```
>> sin([pi/2,pi/4])
```

```
ans =
```

```
1.0000000000000000    0.707106781186547
```

```
>> sin([pi/2,pi/4 ; pi/8, pi/16])
```

```
ans =
```

```
1.0000000000000000    0.707106781186547
```

```
0.382683432365090    0.195090322016128
```

```
>> help sum
```

```
SUM Sum of elements.
```

$S = \text{SUM}(X)$  is the sum of the elements of the vector  $X$ . If  $X$  is a matrix,  $S$  is a row vector with the sum over each column. For N-D arrays,  $\text{SUM}(X)$  operates along the first non-singleton dimension.

```
>> sum([1 2 3])
```

```
ans =      6
```

```
>> A=[1 2 3 ; 4 5 6]
```

```
A =
```

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

```
\eehand
```

```
>> sum(A)
```

```
ans =      5      7      9
```

```
>> help sum
...
S = SUM(X,DIM) sums along the dimension DIM.
```

```
>> A
A =
     1     2     3
     4     5     6
```

```
>> sum(A,1)
ans =     5     7     9
```

```
>> sum(A,2)
ans =
     6
    15
```

```
>> sum(A)
ans =
     5     7     9
```

```
>> help eig
```

```
EIG      Eigenvalues and eigenvectors.
```

```
E = EIG(X) is a vector containing the eigenvalues of a square matrix X.
```

```
[V,D] = EIG(X) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that  $X*V = V*D$ .
```

```
A =
```

```
 1      0
 2      0
```

```
>> [V,D]=eig(A)
```

```
V =
```

```
          0      0.447213595499958
1.0000000000000000      0.894427190999916
```

```
D =
```

```
0 0  
0 1
```

```
>> E=eig(A)
```

```
E =
```

```
0  
1
```

## Writing your own functions

- Example 1 `mysum` (from lab manual by Guckenheimer and Ellner)

```
%sums all rows and columns of input matrix
%(of any size, including vector
%or scalar) A
function f=mysum(A);
f=sum(sum(A));
```

```
A =
     1     2
     5     6
```

```
>> mysum(A)
ans =     14
```

- **Example 2** `elementwise_powerfun`

```
%input is a matrix  
%(of any size, including vector or scalar)  
% x and a power p  
%takes each element of x to power p
```

```
function f=elementwise_powerfun(x,p)  
    f=x.^p;
```

```
>> x=[10 2];  
>> elementwise_powerfun(x,2)  
ans =    100     4
```

## Combining multiple functions in one function.m file: subfunction command

```
%input is a matrix
%(of any size, including vector or scalar)
% x and a power p
%takes each element of x to power p
%then sums every element

function f=elementwise_power_then_sum_function(x,p)
    y=x.^p;

    f=mysum(y);
end

%if mysum isn't defined elsewhere, define it right here

function g=mysum(A)
    g=sum(sum(A)) ;
end
```

```
x =  
    10    2  
>> elementwise_power_then_sum_function(x, 2)  
ans =    104
```



## Using a function as an input to another function: function handles

```
% takes a function handle as an input, and plots it
function f=plotter_function(function_to_plot)

xlist=-2:.01:2;
ylist=function_to_plot(xlist) ;

figure
plot(xlist,ylist)
```

The following does not work:

```
>> plotter_function(sin)
??? Error using ==> sin
Not enough input arguments.
```

```
>> plotter_function(sin(x))
??? Undefined function or variable 'x'.
```

## **need ...**

A function handle is a MATLAB value that provides a means of calling a function indirectly. You can pass function handles in calls to other functions (often called function functions).

@functionname returns a handle to the specified MATLAB function

## **Use:**

```
>> plotter_function(@sin)
```

**Extend the bisection code to operate on user-specified functions: `bisection_function.m`**

```
>> plotter_function(@decreasing_function_for_bisection_demo)
```

```
>> bisection_function(@decreasing_function_for_bisection_demo)
```

```
ans = -0.739089965820312
```

## As promised ... an improved version of bisection

- MATLAB built-in function `fzero`

```
>> help fzero
```

```
FZERO Single-variable nonlinear zero finding.
```

```
    X = FZERO(FUN,X0) tries to find a zero of the function FUN near X0.
```

```
>> fzero(@decreasing_function_for_bisection_demo,-1)
```

```
ans = -0.739085133215161
```