Provide feedback about this page

Anonymous Functions

On this page…

Constructing an Anonymous Function

Anonymous functions give you a quick means of creating simple functions without having to create M-files each time. You can construct an anonymous function either at the MATLAB command line or in any M-file function or script.

The syntax for creating an anonymous function from an expression is

```
fhandle = @(arglist) expr
```

Starting from the right of this syntax statement, the term `expr` represents the body of the function: the code that performs the main task your function is to accomplish. This consists of any single, valid MATLAB expression. Next is `arglist`, which is a comma-separated list of input arguments to be passed to the function. These two components are similar to the body and argument list components of any function.

Leading off the entire right side of this statement is an `@` sign. The `@` sign is the MATLAB operator that constructs a function handle. Creating a function handle for an anonymous function gives you a means of invoking the function. It is also useful when you want to pass your anonymous function in a call to some other function. The `@` sign is a required part of an anonymous function definition.

> **Note** Function handles not only provide access to anonymous functions. You can create a function handle to any MATLAB function. The constructor uses a different syntax: `fhandle = @functionname` (e.g., `fhandle = @sin`). To find out more about function handles, see Function Handles.

The syntax statement shown above constructs the anonymous function, returns a handle to this function, and stores the value of the handle in variable `fhandle`. You can use this function handle in the same way as any other MATLAB function handle.

Simple Example

The statement below creates an anonymous function that finds the square of a number. When you call this function, MATLAB assigns the value you pass in to variable `x`, and then uses x in the equation `x.^2`:

```
sqr = @(x) x.^2;
```

The @ operator constructs a function handle for this function, and assigns the handle to the output variable sqr. As with any function handle, you execute the function associated with it by specifying the variable that contains the handle, followed by a comma−separated argument list in parentheses. The syntax is

```
fhandle(arg1, arg2, ..., argN)
```

To execute the sqr function defined above, type

```
a = sqr(5)
a =
   25
```

Because sqr is a function handle, you can pass it in an argument list to other functions. The code shown here passes the sqr anonymous function to the MATLAB quad function to compute its integral from zero to one:

```
quad(sqr, 0, 1)
ans =
   0.3333
```

A Two−Input Example

As another example, you could create the following anonymous function that uses two input arguments, x and y. (The example assumes that variables A and B are already defined):

```
sumAxBy = @(x, y) (A*x + B*y);

whos sumAxBy
Name        Size                   Bytes  Class

sumAxBy     1x1                       16  function_handle
```

To call this function, assigning 5 to x and 7 to y, type

```
sumAxBy(5, 7)
```

Evaluating With No Input Arguments

For anonymous functions that do not take any input arguments, construct the function using empty parentheses for the input argument list:

```
t = @() datestr(now);
```

Also use empty parentheses when invoking the function:

```
t()

ans =
04−Sep−2003 10:17:59
```

You must include the parentheses. If you type the function handle name with no parentheses, MATLAB just identifies the handle; it does not execute the related function:

```
t
```

```
t =
    @() datestr(now)
```

Back to Top

Arrays of Anonymous Functions

To store multiple anonymous functions in an array, use a cell array. The example shown here stores three simple anonymous functions in cell array A:

```
A = {@(x)x.^2, @(y)y+10, @(x,y)x.^2+y+10}
A =
    [@(x)x.^2]     [@(y)y+10]     [@(x,y)x.^2+y+10]
```

Execute the first two functions in the cell array by referring to them with the usual cell array syntax, A{1} and A{2}:

```
A{1}(4) + A{2}(7)
ans =
    33
```

Do the same with the third anonymous function that takes two input arguments:

```
A{3}(4, 7)
ans =
    33
```

Space Characters in Anonymous Function Elements

Note that while using space characters in the definition of any function can make your code easier to read, spaces in the body of an anonymous function that is defined in a cell array can sometimes be ambiguous to MATLAB. To ensure accurate interpretation of anonymous functions in cell arrays, you can do any of the following:

- Remove all spaces from at least the body (not necessarily the argument list) of each anonymous function:

  ```
  A = {@(x)x.^2, @(y)y+10, @(x, y)x.^2+y+10};
  ```

- Enclose in parentheses any anonymous functions that include spaces:

  ```
  A = {(@(x)x .^ 2), (@(y) y +10), (@(x, y) x.^2 + y+10)};
  ```

- Assign each anonymous function to a variable, and use these variable names in creating the cell array:

  ```
  A1 = @(x)x .^ 2;  A2 = @(y) y +10;  A3 = @(x, y)x.^2 + y+10;
  A = {A1, A2, A3};
  ```

Back to Top

Outputs from Anonymous Functions

As with other MATLAB functions, the number of outputs returned by an anonymous function depends mainly on how many variables you specify to the left of the equals (=) sign when you call the function.

For example, consider an anonymous function getPersInfo that returns a person's address, home phone, business phone, and date of birth, in that order. To get someone's address, you can call the function specifying just one output:

```
address = getPersInfo(name);
```

To get more information, specify more outputs:

```
[address, homePhone, busPhone] = getPersInfo(name);
```

Of course, you cannot specify more outputs than the maximum number generated by the function, which is four in this case.

Example

The anonymous getXLSData function shown here calls the MATLAB xlsread function with a preset spreadsheet filename (records.xls) and a variable worksheet name (worksheet):

```
getXLSData = @(worksheet) xlsread('records.xls', worksheet);
```

The records.xls worksheet used in this example contains both numeric and text data. The numeric data is taken from instrument readings, and the text data describes the category that each numeric reading belongs to.

Because the MATLAB xlsread function is defined to return up to three values (numeric, text, and raw data), getXLSData can also return this same number of values, depending on how many output variables you specify to the left of the equals sign in the call. Call getXLSData a first time, specifying only a single (numeric) output, dNum:

```
dNum = getXLSData('Week 12');
```

Display the data that is returned using a for loop. You have to use generic names (v1, v2, v3) for the categories, due to the fact that the text of the real category names was not returned in the call:

```
for k = 1:length(dNum)
   disp(sprintf('%s    v1: %2.2f    v2: %d    v3: %d', ...
      datestr(clock, 'HH:MM'), dNum(k,1), dNum(k,2), ...
      dNum(k,3)));
   end
```

Here is the output from the first call:

```
12:55   v1: 78.42    v2: 32    v3: 37
13:41   v1: 69.73    v2: 27    v3: 30
14:26   v1: 77.65    v2: 17    v3: 16
15:10   v1: 68.19    v2: 22    v3: 35
```

Now try this again, but this time specifying two outputs, numeric (dNum) and text (dTxt):

```
[dNum, dTxt] = getXLSData('Week 12');

for k = 1:length(dNum)
   disp(sprintf('%s    %s: %2.2f    %s: %d    %s: %d', ...
      datestr(clock, 'HH:MM'), dTxt{1}, dNum(k,1), ...
      dTxt{2}, dNum(k,2), dTxt{3}, dNum(k,3)));
   end
```

This time, you can display the category names returned from the spreadsheet:

```
12:55    Temp: 78.42    HeatIndex: 32    WindChill: 37
13:41    Temp: 69.73    HeatIndex: 27    WindChill: 30
14:26    Temp: 77.65    HeatIndex: 17    WindChill: 16
15:10    Temp: 68.19    HeatIndex: 22    WindChill: 35
```

Back to Top

Variables Used in the Expression

Anonymous functions commonly include two types of variables:

- Variables specified in the argument list. These often vary with each function call.
- Variables specified in the body of the expression. MATLAB captures these variables and holds them constant throughout the lifetime of the function handle.

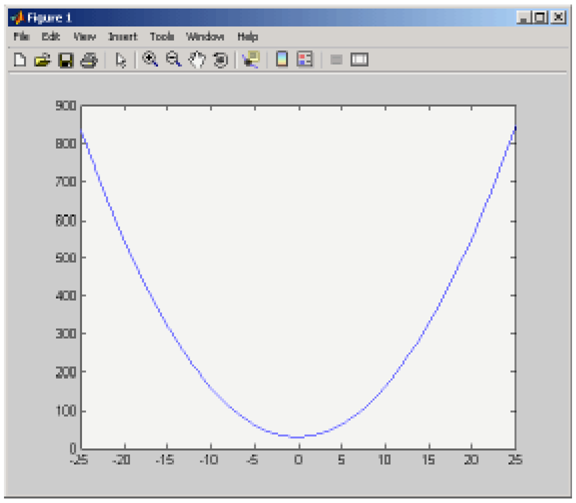
The latter variables must have a value assigned to them at the time you construct an anonymous function that uses them. Upon construction, MATLAB captures the current value for each variable specified in the body of that function. The function will continue to associate this value with the variable even if the value should change in the workspace or gc out of scope.

The fact that MATLAB captures the values of these variables when the handle to the anonymous function is constructed enables you to execute an anonymous function from anywhere in the MATLAB environment, even outside the scope in which its variables were originally defined. But it also means that to supply new values for any variables specified within the expression, you must reconstruct the function handle.

Changing Variables Used in an Anonymous Function

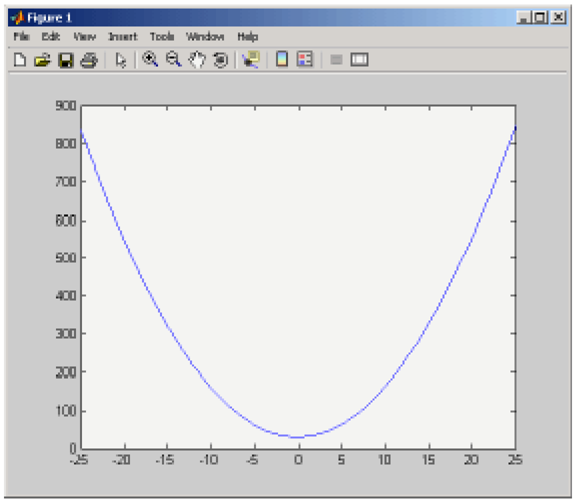
The second statement shown below constructs a function handle for an anonymous functior called `parabola`that uses variables `a`, `b`, and `c` in the expression. Passing the function handle to the MATLAB`fplot` function plots it out using the initial values for these variables:

```
a = 1.3;   b = .2;   c = 30;
parabola = @(x) a*x.^2 + b*x + c;
fplot(parabola, [-25 25])
```



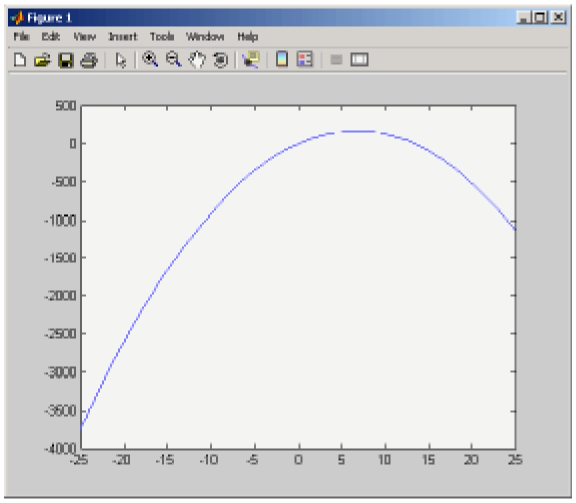
If you change the three variables in the workspace and replot the figure, the parabola remains unchanged because the `parabola` function is still using the initial values of a, b, and c:

```
a = -3.9;   b = 52;   c = 0;
fplot(parabola, [-25 25])
```



To get the function to use the new values, you need to reconstruct the function handle, causing MATLAB to capture the updated variables. Replot using the new construct, and this time the parabola takes on the new values:

```
a = -3.9;   b = 52;   c = 0;
parabola = @(x) a*x.^2 + b*x + c;
fplot(parabola, [-25 25])
```



For the purposes of this example, there is no need to store the handle to the anonymous function in a variable (`parabola`, in this case). You can just construct and pass the handle right within the call to `fplot`. In this way, you update the values of a, b, and c on each call:

```
fplot(@(x) a*x.^2 + b*x + c, [-25 25])
```

Back to Top

Examples of Anonymous Functions

This section shows a few examples of how you can use anonymous functions. These examples are intended to show you how to program with this type of function. For more mathematically oriented examples, see the MATLAB Mathematics documentation.

The examples in this section include

- Example 1 — Passing a Function to quad
- Example 2 — Multiple Anonymous Functions

Example 1 — Passing a Function to quad

The equation shown here has one variable t that can vary each time you call the function, and two additional variables, g and omega. Leaving these two variables flexible allows you to avoid having to hardcode values for them in the function definition:

```
x = g * cos(omega * t)
```

One way to program this equation is to write an M−file function, and then create a function handle for it so that you can pass the function to other functions, such as the MATLAB quad function as shown here. However, this requires creating and maintaining a new M−file for a purpose that is likely to be temporary, using a more complex calling syntax when calling quad, and passing the g and omega parameters on every call. Here is the function M−file:

```
function f = vOut(t, g, omega)
f = g * cos(omega * t);
```

This code has to specify g and omega on each call:

```
g = 2.5;  omega = 10;

quad(@vOut, 0, 7, [], [], g, omega)
ans =
   0.1935

quad(@vOut, -5, 5, [], [], g, omega)
ans =
   -0.1312
```

You can simplify this procedure by setting the values for `g` and `omega` just once at the start, constructing a function handle to an anonymous function that only lasts the duration of you MATLAB session, and using a simpler syntax when calling `quad`:

```
g = 2.5;  omega = 10;

quad(@(t) (g * cos(omega * t)), 0, 7)
ans =
   0.1935

quad(@(t) (g * cos(omega * t)), -5, 5)
ans =
   -0.1312
```

To preserve an anonymous function from one MATLAB session to the next, save the function handle to a MAT-file

```
save anon.mat f
```

and then load it into the MATLAB workspace in a later session:

```
load anon.mat f
```

Example 2 — Multiple Anonymous Functions

This example solves the following equation by combining two anonymous functions:

$$g(c) = \int_0^1 (x^2 + cx + 1)dx$$

The equivalent anonymous function for this expression is

```
g = @(c) (quad(@(x) (x.^2 + c*x + 1), 0, 1));
```

This was derived as follows. Take the parenthesized part of the equation (the integrand) and write it as an anonymous function. You don't need to assign the output to a variable as it will only be passed as input to the quad function:

```
@(x) (x.^2 + c*x + 1)
```

Next, evaluate this function from zero to one by passing the function handle, shown here as the entire anonymous function, to `quad`:

```
quad(@(x) (x.^2 + c*x + 1), 0, 1)
```

Supply the value forc by constructing an anonymous function for the entire equation and you are done:

```
g = @(c) (quad(@(x) (x.^2 + c*x + 1), 0, 1));

g(2)
ans =
   2.3333
```

Back to Top

Provide feedback about this page

Overview of MATLAB Function Types   Primary M−File Functions