

## 5.1 Finding Zeros of Functions

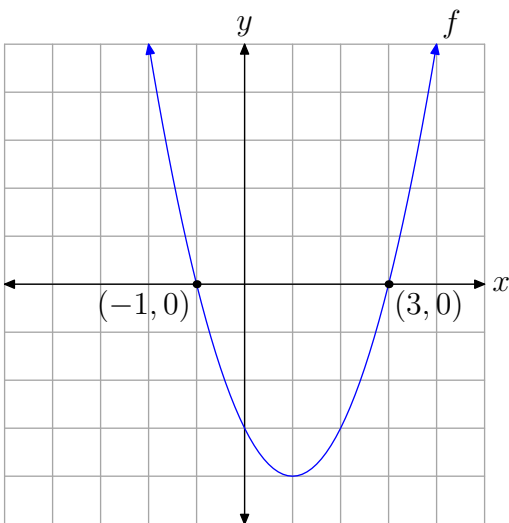
We start with the definition of a zero of a function.

**Definition 1.** A number  $a$  is called a **zero** of a function  $f$  if and only if  $f(a) = 0$ .

As an example, 3 is a zero of the function  $f(x) = x^2 - 2x - 3$  because

$$f(3) = 3^2 - 2(3) - 3 = 0.$$

Similarly, readers may check that  $f(-1) = 0$ , thus making  $-1$  a second zero of the function  $f$ . In **Figure 5.1**, note that the zeros of the function ( $-1$  and  $3$ ) are the  $x$ -values of the points where the graph of  $f$  crosses the  $x$ -axis (the  $x$ -intercepts).



**Figure 5.1.** The zeros of a function are found by noting where the graph of the function crosses the  $x$ -axis (the  $x$ -intercepts).

This relationship between the zeros of the function and the  $x$ -intercepts of its graph suggest that we can find approximations of the zeros of a function by interpreting its graph.

► **Example 2.** Sketch the graph of  $f(x) = x^3 - 4x^2 - 11x + 30$  and use the graph to estimate the zeros of the function. Check your results with Matlab.

<sup>1</sup> Copyrighted material. See: <http://msenux.redwoods.edu/Math4Textbook/>

We begin by setting up an “array smart” anonymous function for  $f(x) = x^3 - 4x^2 - 11x + 30$ .

```
>> f=@(x) x.^3-4*x.^2-11*x+30;
```

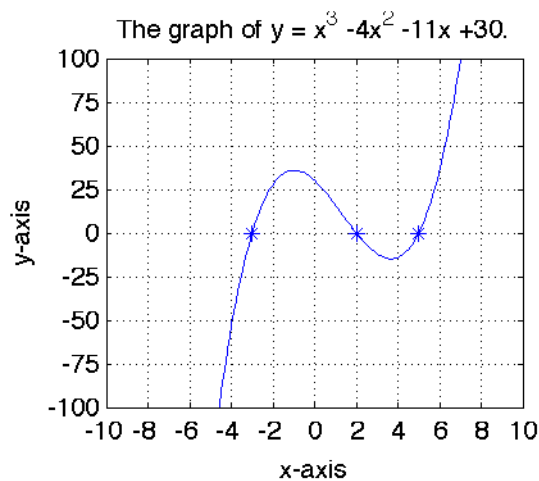
Plot the graph of  $f$ .

```
>> x=linspace(-10,10,200);
>> y=f(x);
>> plot(x,y)
```

Adjust the viewing window and turn on a grid to help with the approximations of the zeros.

```
>> axis([-10,10,-100,100])
>> grid on
```

In **Figure 5.2**, it appears that the graph of  $f$  crosses the  $x$ -axis at  $(-3, 0)$ ,  $(2, 0)$ , and  $(5, 0)$ . Hence, estimates of the zeros are  $-3$ ,  $2$ , and  $5$ .



**Figure 5.2.** Estimating the zeros from the graph of  $f$ .

We can check these results by evaluating the function  $f$  at each approximation of a zero.

```
>> x=[-3,2,5];
>> f(x)
ans =
     0     0     0
```

Hence, each of the values  $-3$ ,  $2$ , and  $5$  are zeros of the function  $f$ .



It is unusual to make approximations that turn out to be the exact zeros of a function. As readers might intuit, this example is a bit of a “setup,” designed to help introduce the concepts. Let’s look at another example where the exact zeros are not so easily found.

► **Example 3.** Use the graph of  $f(x) = x^4 - 29x^2 - 132$  to approximate the zeros of  $f$ . Then use Matlab’s **fzero** function to determine more accurate approximations of the zeros.

Note that

$$f(-x) = (-x)^4 - 29 * (-x)^2 - 132 = x^4 - 29x^2 - 132 = f(x),$$

so the function is *even* and the graph is *symmetric* with respect to the  $y$ -axis, as seen in **Figure 5.3**.

First, define an anonymous function to emulate  $f(x) = x^4 - 29x^2 - 132$ .

```
>> f=@(x) x.^4-29*x.^2-132;
```

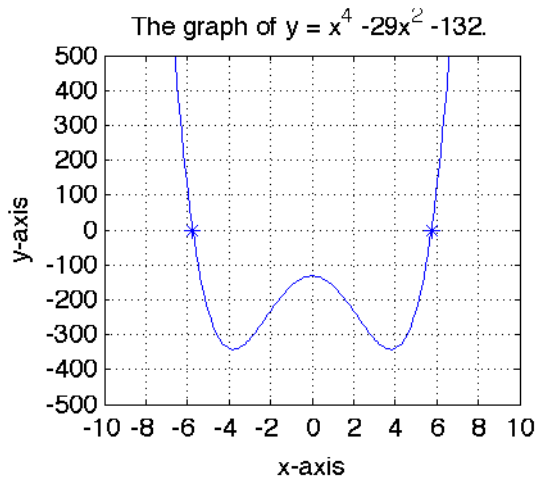
Plot the graph.

```
>> x=linspace(-10,10,200);
>> y=f(x);
>> plot(x,y)
```

Adjust the viewing window and turn on the grid to help with approximation of the zeros.

```
>> axis([-10,10,-500,500])
>> grid on
```

In **Figure 5.3**, note that it appears that we have a zero crossing ( $x$ -intercept) near  $x \approx -5.5$ . Because the graph is symmetric with respect to the  $x$ -axis, the second zero crossing is near  $x \approx 5.5$ .



**Figure 5.3.** Symmetry helps.

Let's use Matlab's **fzero** function to improve on these approximations. One syntax of use follows.

```
X = FZERO(FUN,X0)
```

Here, **FUN** is a function handle and **X0** is an initial guess of the zero. **FUN** must accept a real scalar input and return a real scalar function value. Readers will recall that we have already made the following anonymous function definition.

```
>> f=@(x) x.^4-29*x.^2-132;
```

Thus, the variable **f** contains a function handle that refers to this function. As we believe we have a zero near  $x \approx -5.5$ , we will use this as our initial guess to **fzero**.

```
>> x=fzero(f,-5.5)
x =
   -5.7446
```

We can obtain more digits with **format long**.

```
>> format long
>> x=fzero(f,-5.5)
x =
-5.74456264653803
```

We can check this result manually.

```
>> f(x)
ans =
0
```

It is doubtful that this zero is exact (remember roundoff error), but it is far more accurate than our initial estimate  $-5.5$ .

We can find the second zero (i.e.,  $x \approx 5.74456264653803$ ) by appealing to symmetry, or we can issue a second call to **fzero**, feeding it a second estimate at  $5.5$ .

```
>> x=fzero(f,5.5)
x =
5.74456264653803
```



## Solving Equations with FZERO

Matlab's **fzero** command can be used to solve equations.

► **Example 4.** Solve the equation  $5 - 2x = e^{-0.25x}$  for  $x$ .

The approach is simple. First, make one side of the equation equal to zero.

$$5 - 2x - e^{-0.25x} = 0$$

We could set up an anonymous function to emulate the function on the left-hand side of this equation; i.e., **f=@(x) 5-2\*x-exp(-0.25\*x)**. However, let's write a function m-file instead. Open the editor and enter the following lines.

```
function y = f(x)
y=5-2*x-exp(-0.25*x);
```

Save the function m-file as **f.m** in the current working directory (or save in a directory of choice, then switch the current directory to point at the directory where you've saved **f.m**).

Test to see which **f** will be executed at the Matlab prompt. Here is the response we received.

```
>> which f
f is a variable.
```

We received this response because the anonymous function we created earlier still existed in the command window workspace. Clearing **f** and trying **which** again produced the following result on our system.

```
>> clear f
>> which f
/Users/darnold/Documents/Math4Textbook/trunk/Programming/f.m
```

This path response is where we saved the function m-file on our system. It is also the current directory. However, here is one more piece of evidence that we're good to go.

```
>> type f

function y=f(x)
y=5-2*x-exp(-0.25*x);
```

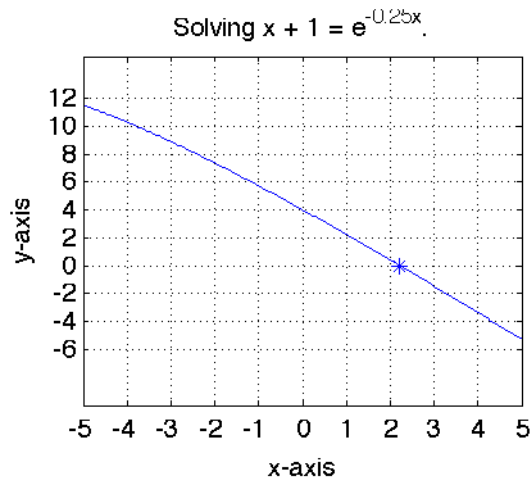
This is exactly what we entered in the function m-file, so we are sure that Matlab is calling the correct function. Now, plot the function (the result is shown in **Figure 5.4**). The commands that follow can be executed at the Matlab prompt (provided the current directory matches where you stored **f.m**), or you can save the commands in a script file (again, in the same directory where you stored **f.m**). You can then execute the script by typing its name at the command line.

```
x=linspace(-5,5,200);
y=f(x);
plot(x,y)
```

Turn on the grid to help estimate the zero of  $f$  from the graph of  $f$ .

```
grid on
```

In **Figure 5.4**, note that the graph of  $f$  crosses the  $x$ -axis between 2 and 3.



**Figure 5.4.** Solving equations with **fzero**.

A second syntax of use for **fzero** follows.

```
X = FZERO(FUN,X0)
```

At first glance, this looks identical to the first syntax offered, but in this case **X0** is a vector of length 2, which represents an *interval* containing a zero of **FUN**. It is required that the function values have *opposite sign* at the endpoints of this interval. In **Figure 5.4**, we note that we have a zero crossing in the interval  $[2, 3]$ . Additionally, the function is *positive* at one endpoint of  $[2, 3]$  and *negative* at the other endpoint. Finally, remember that we must pass a *function handle* to **fzero**, which we can create “on the fly” with the syntax **@functionname**.

```
>> x=fzero(@f,[2,3])
x =
    2.21241937532019
```

We can check the solution by evaluating our function at this zero.

```
>> f(x)
ans =
    2.220446049250313e-16
```

Note that the function value is approximately  $2.22 \times 10^{-16}$ , which is essentially zero. Therefore,  $x \approx 2.21241937532019$  is a solution of  $5 - 2x - e^{-0.25x} = 0$ , and is also a solution of the equivalent equation  $5 - 2x = e^{-0.25x}$ .



## Matlab's Definition of a Zero

Matlab's definition of a zero differs from the mathematical definition of a zero, which is defined as a number  $a$  such that  $f(a) = 0$ . Because Matlab uses floating point arithmetic, it might not be possible to find an exact zero. Consider for instance, the function  $f(x) = x^3 - 2$ , which has a single real zero (and two complex zeros, but that's another story), namely  $x = \sqrt[3]{2}$ . The cube root of 2 has no exact floating point representation, so we will have to be content with finding an approximation of this zero of  $f$ .

Instead of searching for a point where the function actually equals zero, Matlab searches for a point where the function changes sign (from positive to negative or from negative to positive). For a continuous function, this will be near a point where the function equals zero, but strange things can happen if the function is discontinuous.

```
>> fzero(@tan,2)
ans =
    1.5708
```

Readers should note that this is an approximation of  $\pi/2$ .

```
>> pi/2
ans =
    1.5708
```

The graph of the tangent has a vertical asymptote at  $x = \pi/2$  and is positive on the left-hand side of this vertical asymptote, negative on the right. Although  $\pi/2$  is not a zero of  $f(x) = \tan x$ , the function does change sign near this point and **fzero**, in this case, converges to a floating point number near  $\pi/2$ .

However, if a continuous function changes sign, then it must cross the  $x$ -axis and have a real zero. In this case, **fzero** will converge to a floating point number near the actual zero.

Again, Matlab defines a zero as a point where the function crosses the  $x$ -axis (i.e., where it changes sign). Therefore, in the case of  $f(x) = x^2$ , which has a zero  $x = 0$ , **fzero** will fail to find this zero because the function does not change sign near this number. We can see this by first defining an anonymous function.

```
>> f=@(x) x^2;
```

Regardless of what initial guess you feed **fzero**, you will get an error message similar to the following.

```
>> fzero(f,-1)
Exiting fzero: aborting search for an interval containing a sign
change
    because NaN or Inf function value encountered during search.
(Function value at 1.7162e+154 is Inf.)
Check function or try again with a different starting value.
ans =
    NaN
```

In attempting to find an interval where the function changes sign, **fzero** manages to leak out to infinity, and as a result, crashes and fails to find a zero of the function.

## Options

We can use Matlab's **optimset** command to set options for use with **fzero**. You can build a *default* options structure for **fzero** with the following command.

```
>> options=optimset(@fzero)
options =
    Display: 'notify'
    MaxFunEvals: []
    MaxIter: []
    TolFun: []
    TolX: 2.2204e-16
    FunValCheck: 'off'
    OutputFcn: []
```

The **optimset** command returns a Matlab *structure* with several fields. The name on the left-hand side of this command is optional, but we typically use the name **options**. You can access each field of the structure by using the variable name, followed by a dot, followed by the name of the field you wish to access.

```
>> options.Display
ans =
    notify
```

The structure built by the **optimset** command contains fields that are used by a number of functions, such as **fminbnd**, **fminsearch**, **fzero**, and **lsqnonneg**. The structure contains 6 fields, only four of which the **fzero** command will recognize: Display, FunValCheck, OutputFcn, and TolX.<sup>2</sup> Note that TolX is set to 2.2204e-16, which equals **eps**, the distance between the number 1 and the next largest possible floating point number that follows the number 1.

```
>> eps
ans =
    2.2204e-16
```

Thus, this default tolerance provides very accurate approximations of zeros. You can lessen this tolerance should you want to sacrifice precision for an improvement in speed.

It is instructive to set the Display field to 'iter'. This can be done using the dot notation described above.

<sup>2</sup> For a complete description of the options available to **fzero**, type **doc fzero** at the command prompt.

```
>> options.Display='iter'
options =
    Display: 'iter'
  MaxFunEvals: []
    MaxIter: []
    TolFun: []
         TolX: 2.2204e-16
  FunValCheck: 'off'
    OutputFcn: []
```

However, you can use **optimset** to achieve the same result.<sup>3</sup>

```
>> options=optimset(options,'Display','iter')
options =
    Display: 'iter'
  MaxFunEvals: []
    MaxIter: []
    TolFun: []
         TolX: 2.2204e-16
  FunValCheck: 'off'
    OutputFcn: []
```

The function  $f(x) = x^2 - 2x - 2$  has two real zeros,  $x = 1 \pm \sqrt{3}$ . We will use **fzero** to find one of them. First, define an anonymous function.

```
>> f=@(x) x^2-2*x-2;
```

Next, we call **fzero**, passing it an initial guess and the options structure built above.

```
>> fzero(f,-1,options)
```

Because **options.Display** equals 'iter', **fzero** displays output describing its progress as it searches for a floating point approximation of the zero. The first part of the

<sup>3</sup> For a complete description of the use of **optimset**, type **doc optimset** at the Matlab prompt.

output shows **fzero** searching for an interval around the initial guess where the function changes sign. We've left off the last column of the output to save space.

| Search for an interval around -1 containing a sign change: |           |          |          |         |  |
|--|-----------|----------|----------|---------|--|
| Func-count   | a         | f(a)     | b        | f(b)    |  |
| 1  | -1        | 1        | -1       | 1       |  |
| 3  | -0.971716 | 0.887663 | -1.02828 | 1.11394 |  |
| 5  | -0.96     | 0.8416   | -1.04    | 1.1616  |  |
| 7  | -0.943431 | 0.776926 | -1.05657 | 1.22947 |  |
| 9  | -0.92     | 0.6864   | -1.08    | 1.3264  |  |
| 11   | -0.886863 | 0.560252 | -1.11314 | 1.46535 |  |
| 13   | -0.84     | 0.3856   | -1.16    | 1.6656  |  |
| 15   | -0.773726 | 0.146103 | -1.22627 | 1.9563  |  |
| 16   | -0.68     | -0.1776  | -1.22627 | 1.9563  |  |

Note how **fzero** expands the interval  $[a, b]$  around the initial guess as it searches for a change in sign. In step 16, note that the interval has expanded to  $[a, b] = [-0.68, -1.22627]$ . Most importantly, note that  $f(a) = -0.1776$  and  $f(b) = -1.22627$ , so the function  $f(x) = x^2 - 2x - 2$  changes sign on this interval. Hence, this interval must contain a zero of the function  $f$ .

The second half of the output details how **fzero** searches the interval  $[a, b] = [-0.68, -1.2263]$  for a zero. Note how the values of  $x$  converge to the zero while the values of  $f(x)$  converge toward zero. The procedure **fzero** will stop if  $f(x) = 0$  or when two consecutive iterations of  $x$  differ by an amount set by a rule that involves TolX, in the case 2.2204e-16.

| Search for a zero in the interval [-0.68, -1.2263]: |           |              |               |
|---|-----------|--------------|---------------|
| Func-count  | x         | f(x)         | Procedure     |
| 16  | -0.68     | -0.1776      | initial       |
| 17  | -0.725465 | -0.0227694   | interpolation |
| 18  | -0.732075 | 8.22065e-05  | interpolation |
| 19  | -0.732051 | -1.56576e-07 | interpolation |
| 20  | -0.732051 | -1.07248e-12 | interpolation |
| 21  | -0.732051 | 0            | interpolation |

Note that  $x$  converges to a floating point approximation of  $1 - \sqrt{3}$ .

```
>> 1-sqrt(3)
ans =
    -0.7321
```

In the next section, we will spend some time writing our own zero finding functions. They will incorporate what we see above, looking for a change in sign of the function to identify an interval containing a zero of the function.

## Epilogue

Earlier we saw that although  $x = 0$  is a zero of the function  $f(x) = x^2$ , Matlab's **fzero** command failed to find this zero because the function does not change sign in any interval containing this zero. It is instructive to send our options structure to **fzero** in this case to watch what happens.

```
>> fzero(f,-1,options)
```

Because the output is humungus, we list only a few lines.

```
Search for an interval around -1 containing a sign change:
Func-count   a           f(a)           b           f(b)
    1         -1           1             -1           1
    ...         ...           ...           ...           ...

2069      8.581e+153  7.36335e+307  -8.581e+153  7.36335e+307
2071     1.21354e+154  1.47267e+308 -1.21354e+154  1.47267e+308
```

Note that as **fzero** searches for an interval surrounding the initial guess, it does not encounter a change in the sign of the function. Hence, the endpoints of the search interval reach outwards, approaching infinity. The previous error message we received should now make sense.

```
Exiting fzero: aborting search for an interval containing a
sign change because NaN or Inf function value encountered
during search. (Function value at 1.7162e+154 is Inf.)
Check function or try again with a different starting value.
ans =
    NaN
```

## 5.1 Exercises

---

In **Exercises 1-6**, perform each of the following tasks for the given function.

- i. Write an anonymous function and use the function to draw the graph of the given function over the given domain. Turn on the grid.
- ii. Use **fzero** to find the zeros of the function in the given domain. Use the **line** command to mark each zero with a marker or choice and use the **text** command to label each with its coordinates on the plot.

1.  $f(x) = 9 - 4x - x^2$  on  $[-7, 3]$ .

2.  $f(x) = 2x^2 - x - 8$  on  $[-3, 4]$ .

3.  $f(x) = x^3 - 9x^2 + 2x + 8$  on  $[-2, 9]$ .

4.  $f(x) = 4 - 13x^2 + 2x^4$  on  $[-3, 3]$ .

5.  $f(x) = e^{-0.25x} \sin(2x)$  on  $[0, 2\pi]$ .

6.  $f(x) = e^{0.10x} \sin(x/2)$  on  $[0, 8\pi]$ .

---

In **Exercises 7-12**, perform each of the following tasks for the given equation.

- i. Make one side of the equation zero, then write an anonymous function to draw the graph of the non-zero side of the equation on the given domain. Turn on the grid.
- ii. Use **fzero** to find the solutions on the given domain, then mark them on your graph with the **line** command and annotate them with the **text** command.

7.  $e^{-0.25x} = x - 4$  on  $[0, 8]$ .

8.  $e^{0.10x} = 5 - x$  on  $[0, 7]$ .

9.  $\sin(2x) = x/2$  on  $[0, 2\pi]$ .

10.  $2 \cos(x/2) = -x/5$  on  $[0, 4\pi]$ .

11.  $\frac{x}{1+x^2} = x^2$  on  $[-1, 1]$ .

12.  $e^{-x^2} = 4x^2$  on  $[-1, 1]$ .

In **Exercises 13-18**, perform each of the following tasks for the given equation.

- i. Create two anonymous functions  $f$  and  $g$ , one for the left-hand side of the equation, one for the right-hand side of the equation. Use these to draw the graphs of  $f$  and  $g$  on the same coordinate system for the given domain. Use different colors and/or linestyles for each graph. Turn on the grid.
- ii. Make one side of the equation zero, then create a third anonymous function  $h$  to evaluate the nonzero side of the resulting equation. Use  $h$  and **fzero** to find the zeros of the function  $h$ .
- iii. Use the line command and the information provided by the zeros from the previous item to mark the points of intersection of the graphs of  $f$  and  $g$ , then use the text command to mark the points of intersection with their coordinates.

**13.**  $9 - x^2 = 2x + 1$  on  $[-5, 5]$ .

**14.**  $x^2 - 3 = 5 - x$  on  $[-5, 5]$ .

**15.**  $3e^{-0.1x} - 2 = \sin(x)$  on  $[0, 2\pi]$ .

**16.**  $6 - 5e^{0.1x} = 3 \cos(x)$  on  $[0, 2\pi]$ .

**17.**  $2x^2/(1 + 0.5x^2) = 5 - 3x^2$  on  $[-2, 2]$ .

**18.**  $10xe^{-x^2/2} = x$  on  $[-4, 4]$ .

Matlab's **fzero** routine demands that the function you pass as an argument must have exactly one input, a vector or scalar, and only one output, which must be a scalar. Any deviation from this dictum and **fzero** will fail. So, how can we find zeros of functions that have one or more parameters? One possible method is to first define the function's parameters in the workspace, then create an anonymous function of one variable that includes the use of the parameters. For example, suppose that we wish to find zeros of the function  $f(x) = x^3 - x + c$  for various values of the parameter  $c$ . We would first assign a value to the parameter  $c$  in the workspace, then create an anonymous function of a single variable.

```
c=-3;
f=@(x) x.^3-x+c;
```

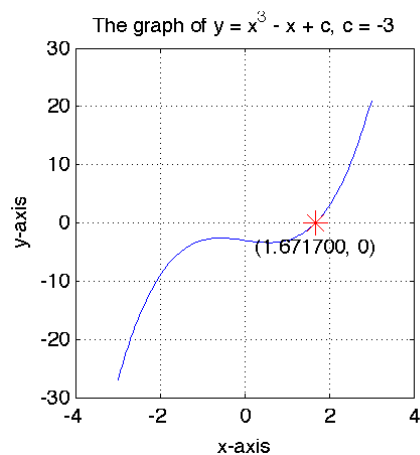
We can now evaluate the anonymous function over a domain, plot and annotate the plot as follows.

```
x=linspace(-3,3);
plot(x,f(x))
grid on
xlabel('x-axis')
ylabel('y-axis')
titleStr=sprintf('The graph of y = x^3 - x + c, c = %d',c);
title(titleStr)
```

Once the graph is drawn, we note the existence of a zero in the interval  $[1, 2]$ , call **fzero**, then annotate the graph with the result.

```
xz=fzero(f,[1,2]);
hLine=line(xz,0);
set(hLine,...
    'LineStyle','None',...
    'Marker','*',...
    'MarkerSize',12,...
    'Color','r')
textStr=sprintf('%f, 0',xz);
hText=text(xz,0-2,textStr);
set(hText,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')
```

The resulting plot follows.



The reason that this works is the fact that the anonymous function is a function of a single variable  $x$ , as determined in its argument list surrounded by parentheses.

In **Exercises 19-22**, perform each of the following tasks for the given function.

- i. Assign the given values to the parameters in the calling workspace.
- ii. Create an anonymous function for the given function, then use the anonymous function to draw the graph of the function.
- iii. Use the graph to find an intervals bracketing the zeros of the function and pass these intervals, along with the handle to the anonymous function, to Matlab's **fzero** command to find accurate approximations of the function's zeros on the given domain. Use the **line** command to mark the zeros and the **text** command to annotate the zeros with their coordinates.

**19.**  $f(x) = x^3 + cx + 1$ ,  $c = -5$ , on  $[-3, 3]$ .

**20.**  $f(x) = cx^3 + 5x + 1$ ,  $c = -2$ , on  $[-2, 2]$ .

**21.**  $f(x) = x^3 + cx + d$ ,  $c = -4$ ,  $d = -1$ , on  $[-3, 3]$ .

**22.**  $f(x) = cx^3 + dx + 2$ ,  $c = -1$ ,  $d = 4$ , on  $[-3, 3]$ .

## 5.1 Answers

---

1. The following code was used to generate the figure that follows.

```
%% Exercise #1
close all
clear all
clc

f=@(x) 9-4*x-x.^2;
x=linspace(-7,3);
y=f(x);
plot(x,y)
grid on

xlabel('x-axis')
ylabel('y-axis')
title('Zeros of f(x) = 9 - 4x - x^2.')
```

```
z1=fzero(f,[-6,-5]);
z2=fzero(f,[1,2]);

line(z1,f(z1),...
     'LineStyle','none',...
     'Marker','*',...
     'MarkerSize', 12,...
     'Color','r')

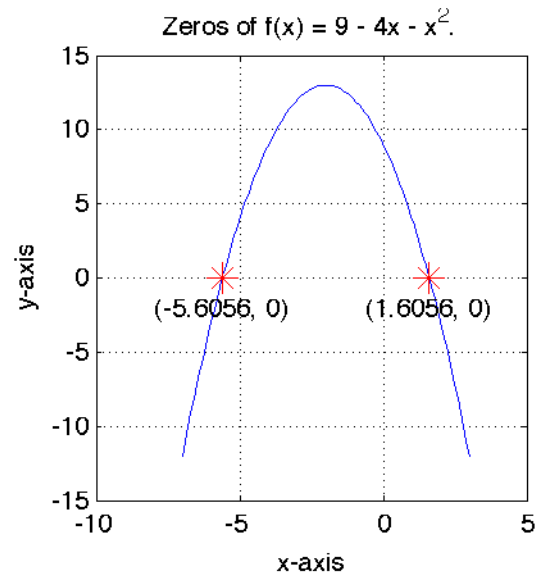
hText1=text(z1,f(z1)-1, strcat('(' ,num2str(z1),', 0)'));
set(hText1,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')
```

```

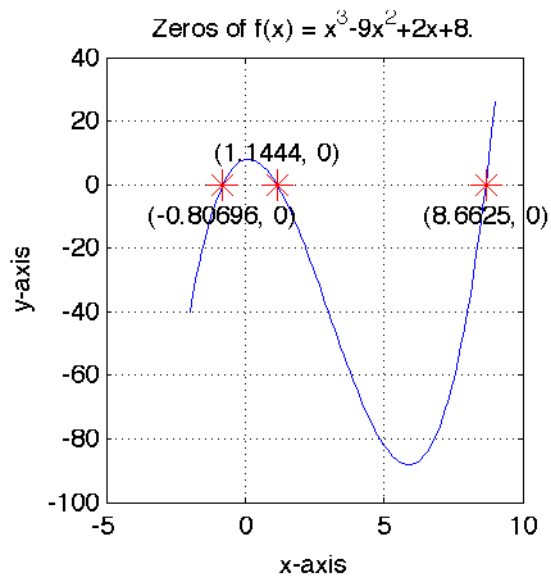
line(z2,f(z2),...
     'LineStyle','none',...
     'Marker','*',...
     'MarkerSize', 12,...
     'Color','r')

hText1=text(z2,f(z2)-1, strcat('(' ,num2str(z2),',', 0)'));
set(hText1,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')

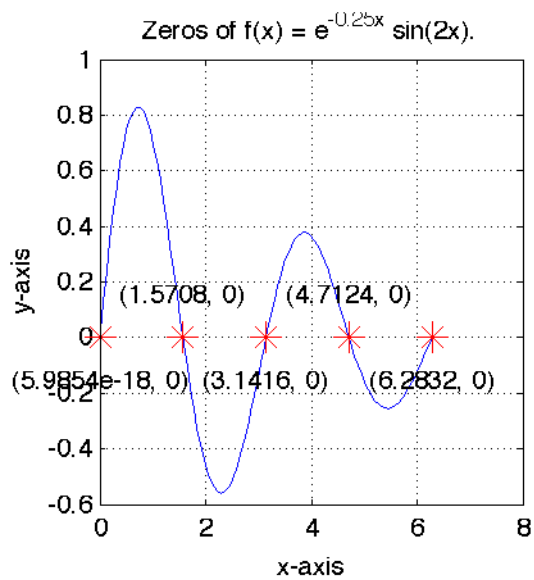
```



3.



5.



7. The following code was used to produce the solutions shown in the figure that follows.

```
%% Exercise #7
close all
clear all
clc

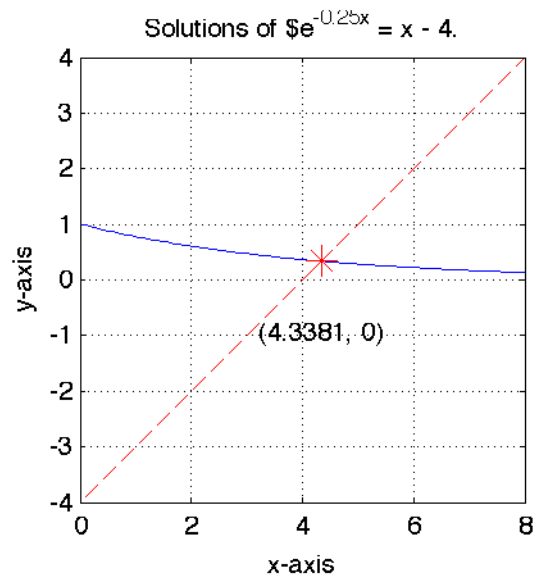
f=@(x) exp(-0.25*x)-x+4;
x=linspace(0,8);
y=f(x);
plot(x,y)
grid on

xlabel('x-axis')
ylabel('y-axis')
title('Solutions of  $e^{-0.25x} = x - 4.$ ')

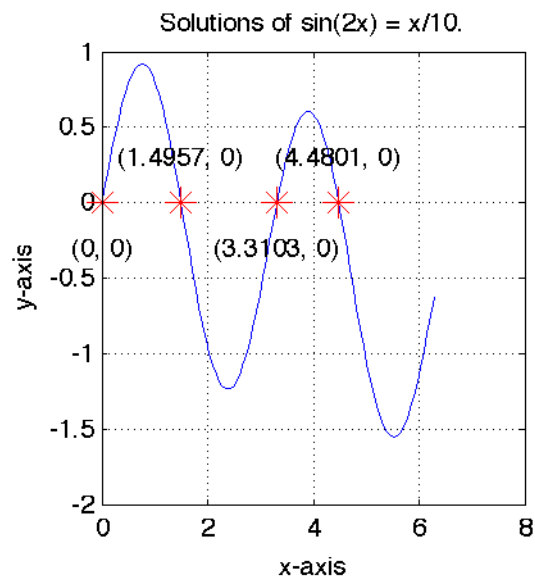
z1=fzero(f, [4,5]);

line(z1,f(z1),...
     'LineStyle','none',...
     'Marker','*',...
     'MarkerSize', 12,...
     'Color','r')

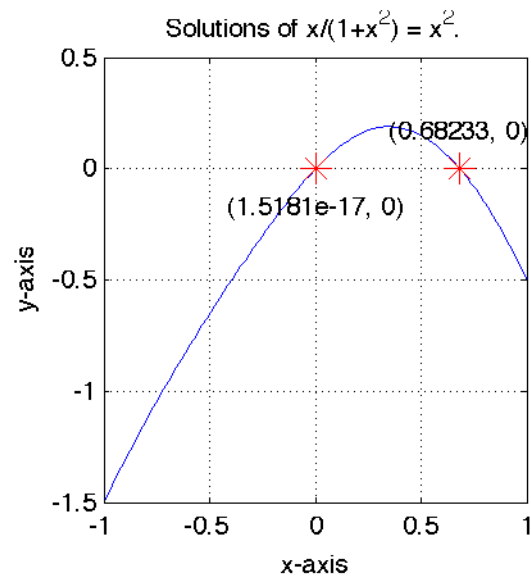
hText1=text(z1,f(z1)-1, strcat('( ',num2str(z1),' ', 0)'));
set(hText1,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')
```



9.



11.



13. The following code produces the image that follows.

```

%% Exercise #13
close all
clear all
clc

f=@(x) 9-x.^2;
g=@(x) 2*x+1;
x=linspace(-5,5);
plot(x,f(x),'b-',x,g(x),'r--')
grid on
xlabel('x-axis')
ylabel('y-axis')
title('Solutions of  $9 - x^2 = 2x + 1$ .')

h=@(x) f(x)-g(x);
z1=fzero(h, [-5,-3]);
z2=fzero(h, [1,3]);

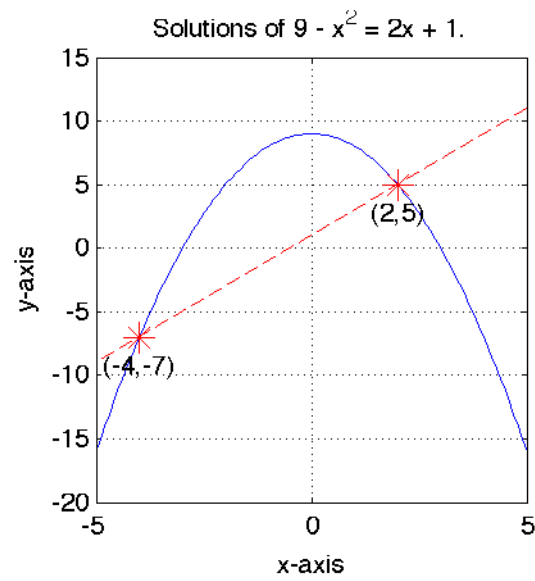
line(z1,f(z1),...
     'LineStyle','none',...
     'Marker','*',...
     'MarkerSize', 12,...
     'Color','r')

textStr1=strcat('(' ,num2str(z1),',',',',num2str(f(z1)),',')');
hText1=text(z1,f(z1)-1, textStr1);
set(hText1,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')

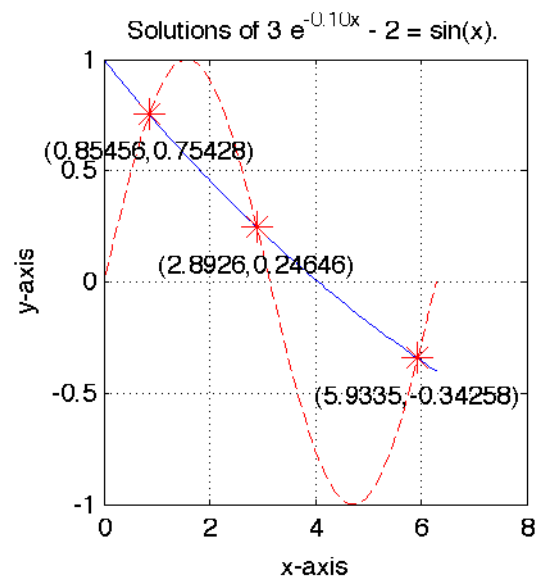
line(z2,f(z2),...
     'LineStyle','none',...
     'Marker','*',...
     'MarkerSize', 12,...
     'Color','r')

textStr2=strcat('(' ,num2str(z2),',',',',num2str(f(z2)),',')');
hText2=text(z2,f(z2)-1, textStr2);
set(hText2,...
    'HorizontalAlignment','center',...
    'VerticalAlignment','top')

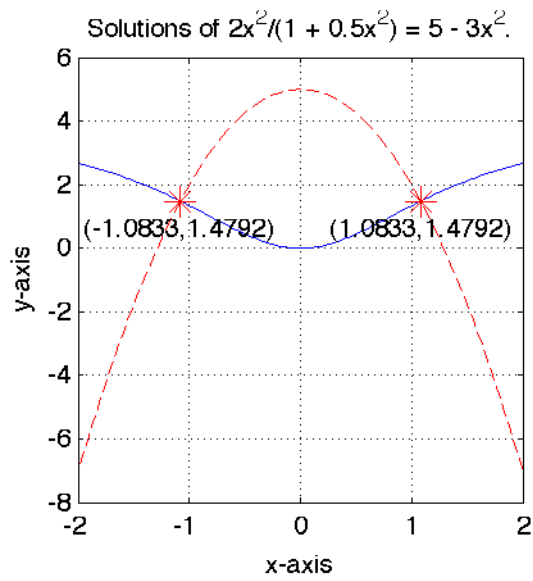
```



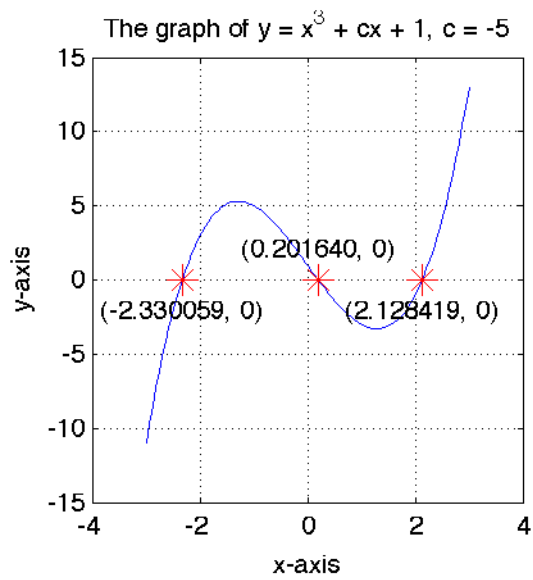
15.



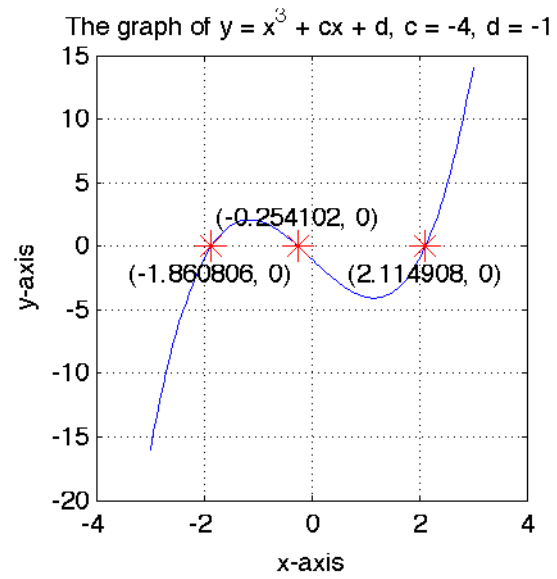
17.



19.



21.



22.

