Solving Algebraic Equations and Other Symbolic Tools

In this chapter we will begin to look at using MATLAB to solve equations. We start with simple algebraic equations considering solutions for single variables and solving systems of equations. Then we will look at working with transcendental, trig, and hyperbolic functions. Finally we will see how MATLAB handles complex numbers. In the next chapter we will take a look at using MATLAB to solve differential equations.

Solving Basic Algebraic Equations

To solve an algebraic equation in MATLAB we can call upon the *solve* command. At its most basic all we have to do is type in the equation we want to solve enclosed in quotes and hit return. Let's start by looking at a trivial example. Suppose that we wanted to use MATLAB to find the value of x that solves:

$$x + 3 = 0$$

Now the suspense is building—but many clever readers will deduce the answer is $x = -3$ and wonder why we're bothering with this. Well the reason is that it will make seeing how to use MATLAB for symbolic computing a snap. We can find the solution in one step. All we do is create a variable and assign it the value returned by solve in the following way:

```
>> x = solve('x + 3 = 0')

x =

-3
```

Now it isn't necessary to include the right-hand side of the equation. As you can see from the following example, MATLAB *assumes* that when you pass $x + 8$ to solve that you mean $x + 8 = 0$. To verify this, we run this command line:

```
>> x = solve('x+8')

x =

-8
```

So enter the equations whichever way you want. I prefer to be as clear as possible with my intentions, so would rather use $x + 8 = 0$ as the argument.

It is possible to include multiple symbols in the equation you pass to solve. For instance, we might want to have a *constant* included in an equation like this:

$$ax + 5 = 0$$

If we enter the equation in MATLAB, it seems to just assume that we want to solve for x:

```
>> solve('a*x+5')

ans =

-5/a
```

However, there is a second way to call solve. We can tell it *what symbol* we want it to solve for. This is done using the following syntax:

```
solve(equation, variable)
```

Like the equation that you pass to solve, the variable must be enclosed in single quotes. Returning to the equation $ax + 5 = 0$, let's tell MATLAB to find a instead. We do this by typing:

```
>> solve('a*x + 5','a')
```

MATLAB responds with the output:

```
ans =

-5/x
```

Solving Quadratic Equations

The solve command can be used to solve higher order equations, to the delight of algebra students everywhere. For those of us who have moved beyond algebra MATLAB offers us a way to check results when quadratic or cubic equations pop up or to save us from the tedium of solving the equations.

The procedure used is basically the same as we've used so far, we just use a caret character (^) to indicate exponentiation. Let's consider the equation:

$$x^2 - 6x - 12 = 0$$

We could solve it by hand. You can complete the square or just apply the quadratic formula. To solve it using MATLAB, we write:

```
>> s = solve('x^2 -6*x -12 = 0')
```

MATLAB responds with the two roots of the equation:

```
s =

 3+21^(1/2)
 3-21^(1/2)
```

Now, how do you work with the results returned by solve? We can extract them and use them just like any other MATLAB variable. In the case of two roots like this one, the roots are stored as $s(1)$ and $s(2)$. So we can use one of the roots to define a new quantity:

```
>> y = 3 + s(1)

y =

6+21^(1/2)
```

Here is another example where we refer to both roots returned by solve:

```
>> s(1) + s(2)

ans =

6
```

When using solve to return a single variable, the array syntax is not necessary. For example, we use x to store the solution of the following equation:

```
>> x = solve('3*u + 9 = 8')
```

MATLAB dutifully tells us that:

```
x =

-1/3
```

Now we can use the result in another equation:

```
>> z = x + 1

z =

2/3
```

It is possible to assign an equation to a variable and then pass the variable to solve. For instance, let's create an equation (generated at random on the spot) and assign it to a variable with the meaningless name d:

```
>> d = 'x^2 + 9*x -7 = 0';
```

Now we call solve this way:

```
>> solve(d)
```

MATLAB correctly tells us the two roots of the equation:

```
ans =

 -9/2+1/2*109^(1/2)

 -9/2-1/2*109^(1/2)
```

Plotting Symbolic Equations

Let's take a little detour from solving equations, to see how we can plot symbolically entered material. OK while I argued that writing 'x^2 – 6 * x – 12 = 0' was better than 'x^2 – 6*x – 12' it turns out there is a good reason why you might choose the latter method. The reason is that MATLAB allows us to generate plots of symbolic equations we've entered. This can be done using the *ezplot* command. Let's do that for this example.

First let's create the string to represent the equation:

```
>> d = 'x^2 -6*x - 12';
```

Now we call ezplot:

```
>> ezplot(d)
```

MATLAB responds with the graph shown in Figure 5-1.

A couple of things to notice are:

- ezplot has conveniently generated a title for the plot shown at the top, without us having to do any work.
- It has labeled the x axis for us.

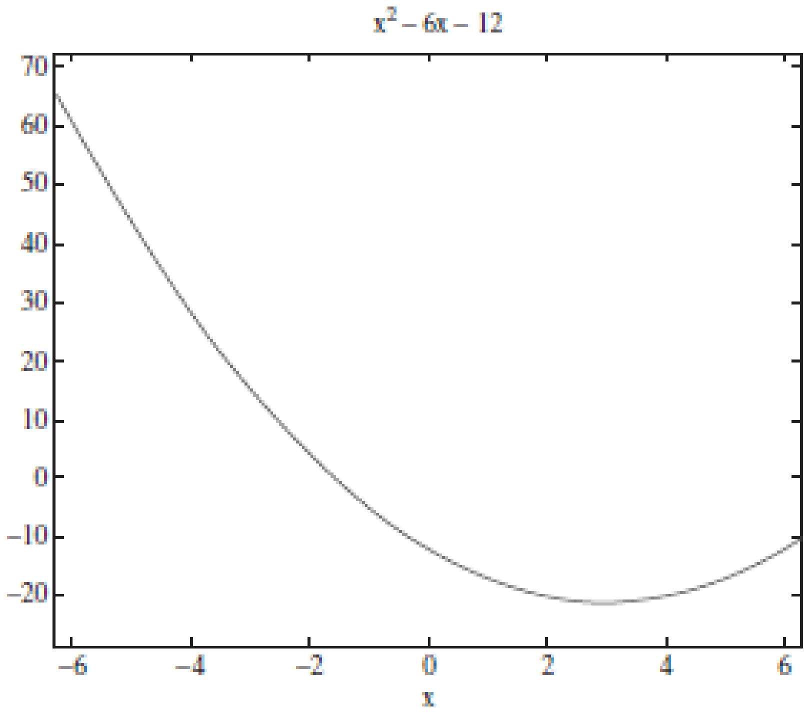


Figure 5-1 A plot of a symbolic function generated using ezplot

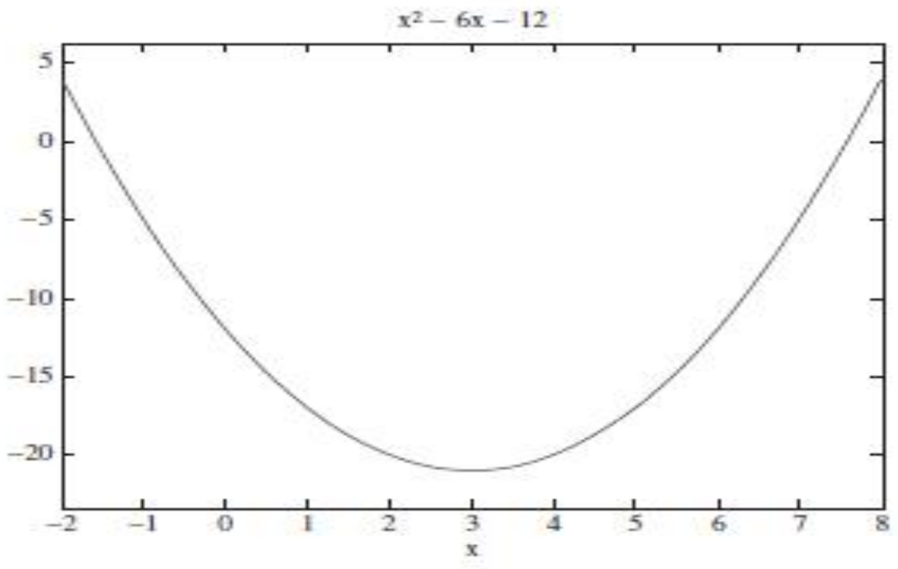


Figure 5-2 Using ezplot while specifying the domain

The function also picked what values to use for the domain and range in the plot. Of course we may not like what it picked. We can specify what we want by specifying the domain with the following syntax:

```
ezplot(f, [x1, x2])
```

This plots f for $x_1 < x < x_2$. Returning to the previous example, let's say we wanted to plot it for $-2 < x < 8$. We can do that using the following command:

```
>> d = 'x^2 -6*x - 12';
>> ezplot(d,[-2,8])
```

The plot generated this time is shown in Figure 5-2.

Before we go any further, let's get back to the " = 0" issue. Suppose we tried to plot:

```
>> ezplot('x+3=0')
```

MATLAB doesn't like this at all. It spits out a series of meaningless error messages:

```
??? Error using ==> inlineeval
Error in inline expression ==> x+3=0
```

```
??? Error: The expression to the left of the equals sign is
not a valid target for an assignment.

Error in ==> inline.feval at 34
        INLINE_OUT_ = inlineeval(INLINE_INPUTS_, INLINE_OBJ_
.inputExpr, INLINE_OBJ_.expr);

Error in ==> specgraph\private\ezplotfeval at 54
    z = feval(f,x(1));

Error in ==> ezplot>ezplot1 at 448
[y,f,loopflag] = ezplotfeval(f,x);

Error in ==> ezplot at 148
    [hp,cax] = ezplot1(cax,f{1},vars,labels,args{:});
```

Now, if we instead type:

```
>> ezplot('x+3')
```

It happily generates the straight line shown in Figure 5-3.

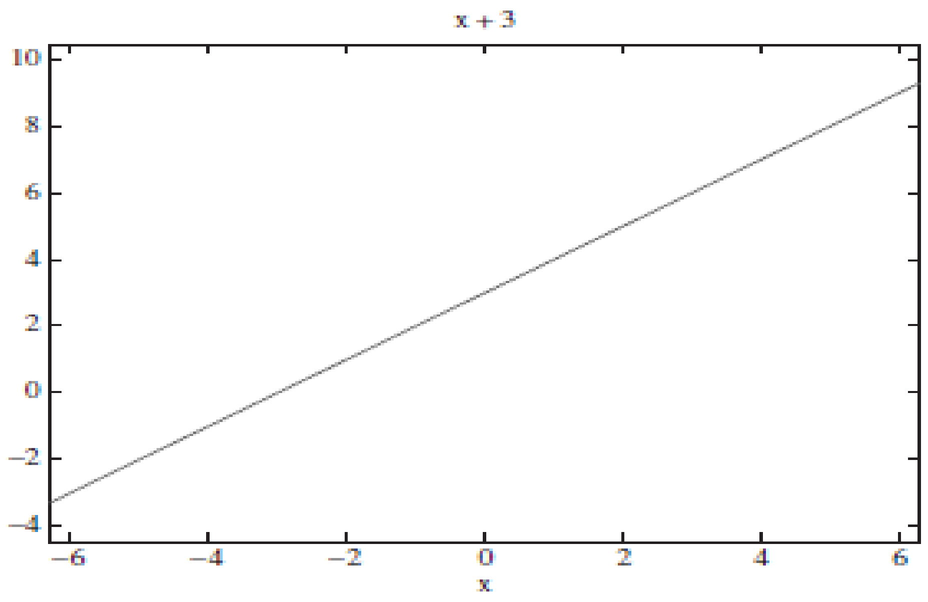


Figure 5-3 Using ezplot('$x + 3$') works, but *ezplot*('$x + 3 = 0$') will generate an error

Now we just mentioned a while ago that we could tell ezplot how to specify the domain to include in the plot. Naturally it also allows us to specify the range. Just for giggles, let's say we wanted to plot:

$$x + 3 = 0$$
$$-4 < x < 4, -2 < y < 2$$

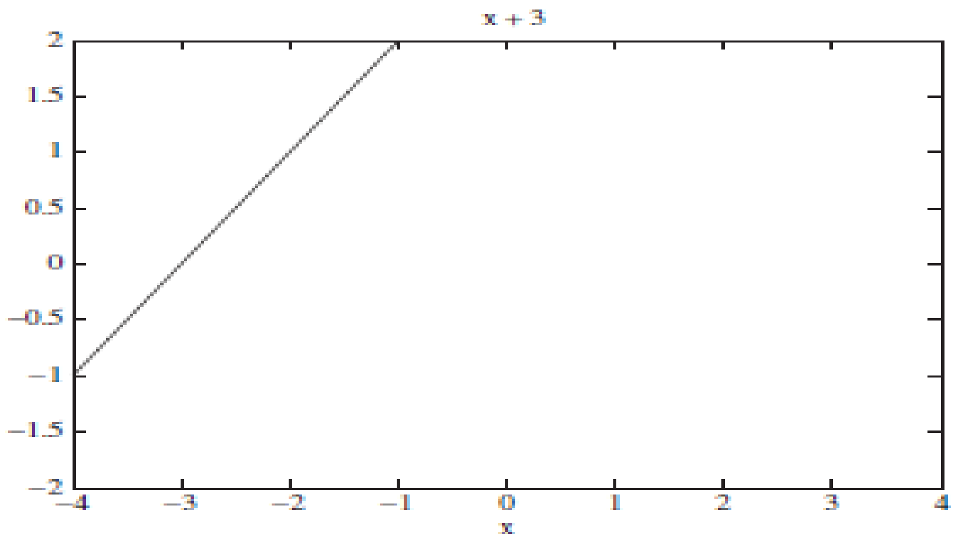
We can do this by typing:

```
>> ezplot('x+3',[-4,4,-2,2])
```

The plot generated is shown in Figure 5-4. So, to specify you want the plot to be over $x_1 < x < x_2$ and $y_1 < y < y_2$ include $[x_1, x_2, y_1, y_1]$ in your call to ezplot.

EXAMPLE 5-1

Find the roots of $x^2 + x - \sqrt{2} = 0$ and plot the function. Determine the numerical value of the roots.



SOLUTION 5-1

First let's create a string to represent the equation. First as an aside, note that you *can* include predefined MATLAB expressions in your equation. So it would be perfectly OK to enter the equation as:

```
>> eq = 'x^2 + x - sqrt(2)';
```

Or if you like, you could write:

```
>> eq = 'x^2 + x - 2^(1/2)';
```

Next we call solve to find the roots:

```
>> s = solve(eq)

s =
 -1/2+1/2*(1+4*2^(1/2))^(1/2)
 -1/2-1/2*(1+4*2^(1/2))^(1/2)
```

To determine the numerical value of the roots, we need to extract them from the array and convert them into type double. This is done by simply passing them to the double(.) command. For example, we get the first root out by writing:

```
>> x = double(s(1))

x =

    0.7900
```

And the second root:

```
>> y = double(s(2))

y =

   -1.7900
```

To plot the function, we use a call to ezplot:

```
>> ezplot(eq)
```

The result is shown in Figure 5-5.

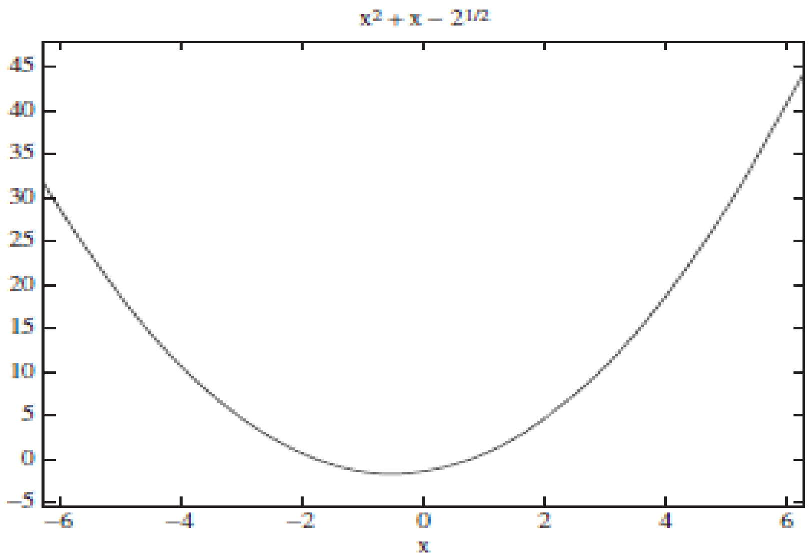


Figure 5-5 A plot of the quadratic equation solved in Example 5-1

Solving Higher Order Equations

Of course we can use MATLAB to solve higher order equations. Let's try a cubic. Suppose we are told that:

$$(x + 1)^2 (x - 2) = 0$$

Solving an equation like this is no different than what we've done so far. We find that the roots are:

```
>> s = solve(eq)

s =

   2
  -1
  -1
```

EXAMPLE 5-2

Find the roots of the fourth order equation

$$x^4 - 5x^3 + 4x^2 - 5x + 6 = 0$$

and plot the function for $-10 < x < 10$.

SOLUTION 5-2

First we define the function by creating a character string to represent it:

```
>> eq1 = 'x^4-5*x^3+4*x^2-5*x+6';
```

Then we call solve to find the roots:

```
>> s = solve(eq1);
```

Now let's define some variables to extract the roots from *s*. If you list them symbolically, you will get a big mess. We show part of the first root here:

```
>> a = s(1)

a =

5/4+1/12*3^(1/2)*((43*(8900+12*549093^(1/2))^(1/3)+2*(8900+
12*549093^(1/2))^(2/3)+104)....
```

Try it and you will see this term goes on a long way. So let's use double to get a numerical result:

```
>> a = double(s(1))

a =

    4.2588
```

Now that is much nicer. We get the rest of the roots. Since it's a fourth order equation, well there are four roots:

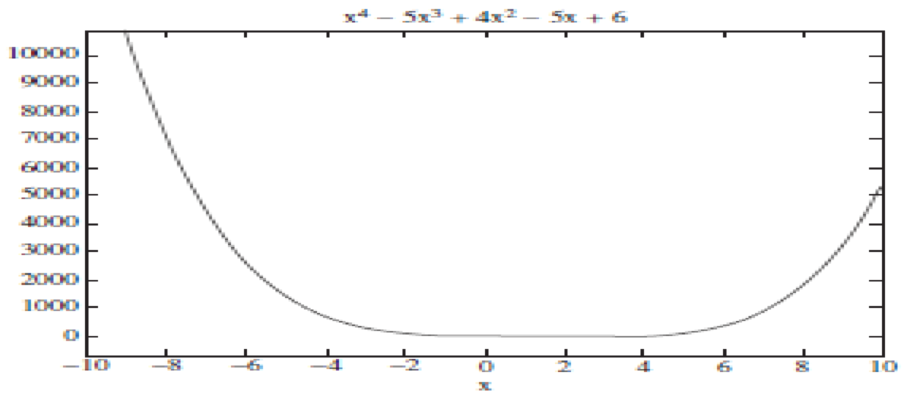
```
>> b = double(s(2))

b =

    1.1164
```

```
>> c = double(s(3))

c =

  -0.1876 + 1.1076i

>> d = double(s(4))

d =

  -0.1876 - 1.1076i
```

Notice two of the roots are complex numbers. Now let's plot the function over the domain indicated:

```
>> ezplot(eq1, [-10 10])
```

The result is shown in Figure 5-6.



Basic Symbolic Calculus and Differential Equations

In this chapter we will learn how to use MATLAB to do symbolic calculus. Specifically, we will start by examining limits and derivatives, and then see how to solve differential equations. We will cover integration in the next chapter.

Calculating Limits

MATLAB can be used to calculate limits by making a call to the *limit* command. The most basic way to use this command is to type in the expression you want to use. MATLAB will then find the limit of the expression as the independent variable goes to zero. For example, if you enter a function $f(x)$, MATLAB will then find $\lim_{x \to 0} f(x)$. Here is an example:

```
>> syms x
>> limit((x^3 + 1)/(x^4 + 2))

ans =

1/2
```

Remember, the limit command falls in the realm of symbolic computing, so be sure to use the *syms* command to tell MATLAB which symbolic variables you are using.

To compute $\lim_{x \to a} f(x)$, the limit command is called using the syntax *limit(f, a)*. For example:

```
>> limit(x + 5,3)

ans =

8
```

EXAMPLE 6-1

Let $f(x) = \frac{2x+1}{x-2}$ and $g(x) = x^2 + 1$. Compute the limit as $x \to 3$ of both functions and verify the basic properties of limits using these two functions and MATLAB.

SOLUTION 6-1

First we tell MATLAB what symbolic variables we will use and define the functions:

```
>> syms x
>> f = (2*x + 1)/(x-2);
>> g = x^2 + 1;
```

Now let's find the limit of each function, and store the result in a variable we can use later:

```
>> F1 = limit(f,3)

F1 =

7


>> F2 = limit(g,3)

F2 =

10
```

The first property of limits we wish to verify is:

$$\lim_{x\to a}(f(x)+g(x)) = \lim_{x\to a} f(x) + \lim_{x\to a} g(x)$$

From our calculations so far we see that:

$$\lim_{x\to 3} f(x) + \lim_{x\to 3} g(x) = 7 + 10 = 17$$

Now let's verify the relation by calculating the left-hand side:

```
>> limit(f+g,3)

ans =

17
```

Next we can verify:

$$\lim_{x\to a} k\ f(x) = k \lim_{x\to a} f(x)$$

for any constant k. Let's let $k = 3$ for which we should find:

$$\lim_{x\to a} k\ f(x) = k \lim_{x\to a} f(x) = (3)(7) = 21:$$

```
>> k=3;
>> limit(k*f,3)

ans =

21
```

Now let's check the fact that the limit of the product of two functions is the product of their limits, that is:

$$\lim_{x\to a} f(x)g(x) = \lim_{x\to a} f(x) \lim_{x\to a} g(x)$$

The product of the limits is:

```
>> F1*F2

ans =

70
```

And we find the limit of the product to be:

```
>> limit(f*g,3)

ans =

70
```

Finally, let's verify that:

$$\lim_{x\to a} f(x)^{g(x)} = \left(\lim_{x\to a} f(x)\right)^{\lim_{x\to a} g(x)}$$

We can create $f(x)^{g(x)}$ in MATLAB:

```
>> h = f^g

h =

((2*x+1)/(x-2))^(x^2+1)
```

Computing the limit:

```
>> limit(h,3)

ans =

282475249
```

Checking the right side of the relation, we find that they are equal:

```
>> A = F1^F2

A =

282475249
```

As an aside, we can check if two quantities in MATLAB are equal by calling the *isequal* command. If two quantities are not equal, isequal returns 0. Recall that earlier we defined a constant $k = 3$. Here is what MATLAB returns if we compare it to A = F1^F2:

```
>> isequal(A,k)

ans =

      0
```

On the other hand:

```
>> isequal(A,limit(h,3))

ans =

      1
```

Computing $\lim_{x\to\pm\infty} f(x)$.

We can calculate limits of the form $\lim_{x\to\infty} f(x)$ by using the syntax:

```
limit(f,inf)
```

Let's use MATLAB to show that $\lim_{x\to\infty}\left(\sqrt{x^2+x}-x\right)=\frac{1}{2}$:

```
>> limit(sqrt(x^2+x)-x,inf)

ans =

1/2
```

We can also calculate $\lim_{x\to-\infty} f(x)$. For example:

```
>> limit((5*x^3 + 2*x)/(x^10 + x + 7),-inf)

ans =

0
```

MATLAB will also tell us if the result of a limit is ∞. For example, we verify that $\lim_{x\to 0}\frac{1}{|x|}=\infty$:

```
>> limit(1/abs(x))

ans =

Inf
```

LEFT- AND RIGHT-SIDED LIMITS

When a function has a discontinuity, the limit does not exist at that point. To handle limits in the case of a discontinuity at $x = a$, we define the notion of *left-handed* and *right-handed* limits. A left-handed limit is defined as the limit as $x \to a$ from the left, that is x approaches a for values of $x < a$. In calculus we write:

$$\lim_{x \to a^-} f(x)$$

For a right-handed limit, where $x \to a$ from the right, we consider the case when x approaches a for values of $x > a$. The notation used for right-handed limits is:

$$\lim_{x \to a^+} f(x)$$

If these limits are equal, then $\lim_{x \to a} f(x)$ exists. In MATLAB, we can compute left- and right-handed limits by passing the character strings 'left' and 'right' to the limit command as the last argument. We must also tell MATLAB the variable we are using to compute the limit in this case. Let's illustrate with an example.

EXAMPLE 6-2

Show that $\lim_{x \to 3} \frac{x-3}{|x-3|}$ does not exist.

SOLUTION 6-2

First let's define the function in MATLAB:

```
>> f = (x - 3)/abs(x-3);
```

If we plot the function, the discontinuity at $x = 3$ is apparent, as shown in Figure 6-1. Note that we have to give MATLAB the domain over which we want to plot in order to get it to show us the discontinuity:

```
>> ezplot(f,[-1,5])
```

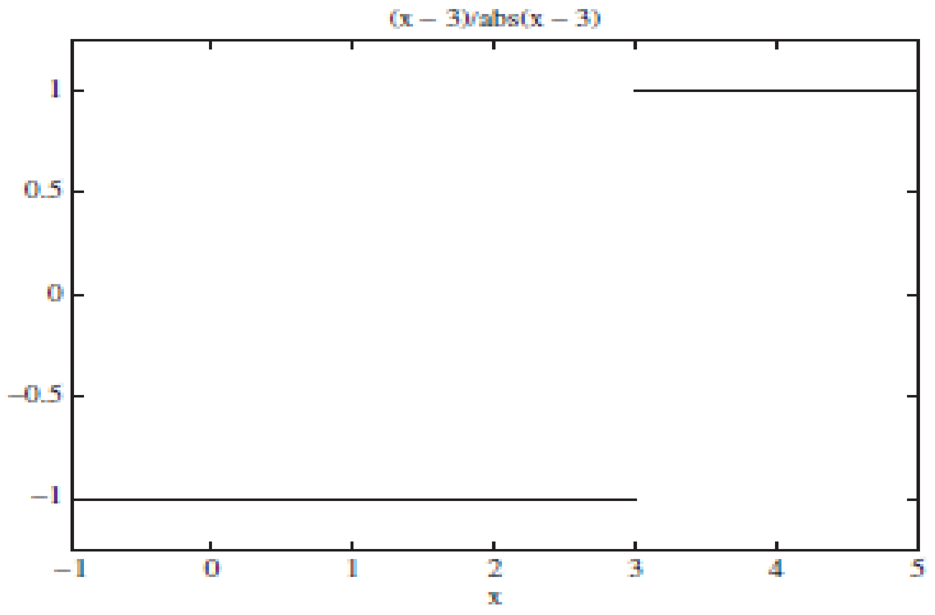


Figure 6-1 A plot showing the discontinuity in $(x - 3)/|x - 3|$

Now let's compute the left-handed limit. To do this, we must pass the function, the variable we are using to take the limit, and the string 'left' as a comma-delimited list:

```
>> a = limit(f,x,3,'left')

a =

-1
```

Now let's take the right-handed limit:

```
>> b = limit(f,x,3,'right')

b =

1
```

Since these two terms are not equal, we have shown that $\lim_{x \to 3} \frac{x-3}{|x-3|}$ does not exist.