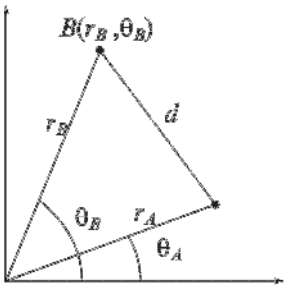
Another example of using an anonymous function with several arguments is shown in Sample Problem 6-3.

### Sample Problem 7-3: Distance between points in polar coordinates

Write an anonymous function that calculates the distance between two points in a plane when the position of the points is given in polar coordinates. Use the anonymous function to calculate the distance between point $A$ (2, π/6) and point $B$ (5, 3π/4).




**Solution**

The distance between two points in polar coordinates can be calculated by using the Law of Cosines:

$$d = \sqrt{r_A^2 + r_B^2 - 2r_A r_B \cos(\theta_A - \theta_B)}$$

The formula for the distance is entered as an anonymous function with four input arguments $(r_A, \theta_A, r_B, \theta_B)$. Then the function is used for calculating the distance between points $A$ and $B$.

```
>> d= @ (rA,thetA,rB,thetB) sqrt(rA^2+rB^2-2*rA*rB*cos(thetB-thetA))

d =
  @(rA,thetA,rB,thetB)sqrt(rA^2+rB^2-2*rA*rB*cos(thetB-thetA))
>> DistAtoB = d(2,pi/6,5,3*pi/4)
DistAtoB =
    5.8461
```

List of input arguments.

The arguments are typed in the order defined in the function.

## 7.9 *Function Functions*

There are many situations where a function (Function $A$) works on (uses) another function (Function $B$). This means that when Function $A$ is executed, it has to be provided with Function $B$. A function that accepts another function is called in MATLAB a function function. For example, MATLAB has a built-in function called `fzero` (Function $A$) that finds the zero of a math function $f(x)$ (Function $B$) — i.e., the value of $x$ where $f(x) = 0$. The program in the function `fzero` is written such that it can find the zero of any $f(x)$. When `fzero` is called, the specific function to be solved is passed into `fzero`, which finds the zero of the $f(x)$. (The function `fzero` is described in detail in Chapter 9.)

A function function, which accepts another function (imported function), includes in its input arguments a name that represents the imported function. The imported function name is used for the operations in the program (code) of the function function. When the function function is used (called), the specific function that is imported is listed in its input argument. In this way different functions can be imported (passed) into the function function. There are two methods for listing the name of an imported function in the argument list of a function function. One is by using a function handle (Section 7.9.1), and the other is by typing the name of the function that is being passed in as a string expression (Section 7.9.2). The method that is used affects the way that the operations in the function function are written (this is explained in more detail in the next two sections). Using function handles is easier and more efficient, and should be the preferred method.

### *7.9.1 Using Function Handles for Passing a Function into a Function Function*

Function handles are used for passing (importing) user-defined functions, built-in functions, and anonymous functions into function functions that can accept them. This section first explains what a function handle is, then shows how to write a user-defined function function that accepts function handles, and finally shows how to use function handles for passing functions into function functions.

**Function handle:**

A function handle is a MATLAB value that is associated with a function. It is a MATLAB data type and can be passed as an argument into another function. Once passed, the function handle provides means for calling (using) the function it is associated with. Function handles can be used with any kind of MATLAB function. This includes built-in functions, user-defined functions (written in function files), and anonymous functions.

- For built-in and user-defined functions, a function handle is created by typing the symbol @ in front of the function name. For example, `@cos` is the function handle of the built-in function `cos`, and `@FtoC` is the function handle of the user-defined function `FtoC` that was created in Sample Problem 7-2.
- The function handle can also be assigned to a variable name. For example, `cosHandle=@cos` assigns the handle `@cos` to `cosHandle`. Then the name `cosHandle` can be used for passing the handle.
- As anonymous functions (see Section 7.8.1), their name is already a function handle.

**Writing a function function that accepts a function handle as an input argument:**

As already mentioned, the input arguments of a function function (which accepts another function) includes a name (dummy function name) that rep-

resents the imported function. This dummy function (including a list of input arguments enclosed in parentheses) is used for the operations of the program inside the function function.

- The function that is actually being imported must be in a form consistent with the way that the dummy function is being used in the program. This means that both must have the same number and type of input and output arguments.

The following is an example of a user-defined function function, named `funplot`, that makes a plot of a function (any function $f(x)$ that is imported into it) between the points $x = a$ and $x = b$. The input arguments are `(Fun,a,b)`, where `Fun` is a dummy name that represents the imported function, and `a` and `b` are the end points of the domain. The function `funplot` also has a numerical output `xyout`, which is a $2 \times 3$ matrix with the values of $x$ and $f(x)$ at the three points $x = a$, $x = (a+b)/2$, and $x = b$. Note that in the program, the dummy function `Fun` has one input argument `(x)` and one output argument `y`, which are both vectors.

A name for the function that is passed in.

```
function xyout=funplot(Fun,a,b)
% funplot makes a plot of the function Fun which is passed in
% when funplot is called in the domain [a, b].
% Input arguments are:
% Fun:  Function handle of the function to be plotted.
% a:  The first point of the domain.
% b:  The last point of the domain.
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in a 3 by 2 matrix.

x=linspace(a,b,100);
y=Fun(x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=Fun((a+b)/2);
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

As an example, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ over the domain $[0.5, 4]$ is passed into the user-defined function `funplot`. This is done in two ways: first by writing a user-defined function for $f(x)$, and then by writing $f(x)$ as an anonymous function.

**Passing a user-defined function into a function function:**

First, a user-defined function is written for $f(x)$. The function, named `Fdemo`, calculates $f(x)$ for a given value of $x$ and is written using element-by-element operations.

```
function y=Fdemo(x)
y=exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3;
```

Next, the function `Fdemo` is passed into the user-defined function function `funplot`, which is called in the Command Window. Note that a handle of the user-defined function `Fdemo` is entered (the handle is `@Fdemo`) for the input argument `Fun` in the user-defined function `funplot`.

```
>> ydemo=funplot(@Fdemo,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Enter a handle of the user-defined function `Fdemo`.

In addition to the display of the numerical output, when the command is executed, the plot shown in Figure 7-3 is displayed in the Figure Window.

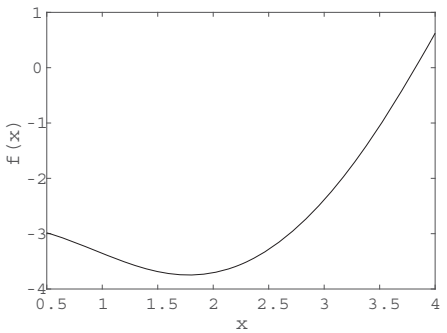



**Figure 7-3: A plot of the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$.**

**Passing an anonymous function into a function function:**

To use an anonymous function, the function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ first has to be written as an anonymous function, and then passed into the user-defined function `funplot`. The following shows how both of these steps are done in the Command Window. Note that the name of the anonymous function `FdemoAnony` is entered without the @ sign for the input argument `Fun` in the user-defined function `funplot` (since the name is already the handle of the anonymous function).

```
>> FdemoAnony=@(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3
FdemoAnony =
    @(x) exp(-0.17*x).*x.^3-2*x.^2+0.8*x-3

>> ydemo=funplot(FdemoAnony,0.5,4)
ydemo =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

Create an anonymous function for $f(x)$.

Enter the name of the anonymous function (`FdemoAnony`).

In addition to the display of the numerical output in the Command Window, the plot shown in Figure 7-3 is displayed in the Figure Window.

### *7.9.2 Using a Function Name for Passing a Function into a Function Function*

A second method for passing a function into a function function is by typing the name of the function that is being imported as a string in the input argument of the function function. The method that was used before the introduction of function handles can be used for importing user-defined functions. As mentioned, function handles are easier to use and more efficient and should be the preferred method. Importing user-defined functions by using their name is covered in the present edition of the book for the benefit of readers who need to understand programs written before MATLAB 7. New programs should use function handles.

When a user-defined function is imported by using its name, the value of the imported function inside the function function has to be calculated with the `feval` command. This is different from the case where a function handle is used, which means that there is a difference in the way that the code in the function function is written that depends on how the imported function is passed in.

**The `feval` command:**

The `feval` (short for "function evaluate") command evaluates the value of a function for a given value (or values) of the function's argument (or arguments). The format of the command is:

```
variable = feval('function name', argument value)
```

The value that is determined by `feval` can be assigned to a variable, or if the command is typed without an assignment, MATLAB displays `ans =` and the value of the function.

- The function name is typed as string.
- The function can be a built-in or a user-defined function.
- If there is more than one input argument, the arguments are separated with commas.

- If there is more than one output argument, the variables on the left-hand side of the assignment operator are typed inside brackets and separated with commas.

Two examples using the `feval` command with built-in functions follow.

```
>> feval('sqrt',64)
ans =
     8
>> x=feval('sin',pi/6)
x =
    0.5000
```

The following shows the use of the `feval` command with the user-defined function `loan` that was created earlier in the chapter (Figure 7-2). This function has three input arguments and two output arguments.

```
                                        A $50,000 loan, 3.9% interest, 10 years.
>> [M,T]=feval('loan',50000,3.9,10)
M =
        502.22                          Monthly payment.
T =
      60266.47                          Total payment.
```

**Writing a function function that accepts a function by typing its name as an input argument:**

As already mentioned, when a user-defined function is imported by using its name, the value of the function inside the function function has to be calculated with the `feval` command. This is demonstrated in the following user-defined function function that is called `funplotS`. The function is the same as the function `funplot` from Section 7.9.1, except that the command `feval` is used for the calculations with the imported function.

```
                                        A name for the function that is passed in.
function xyout=funplotS(Fun,a,b)
% funplotS makes a plot of the function Fun which is passed
in
% when funplotS is called in the domain [a, b].
% Input arguments are:
% Fun: The function to be plotted. Its name is entered as
string expression.
% a:  The first point of the domain.
% b:  The last point of the domain.
```

```
% Output argument is:
% xyout: The values of x and y at x=a, x=(a+b)/2, and x=b
% listed in a 3 by 2 matrix.

x=linspace(a,b,100);
y=feval(Fun,x);
xyout(1,1)=a; xyout(2,1)=(a+b)/2; xyout(3,1)=b;
xyout(1,2)=y(1);
xyout(2,2)=feval(Fun,(a+b)/2);
xyout(3,2)=y(100);
plot(x,y)
xlabel('x'), ylabel('y')
```

Using the imported function to calculate $f(x)$ at 100 points.

Using the imported function to calculate $f(x)$ at the midpoint.

**Passing a user-defined function into another function by using a string expression:**

The following demonstrates how to pass a user-defined function into a function function by typing the name of the imported function as a string in the input argument. The function $f(x) = e^{-0.17x}x^3 - 2x^2 + 0.8x - 3$ from Section 7.9.1, created as a user-defined function named `Fdemo`, is passed into the user-defined function `funplotS`. Note that the name `Fdemo` is typed in a string for the input argument `Fun` in the user-defined function `funplotS`.

```
>> ydemoS=funplotS('Fdemo',0.5,4)
ydemoS =
    0.5000   -2.9852
    2.2500   -3.5548
    4.0000    0.6235
```

The name of the imported function is typed as a string.

In addition to the display of the numerical output in the Command Window, the plot shown in Figure 7-3 is displayed in the Figure Window.

## *7.10 SUBFUNCTIONS*

A function file can contain more than one user-defined function. The functions are typed one after the other. Each function begins with a function definition line. The first function is called the primary function and the rest of the functions are called subfunctions. The subfunctions can be typed in any order. The name of the function file that is saved should correspond to the name of the primary function. Each of the functions in the file can call any of the other functions in the file. Outside functions, or programs (script files), can call only the primary function. Each of the functions in the file has its own workspace, which means that in each the variables are local. In other words, the primary function and the subfunctions cannot access each other's variables (unless variables are