Elementary mathematical functions in MATLAB

By Gilberto E. Urroz, August 2004

This document includes an introduction to the use of elementary mathematical functions in MATLAB. The use of these functions will be illustrated through simple calculations, as well as with plots of the functions.

Plotting functions of the form *y = f(x)*

Functions of the form *y = f(x)* can be presented graphically by using the MATLAB graphics commands *plot*, as illustrated next.

Function *plot* can be used to produce graphs of elementary functions. The simplest call to function *plot* is

```
plot(x,y)
```

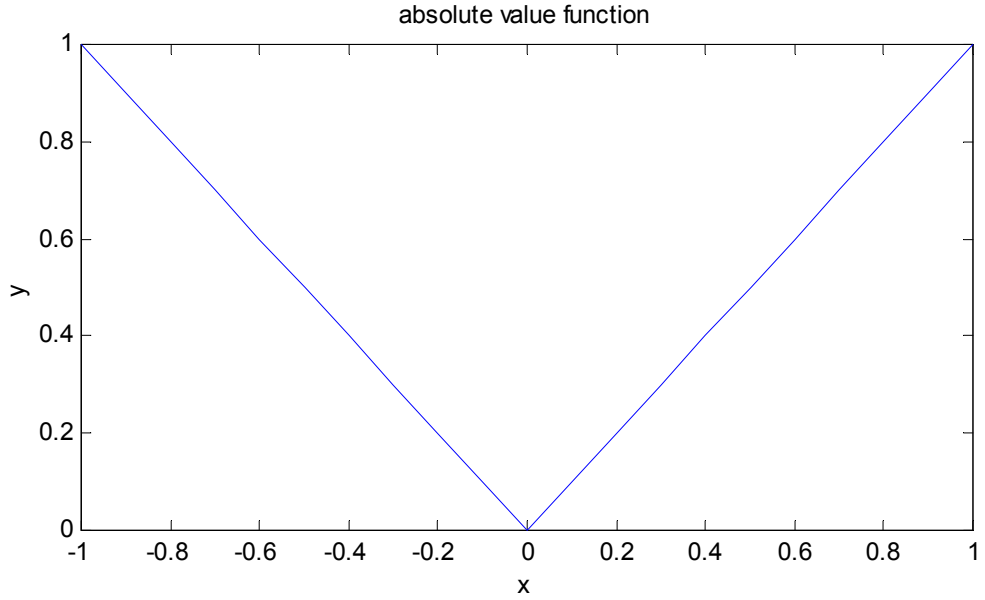
where *x* and *y* are vectors of the same length and $y_i = f(x_i)$. As an example, consider the plot of the function absolute value, *abs()*, in the range *-1* < *x* < *1*:

```
» x = [-1:0.1:1];
» plot(x,abs(x));
```

A plot title and axes labels can be added by using the following commands:

```
» title('absolute value function'); %adds title
» xlabel('x');ylabel('y');          %adds axes labels
```

The resulting graph is shown next:



Function *fplot* can be used to plot user-defined functions. The simplest call to function *fplot* is

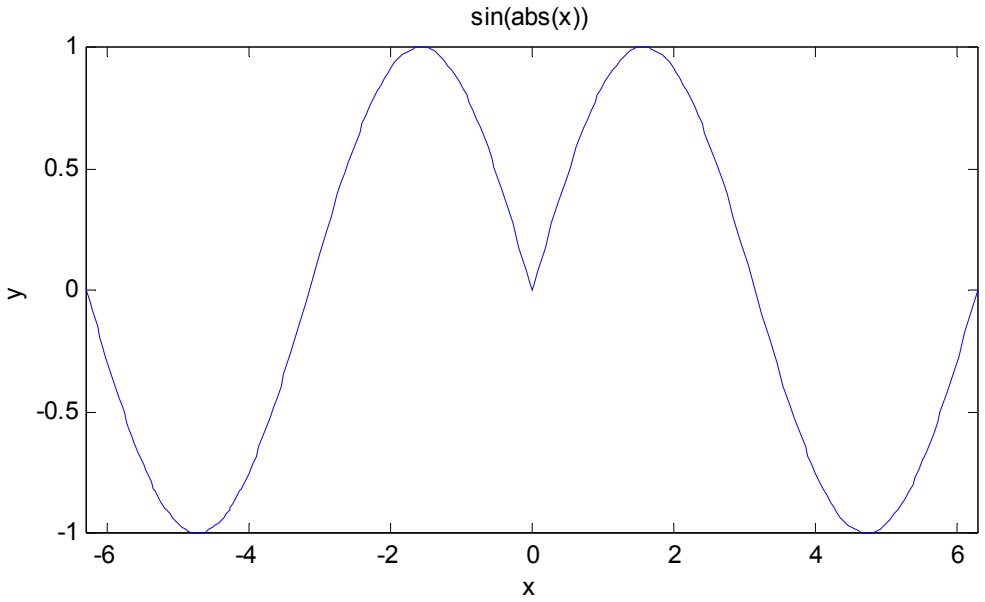
```
fplot(f,xlim)
```

where *f* is a function name and *xlim* is a vector of two values representing the lower and upper limits of the independent variable. The function *f* must be a user-defined function. A user-defined function is a combination of elementary functions defined by using the command *deff* (line-defined function) or a file function.

Consider the following example in which the function $y = sin(|x|)$ is plotted in the range $-2\pi < x < 2\pi$:

```
» f = inline('sin(abs(x))');        % define function y = sin(abs(x))
» xlim = [-2*pi,2*pi];              % define x vector
» fplot(f,xlim)                     % produce plot
» title('sin(abs(x))');             % add title
» xlabel('x'); ylabel('y');         % add labels
```

The plot is shown next:



Elementary number functions

Under this heading we include a number of functions that cannot be listed under trigonometric, exponential and logarithmic, or hyperbolic functions. These include:

`abs` - absolute value of a real number or magnitude of a complex number
`ceil` - ceiling function: rounds up a floating-point number
`floor` - rounds down a floating-point number
`fix` - rounds a floating-point number towards zero
`mod` - arithmetic remainder function
`rem` - remainder after division
`rat` - rational approximation of a floating-point number
`round` - rounds to nearest integer
`sign` - sign function
`sqrt` - square root

To illustrate the use of the function *round*, consider the multiplication of a 3x3 matrix by its inverse. First, we define the matrix **A**:

```
» A = [1,3,2;-1,2,1;4,2,1]

A =

     1     3     2
    -1     2     1
     4     2     1
```

Next, we calculate $\mathbf{B} = \mathbf{A} \cdot \mathbf{A}^{-1}$, which should produce a 3x3 identity matrix:

```
» B = A*inv(A)

B =

    1.0000   -0.0000   -0.0000
         0    1.0000         0
         0   -0.0000    1.0000
```

Due to the introduction of small numerical errors in the calculation of the inverse matrix, some elements off the diagonal that should be zero are shown as small, but nonzero, numbers. Function *round* will round those numbers to zero, thus "cleaning" the result to produce the required identity matrix:

```
» B = A*inv(A)

B =

    1.0000   -0.0000   -0.0000
         0    1.0000         0
         0   -0.0000    1.0000
```

Examples of functions *ceil*, *floor*, *fix*, and *round* are presented next:

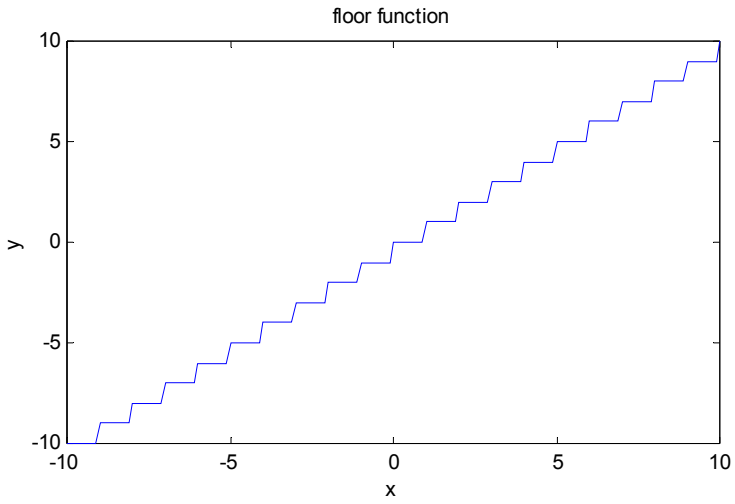
```
»ceil(-1.2)     //rounds up
 ans  = - 1

»ceil(1.2)      //rounds up
 ans  = 2

»floor(-1.2)    //rounds down
 ans  = - 2

»floor(1.2)     //rounds down
 ans  = 1

»round([1.25 1.50 1.75]) //rounds up or down
 ans  =

   1    2    2

-->fix([1.25 1.50 1.75]) //rounds down
 ans  =

   1    1    1
```

Notice that functions *floor* and *fix* both round down to the nearest integer number. Thus, either function can be used to find the integer part of a floating-point number. The fractional or decimal part of a floating-number can be calculated by simply subtracting the number from its integer part, for example:

```
» x = 12.4345

x = 12.4345

» fix(x)

ans = 12

» x-fix(x)

ans = 0.4345
```

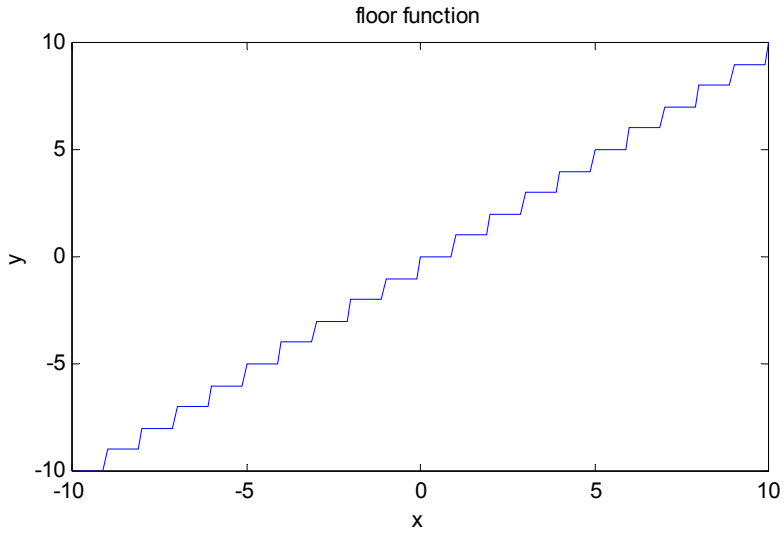
A plot of the function *floor* is shown next. Here are the commands required to produce the plot:

```
-->x = [-10:0.1:10];plot(x,floor(x));
-->xtitle('floor function');xlabel('x');ylabel('y');
```



The plot of function *ceil* is very similar to that of function *floor*:

```
-->x = [-10:0.1:10];plot(x,ceil(x));
-->xtitle('ceil function');xlabel('x');ylabel('y');
```



Functions *ceil, floor, round* and *fix* can be applied to complex numbers. In this case, the effect of each function is applied separately to the real and imaginary parts of the complex number. Some examples are shown next:

```
» z = 2.3 - 5.2*i   % define a complex number

z =   2.3000 - 5.2000i

» ceil(z)

ans = 3.0000 - 5.0000i

» floor(z)

ans = 2.0000 - 6.0000i

» round(z)

ans = 2.0000 - 5.0000i

» fix(z)

ans = 2.0000 - 5.0000i
```

Note: A complex number *z* can be written as $z = x + iy$, where *x* and *y* are real numbers and $i = (-1)^{1/2}$ is the *unit imaginary number*. We say that *x* is the *real part* of *z*, or $x = Re(z)$, and that *y* is the imaginary part of *z*, or $y = Im(z)$. MATLAB provides functions *real* and *imag* to obtain the real and imaginary parts, respectively, of a complex number. Here is an example of their operation:

```
» z = 2.4 + 3.5*i      % define a complex number
z = 2.4000 + 3.5000i

» x = real(z)          % real part of z
x = 2.4000

» y = imag(z)          % imaginary part of z
y =  3.5000
```

Function *abs*, when applied to a real number, returns the absolute value of the number, i.e.,

$$|x| = \begin{cases} x, & if\ x \geq 0 \\ -x, & if\ x < 0 \end{cases}.$$

If applied to a complex number $z = x + iy$, the function *abs* returns the magnitude of the complex number, i.e., $|z| = (x^2+y^2)^{1/2}$. Some examples are shown next:

```
» abs([-3.4 0.0 3.4])          //absolute value of real numbers
 ans =    3.4000         0    3.4000

» abs(-3+4*i) % magnitude of a complex number
ans =  5
```

A plot of the function *abs(x)* in the range $-1 < x < 1$ was presented earlier.

Function *sign* returns either *1.0*, *-1.0* or *0.0* for a real number depending on whether the number is positive, negative, or zero. Here is an example:

```
» sign([-2.4 0.0 5.4])

ans =  -1     0     1
```

On the other hand, when applied to a complex number *z*, function *sign* returns the value z/|z|. For example:

```
» sign(3-48i)

ans = 0.0624 - 0.9981i
```

The resulting complex number has magnitude 1:

```
» abs(ans)

ans = 1.0000
```

The *mod* function is used to calculate the reminder of the division of two numbers. A call to the function of the form `mod(n,m)` produces the number

```
n - m .* fix (n ./ m)
```

where *n* and *m* can be vectors of real numbers. For example:

```
» mod(7.2,3.1)

ans = 1.0000
```

Which is the same as:

```
» 7.2-3.1*fix(7.2/3.1)

ans = 1
```

The *mod* function has applications in programming when used with integer numbers. In such case, the *mod* function represents the integer reminder of the division of two numbers. When dividing two integer numbers, *n* and *m*, we can write

$$\frac{n}{m} = q + \frac{r}{m}$$

where *q* is the quotient and *r* is the *reminder* of the division of *n* by *m*. Thus, *r = mod(n,m)*.

Consider the following example:

```
» mod([0 1 2 3 4 5 6],3)

ans =      0     1     2     0     1     2     0
```

Notice that the result of *mod(n,3)* for *n = 0, 1, 2, …, 6*, is *0, 1*, or *2*, and that *mod(n,3) = 0* whenever *n* is a multiple of 3. Thus, an integer number *n* is a multiple of another integer number *m* if *mod(n,m) = 0*.

Function *rat* is used to determine the numerator and denominator of the rational approximation of a floating-point number. The general call to the function is

```
[n,d] = rat(x)
```

where *x* is a floating-point number, and *n* and *d* are integer numbers such that $x \approx n/d$. For example:

```
» [n,d] = rat(0.353535)

n =  10086
d =  28529
```

To verify this result use:

```
» n/d

ans = 0.3535
```

Another example is used to demonstrate that *0.333… = 1/3*:

```
» [n,d]=rat(0.3333333333333)

n =  1
d =  3
```

Notice that you need to provide a relatively large number of decimals to force the result. If you use only four decimals for the floating-point number, the approximation is not correct:

```
» [n,d] = rat(0.3333)

n = 3333
d = 10000
```

Function *rat* can be called using a second argument representing the error, or tolerance, allowed in the approximation. The call to the function in this case is:

```
[n,d] = rat(x,ε)
```

where ε is the tolerance, and *n, d, x* where defined earlier. The values of *n* and *d* returned by this function call are such that

$$\left| \frac{n}{d} - x \right| \leq \varepsilon \mid x \mid$$

For example:

```
» [n,d] = rat(0.3333,1e-6)
n =   3333
d =  10000
```

In this case, a tolerance $\varepsilon = 0.0001 = 10^{-6}$ produces the approximation $0.3333 \approx 3333/1000$. If we change the tolerance to $\varepsilon = 0.001 = 10^{-3}$, the result changes to $0.3333 \approx 1/3$:

```
» [n,d] = rat(0.3333,1e-3)
n = 1
d = 3
```

The default value for the tolerance is $\varepsilon = 10^{-5}$. Thus, when the call *rat(x)* is used, it is equivalent to *rat(x,1e-5)*.

Function *sqrt* returns the square root of a real or complex number. The square root of a positive real number is a real number, for example:

```
» sqrt(2345.676)
 ans  = 48.4322
```

The square root of a negative real number ($x < 0$) is calculated as

$$\sqrt{x} = i\sqrt{-x}$$

where *i* is the unit imaginary number, for example:

```
» sqrt(-2345.676)
 ans  = 0 +48.4322i
```

To understand the calculation of the square root for a complex number we will use the *polar representation* of a complex number, namely,

$$z = r \cdot e^{i\theta},$$

where

$$r = (x^2+y^2)^{1/2}$$

is the magnitude of the complex number, and

$$\theta = tan^{-1}(y/x)$$

is the *argument* of the complex number.

Euler's formula provides an expression for the term $e^{i\theta}$, namely,

$$e^{i\theta} = \cos\theta + i \sin\theta.$$

With this result, we can write

$$z = r \cdot e^{i\theta} = r \cdot \cos\theta + i\, r \cdot \sin\theta = x + iy,$$

which implies that $x = r \cdot \cos\theta$, and $y = r \cdot \sin\theta$. Thus, when using $z = x+iy$ we are using a *Cartesian* representation for the complex number, while $z = r \cdot e^{i\theta}$ is the *polar* representation of the same number.

Using the polar representation for complex number z, we can write:

$$\sqrt{z} = \left(r \cdot e^{i\theta}\right)^{1/2} = \sqrt{r} \cdot e^{i(\theta/2)} = \sqrt{r} \cdot \cos(\theta/2) + i\sqrt{r} \cdot \sin(\theta/2)\,.$$

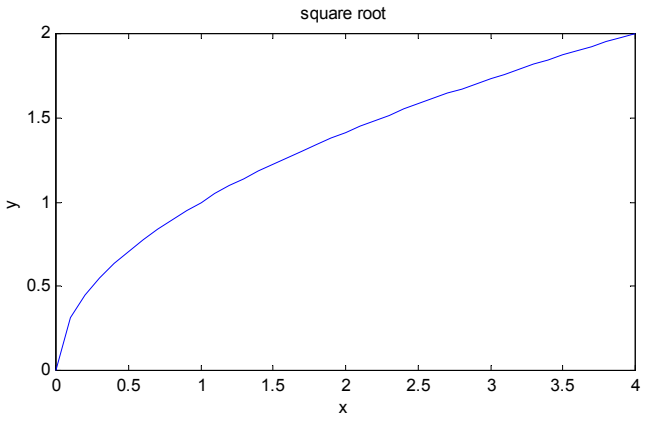
For example,

```
» sqrt(3-5*i)
ans = 2.1013 - 1.1897i
```

A plot of the square root function can only be produced for x > 0:

```
» x = [0:0.14];plot(x,sqrt(x));
» title('square root');xlabel('x');ylabel('y');
```

The resulting graph is:



Exponential and logarithmic functions

The exponential function, *exp*, returns the value of e^x for a real number ***x*** where *e* is the irrational number that constitutes the basis of the natural logarithm, *e = 2.7182818….* This value is calculated as *exp(1)*:

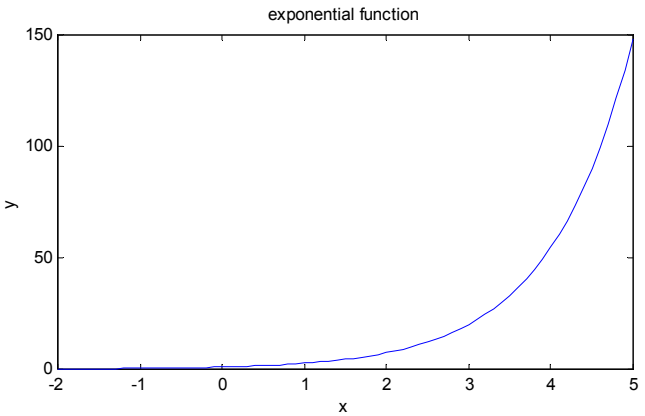
```
» exp(1)
ans = 2.7183
```

Examples of evaluating the exponential function with real arguments are presented next:

```
» exp(-2.3)
ans = 0.1003

» exp([-1 0 1 2 3])
ans = 0.3679    1.0000    2.7183    7.3891   20.0855
```

A plot of the function is produced next:

```
-->x = [-2:0.1:5];plot(x,exp(x));
-->xtitle('exponential function');xlabel('x');ylabel('y');
```



The exponential function applied to a complex number can be easily interpreted as follows

$$\exp(z) = e^{x+iy} = e^x e^{iy} = e^x(\cos y + i \sin y) = e^x \cos y + ie^x \sin y .$$

For example:

```
» exp(3+4*i)
ans = -13.1288 -15.2008i
```

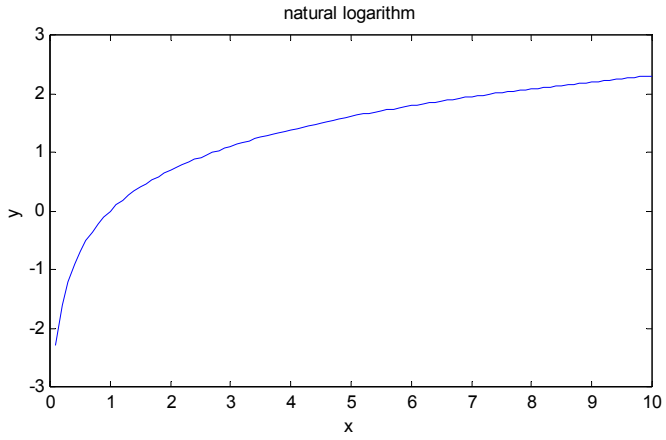
The inverse function to the exponential function is the natural logarithm, *ln(x)*, such that if ***y*** = *ln(x)*, then ***x*** = *exp(y)* = e^y. Thus, *ln(exp(x))* = ***x***, and *exp(ln(x))* = ***x***.

In MATLAB, the natural logarithm function is referred to as *log()*. Examples of calculation of *log()* for real arguments are shown next:

```
» log(2.35)

ans =  0.8544

» log([1 2 3])

ans =  0    0.6931    1.0986
```

A plot of the natural logarithm function is shown below for the range *0* < ***x*** < *10* (the natural log function is not defined in the real of real numbers for $x \leq 0$):

```
» x = [0.1:0.1:10];plot(x,log(x));
» title('natural logarithm');xlabel('x');ylabel('y');
```



If $z = r\,e^{i\theta}$ is a complex number, then *ln(z)* is interpreted as follows

$$\ln(z) = \ln(r \cdot e^{i\theta}) = \ln(r) + \ln(e^{i\theta}) = \ln(r) + i\theta .$$

Calculations of *log()* for complex arguments in MATLAB are shown below:

```
» log(2+3*i)
ans = 1.2825 + 0.9828i

» log([1+i, 2+3*i, 1-i])
ans =  0.3466 + 0.7854i   1.2825 + 0.9828i   0.3466 - 0.7854i
```

Besides the natural logarithm function (or logarithms of base *e*), MATLAB also provide functions *log10()* and *log2()* that calculate logarithms of base 10 and of base 2, respectively. These two functions are defined in a similar manner to the natural logarithmic, i.e.,

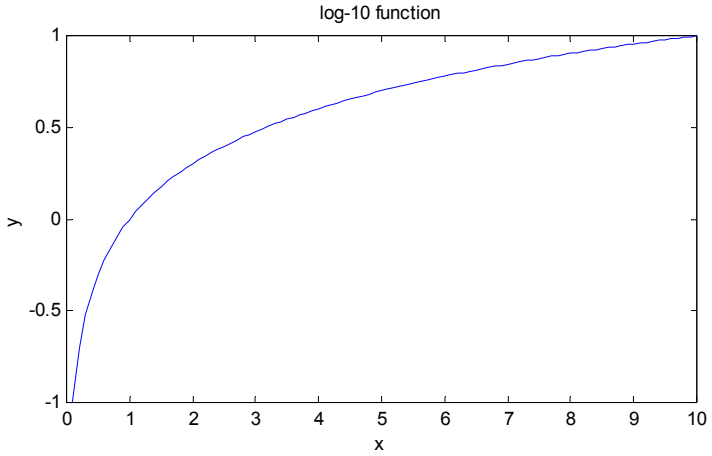
- Logarithms of base 10: if $x = 10^y$, then $y = log_{10}(x)$
- Logarithms of base 2 : if $x = 2^y$, then $y = log_2(x)$

Examples of applications of functions *log10()* and *log2()* with real arguments are shown next:

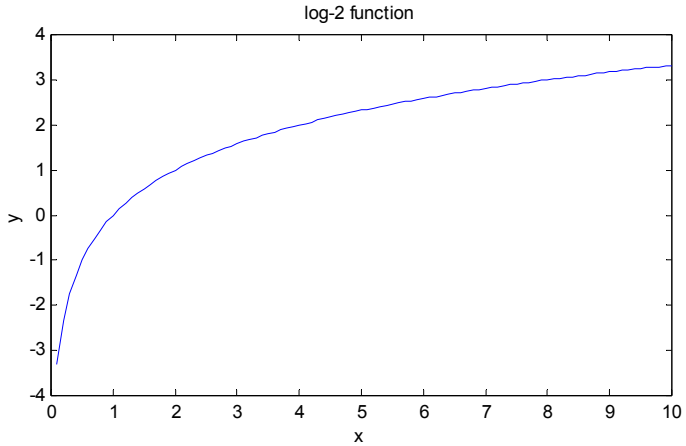
```
» log10([100 200 1000 1500])
ans =  2.0000    2.3010    3.0000    3.1761

» log2([8 23 256 1000])
ans = 3.0000    4.5236    8.0000    9.9658
```

Plots of the functions *log10(x)*, and *log2(x)* are shown next:

```
» x=[0.1:0.1:10]; plot(x,log10(x));
» title('log-10 function');xlabel('x');ylabel('y');
```




```
» x=[0.1:0.1:10]; plot(x,log2(x));
» xtitle('log-2 function');xlabel('x');ylabel('y');
```



Logarithms of any base can be expressed in terms of the natural logarithm as follows: let $x = a^y$, then $y = log_a(x)$. Also, $ln(x) = ln(a^y) = y\ ln(a)$, and $y = ln(x)/ln(a)$, i.e.,

$$log_a(x) = ln(x)/ln(a).$$

Thus, the logarithm of any real base *a* of a complex number *z* = *x* + *iy* can be calculated as

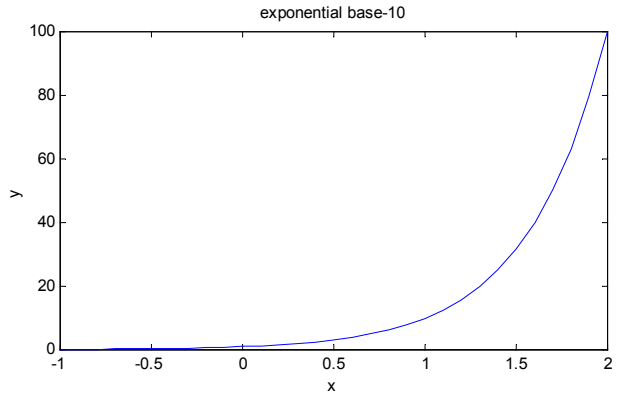
$$\log_a(z) = \frac{\ln(z)}{\ln(a)} = \frac{\ln(r \cdot e^{i\theta})}{\ln(a)} = \frac{\ln(r) + \ln(e^{i\theta})}{\ln(a)} = \frac{\ln(r)}{\ln(a)} + i\frac{\theta}{\ln(a)} = \log_a(r) + i\frac{\theta}{\ln(a)}.$$

Examples of applications of functions *log10()* and *log2()* with complex arguments are shown next:

```
» log10(5-3*i)
ans = 0.7657 - 0.2347i

» log2(4+6*i)
ans = 2.8502 + 1.4179i
```

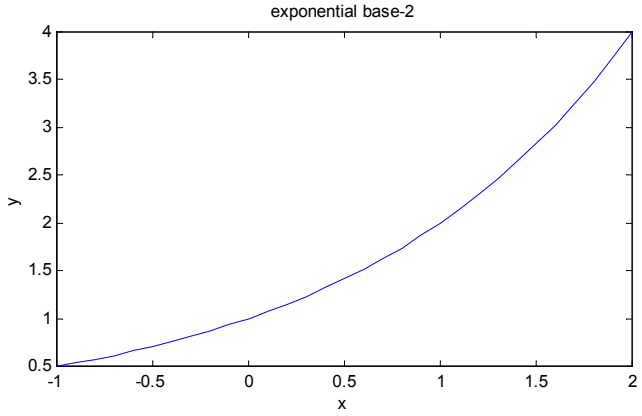
The inverse function of the logarithm base-*a* function, $log_a(x)$, is the exponential base-*a* function, a^x, so that $log_a(a^x) = x$ and $a^{log_a x} = x$. Thus, a plot of the exponential base-*10* function is produced as follows:

```
» x = [-1:0.1:2]; plot(x,10^x);
» title('exponential base-10');xlabel('x');ylabel('y');
```



while a plot of the exponential base-*2* function is produced as follows:

```
» x = [-1:0.1:2]; plot(x,2^x);
» title('exponential base-2');xlabel('x');ylabel('y');
```



In producing this graphics we used the exponentiation operation (^) with real base and exponent. You can use a real base and complex exponent to calculate an exponentiation that will produce a complex number according to the expression

$$a^z = a^{x+iy} = a^x a^{iy} = a^x (e^{\ln(a)})^{iy} = a^x e^{iy\ln(a)} = a^x(\cos(y\ln(a)) + i\sin(y\ln(a))) =$$

$$a^x \cos(y\ln(a)) + ia^x \sin(y\ln(a)).$$

Examples of exponentiation of a real base and a complex exponent are shown next:

```
» 2^(3-5*i)
ans = -7.5834 + 2.5480i

» 10^(2+4*i)
ans = -97.7096 +21.2798i
```

The exponentiation of a complex base with a real exponent follows the following rule:

$$z^a = (x+iy)^a = (r\cdot e^{i\theta})^a = r^a e^{ia\theta} = r^a(\cos(a\theta) + i\sin(a\theta)) = r^a \cos(a\theta) + ir^a \sin(a\theta).$$

Examples of exponentiation of a complex base with a real exponent follow:

```
» (3.25+6.2*i)^4
ans = -8.4693e+002 -2.2469e+003i

» (2.17-6*i)^3.2
ans = -2.6896e+002 +2.6314e+002i
```

Note: A special case of this rule was used earlier when defining the square root of a complex number by taking *a* = ½.

The most general case of exponentiation involves a complex base and a complex exponent.

$$z_1^{z_2} = (r_1 e^{i\theta_1})^{x_2+iy_2} = (r_1 e^{i\theta_1})^{x_2} \cdot (r_1 e^{i\theta_1})^{iy_2} = (r_1\cos(\theta_1) + ir_1\sin(\theta_1))^{x_2} (r_1^{iy_2} e^{-y_2\theta_2}).$$

To simplify this result we look at the term $r_1^{iy_1}$, which can be expanded as

$$r_1^{iy_1} = (e^{\ln(r_1)})^{iy_2} = e^{iy_1\ln(r_1)} = \cos(y_1\ln(y_2)) + i\sin(y_1\ln(y_2)).$$

Thus, the exponentiation operation results in

$$z_1^{z_2} = r_1^{iy_2} \cdot (r_1\cos(\theta_1) + ir_1\sin(\theta_1))^{x_2} \cdot (\cos(y_1\ln(y_2)) + i\sin(y_1\ln(y_2))).$$

The first term in the resulting product is simply a real exponentiation, $r_1^{y_2}$, the second term is a complex number raised to a real exponent, $(r_1\cos(\theta_1) + ir_1\sin(\theta_1))^{x_2}$, and the third term is simply a complex number, $\cos(y_1\ln(y_2)) + i\sin(y_1\ln(y_2))$. Thus, the original exponentiation problem, $z_1^{z_2}$, gets reduced to a product of complex numbers times a real number.

The rules of multiplication of complex numbers are such that

$$z_1 \cdot z_2 = (x_1 + iy_1)(x_2 + iy_2) = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 - x_2 y_1)$$

or, alternatively,

$$z_1 \cdot z_2 = (r_1 e^{i\theta_1})(r_2 e^{i\theta_2}) = r_1 r_2 e^{i(\theta_1+\theta_2)} = r_1 r_2 (\cos(\theta_1 + \theta_2) + i \sin(\theta_1 + \theta_2)) =$$

$$r_1 r_2 \cos(\theta_1 + \theta_2) + i r_1 r_2 \sin(\theta_1 + \theta_2)$$

The rules of multiplying a real and a complex number are the following

$$az = a(x + iy) = ax + iay$$

or,

$$az = are^{i\theta} = ar\cos(\theta) + iar\sin(\theta)\,.$$

Using these rules of multiplication the exponentiation of a complex base with a complex exponent can be easily calculated.

Examples of the exponentiation of a complex base with a complex exponent are presented next:

```
» (-2.3+8.2*i)^(2*i)
ans = -0.0104 - 0.0227i

» (5.2-4*i)^(2+3*i)
ans = -1.1431e+002 -2.8571e+002i
```

Tables of logarithms were developed in the 1600's to facilitate calculations of multiplications, divisions and exponentiation. The rules used to calculate multiplications, divisions, and exponentiations are the following:

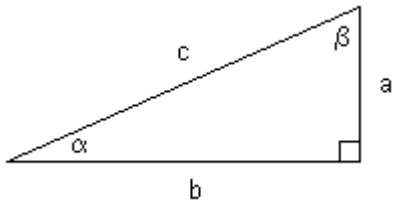
$\log_a(xy) = \log_a x + \log_a y$, $\log_a(x/y) = \log_a x - \log_a y$, and $\log_a(x^y) = y \log_a x$.

These rules follow from the properties of powers:

$a^{x+y} = a^x a^y$, $a^{x-y} = a^x a^{-y} = a^x / a^y$, and $(a^x)^y = a^{xy}$

Trigonometric functions

The six trigonometric functions are sine (*sin*), cosine (*cos*), tangent (*tan*), cotangent (*cot*), secant (*sec*), and cosecant *(csc*). The trigonometric functions can be defined in terms of the angles and sides of a right triangle. Consider the trigonometric functions of the angle α in the triangle shown below.



Side *c* is known as the *hypotenuse* of the right triangle. With respect to the angle α, side *a* is the *opposite side*, and side *b* is the *adjacent side*. The definitions of the trigonometric functions for angle α are as follows:

$$\sin(\alpha) = \frac{opposite\ side}{hypotenuse} = \frac{a}{c}, \quad \cos(\alpha) = \frac{adjacent\ side}{hypotenuse} = \frac{b}{c}$$

$$\tan(\alpha) = \frac{opposite\ side}{adjacent\ side} = \frac{a}{b}, \quad \cot(\alpha) = \frac{adjacent\ side}{opposite\ side} = \frac{b}{a}$$

$$\sec(\alpha) = \frac{hypotenuse}{adjacent\ side} = \frac{c}{b}, \quad \csc(\alpha) = \frac{hypotenuse}{opposite\ side} = \frac{c}{a}$$

From these definitions it follows that

$$sin(\alpha) = 1/csc(\alpha),\ cos(\alpha) = 1/sec(\alpha),$$

$$tan(\alpha) = 1/cot(\alpha),\ cot(\alpha) = 1/tan(\alpha),$$

$$sec(\alpha) = 1/cos\ (\alpha),\ csc(\alpha) = 1/sin(\alpha),$$

$$tan(\alpha) = sin(\alpha)/cos(\alpha),\ cot(\alpha) = cos(\alpha)/sin(\alpha)$$

The Pythagorean theorem indicates that for the right triangle shown earlier

$$a^2 + b^2 = c^2.$$

Dividing by c^2 results in $(a/c)^2 + (b/c)^2 = 1$, which can be re-written as

$$sin^2(\alpha) + cos^2(\alpha) = 1.$$

Dividing the Pythagorean theorem result by b^2 results in $(a/b)^2 + 1 = (c/b)^2$, or

$$tan^2(\alpha) + 1 = sec^2(\alpha).$$

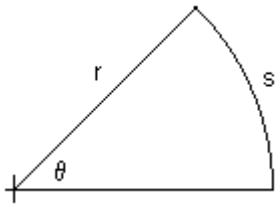
Finally, dividing the Pythagorean theorem result by a^2 results in $1 + (b/a)^2 = (c/a)^2$, or

$$1 + cot^2(\alpha) = csc^2(\alpha).$$

Angular measure

Angles can be measured in degrees (^{o}) so that a right angle constitutes ninety degrees (90^o) and the angle described by a radius rotating about a point so as to describe a complete circumference constitutes 360^o. A more natural way to measure angles is by the use of *radians*. Consider the arc of length *s* on a circle of radius *r* that corresponds to an angle θ^r in radians, by definition

$$\theta^r = s/r.$$



If the arc corresponds to a circumference, then $s = 2\pi r$, and the corresponding angle is $s/r = 2\pi$. Since this angle corresponds to 360^o, we can write

$$\frac{\theta^o}{\theta^r} = \frac{360}{2\pi} = \frac{180}{\pi}.$$

The value of π is available in MATLAB through the constant *pi*, i.e.,

```
» pi
ans = 3.1416
```

In the triangle presented earlier, if the angles α and β are given in radians, it follows that

$$\alpha + \beta = \pi/2.$$

If the angles are in degrees, the relationship is given by

$$\alpha + \beta = 90^o.$$

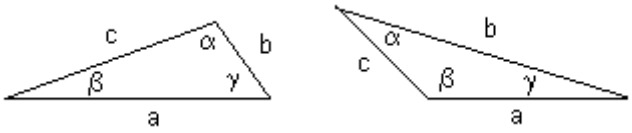
From the definitions of the trigonometric functions given earlier, it follows that

$$\sin(\pi/2-\alpha) = \cos(\alpha),\ \cos(\pi/2-\alpha) = \sin(\alpha),$$

$$\tan(\pi/2-\alpha) = \cot(\alpha),\ \cot(\pi/2-\alpha) = \tan(\alpha),$$

$$\sec(\pi/2-\alpha) = \csc(\alpha),\ \csc(\pi/2-\alpha) = \sec(\alpha)$$

Besides the calculations presented earlier for right triangles, trigonometric functions and their inverses can be used to solve for any triangle such as those shown in the following figure:



These triangles have sides *a, b,* and *c*, opposite to angles α, β, and γ, respectively. For the triangle to exist, $c + b > a$, or $c + a > b$, or $a + b > c$. The angles satisfy the relationship

$$\alpha + \beta + \gamma = 180^o, \text{ or } \alpha + \beta + \gamma = \pi^r.$$

The angles and sides are related by:

- the law of sines,

$$\frac{\sin(\alpha)}{a} = \frac{\sin(\beta)}{b} = \frac{\sin(\gamma)}{c}$$

- the law of cosines

$$c^2 = a^2 + b^2 - 2ab\cos(\gamma)$$
$$b^2 = a^2 + c^2 - 2ac\cos(\beta)$$
$$a^2 = b^2 + c^2 - 2bc\cos(\alpha)$$

Also, the area of the triangle is given by Heron's formula

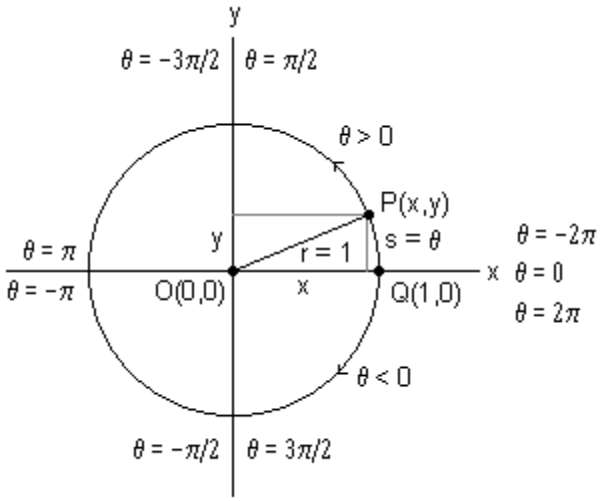
$$A = \sqrt{s(s-a)(s-b)(s-c)}\ ,$$

where

$$s = \frac{a+b+c}{2}$$

is known as the *semi-perimeter* of the triangle.

The trigonometric functions can also be given in terms of the coordinates of a point on a circle of unit radius, as illustrated in the following figure:



Since the circle has radius *r = 1*, the arc of the circle extending from point *Q(1,0)* to point *P(x,y)* has a length θ equal to the angle $\angle$POQ, swept by the radius of the circle, measured in radians, i.e., $\angle POQ = \theta^r$. The angle corresponding to line *OQ* is $\theta = 0$. An angle measured in the counterclockwise direction is, by convention, taken as a positive angle, i.e., $\theta > 0$, while one measured in the clockwise direction is taken as negative, i.e., $\theta < 0$. The figure shows the angles from $\theta = 0$ to $\theta = 2\pi$, in the positive sense, and also the angles from $\theta = 0$ to $\theta = -2\pi$,, in the negative sense. From the definition of polar coordinates, with *r =1*, it follows that the coordinates of point *P(x,y)* on the unit circle provide the values of the functions *sine* and *cosine* of θ, since *x = r cos(θ) = cos(θ)*, and *y = r sin(θ) = sin(θ)*. Thus, for a point *P(x,y)* in the unit circle

$$sin(\theta) = y,\ cos(\theta) = x,$$

$$tan(\theta) = y/x,\ cot(\theta) = x/y,$$

$$sec(\theta) = 1/x,\ csc(\theta) = 1/y.$$

Increasing or decreasing a given angle θ by a multiple of 2π will bring us back to the same point on the unit circle, therefore, we can write, for any integer value *k*, i.e., *k = 0, 1, 2, …*:

$$\sin(\theta \pm 2\pi k) = \sin(\theta),\ \cos(\theta \pm 2\pi k) = \cos(\theta),$$

$$\tan(\theta \pm 2\pi k) = \tan(\theta),\ \cot(\theta \pm 2\pi k) = \cot(\theta),$$

$$\sec(\theta \pm 2\pi k) = \sec(\theta),\ \csc(\theta \pm 2\pi k) = \csc(\theta).$$

Also, from the symmetries of the unit circle it follows that

$$\sin(-\theta) = -\sin(\theta),\ \cos(-\theta) = \cos(\theta),$$

$$\tan(-\theta) = -\tan(\theta),\ \cot(-\theta) = -\cot(\theta),$$

$$\sec(-\theta) = \sec(\theta),\ \csc(-\theta) = -\csc(\theta).$$

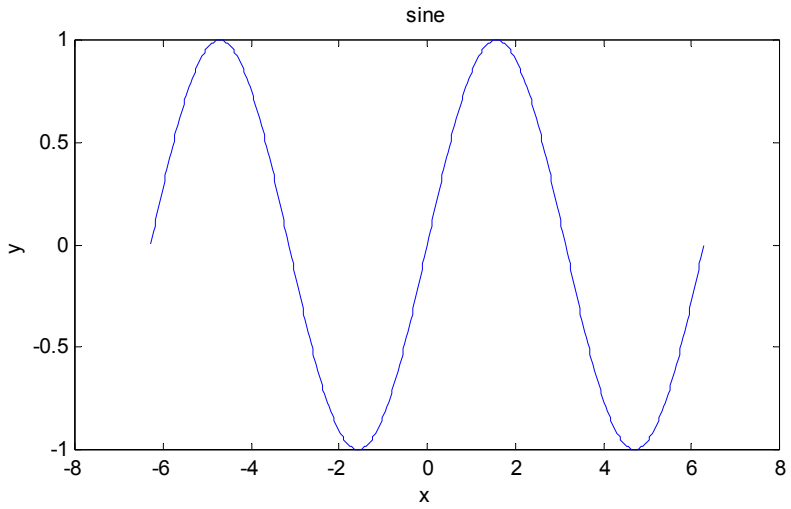
The relationships developed above are referred to as *trigonometric identities* and can be used to simplify trigonometric relationships.

The inverse trigonometric functions are defined as follows:

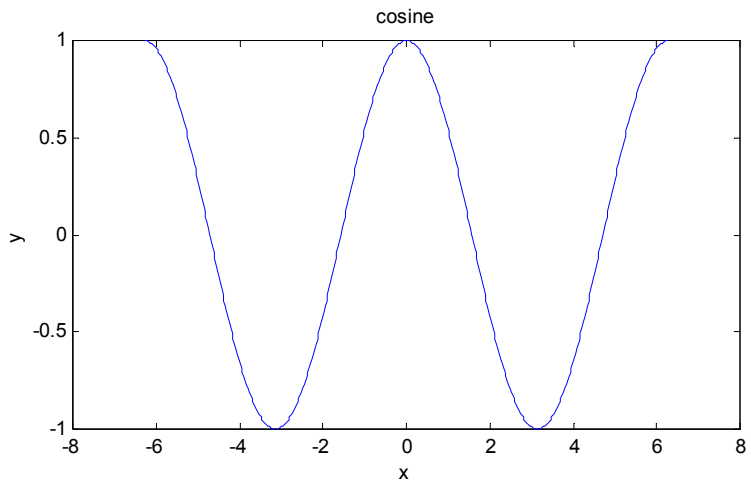
- If $\sin(y) = x$, then $y = \sin^{-1}(x) = \text{asin}(x)$; If $\cos(y) = x$, then $y = \cos^{-1}(x) = \text{acos}(x)$
- If $\tan(y) = x$, then $y = \tan^{-1}(x) = \text{atan}(x)$; If $\cot(y) = x$, then $y = \cot^{-1}(x) = \text{acot}(x)$
- If $\sec(y) = x$, then $y = \sec^{-1}(x) = \text{asec}(x)$; If $\csc(y) = x$, then $y = \csc^{-1}(x) = \text{acsc}(x)$

Four trigonometric functions (*sin, cos, tan, cot*) and three inverse trigonometric functions (*asin, acos, atan*) are pre-defined in MATLAB. Using trigonometric identities it is possible to figure the remaining trigonometric functions (*sec, csc*) and inverse trigonometric functions (*acot, asec, acsc*). These functions can take real as well as complex arguments. Plots of the pre-defined trigonometric and inverse trigonometric functions in appropriate real domains are shown below.

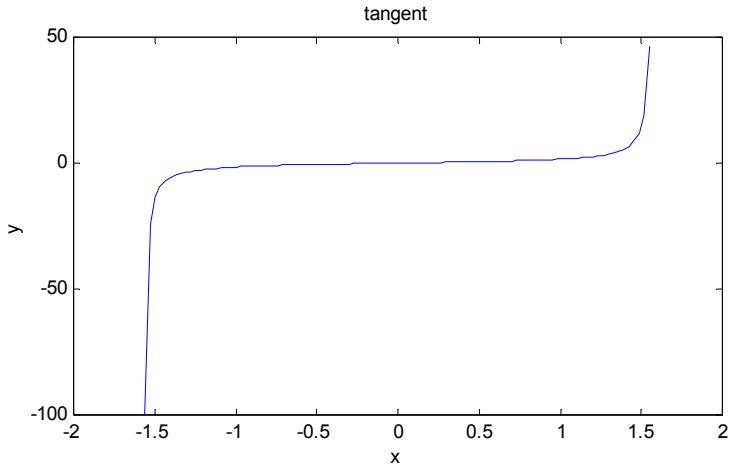
```
» % Sine function
» x = [-2*pi:pi/100:2*pi];
» plot(x,sin(x));title('sine');xlabel('x');ylabel('y');
```



```
» % Cosine function
» x = [-2*pi:pi/100:2*pi];
» plot(x,cos(x));title('cosine');xlabel('x');ylabel('y');
```




```
-->//Tangent function
-->x=[-%pi/2+0.01:0.01:%pi/2-0.01];
-->plot(x,tan(x));xtitle('tangent');xlabel('x');ylabel('y');
```

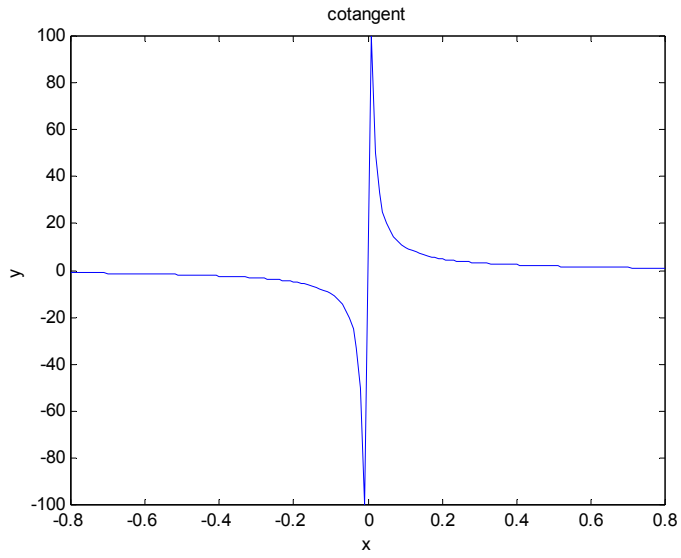


A plot of the cotangent function, using the domain $(-\pi/4, \pi/4)$ for x produces a division by zero:

```
» x = [-pi/2:pi/100:pi/2];plot(x,cot(x))
Warning: Divide by zero.
> In C:\MATLAB_SE_5.3\toolbox\matlab\elfun\cot.m at line 8
```

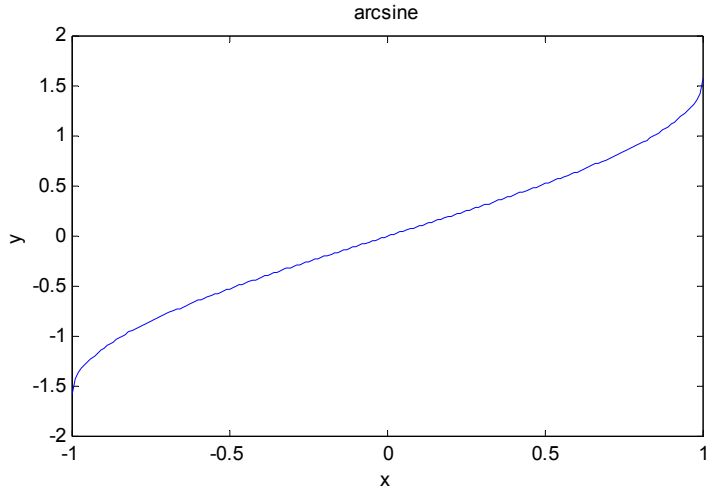
The reason for this result is that, as $x \rightarrow 0$, $cot(x) \rightarrow \pm\infty$, since $tan(0) = 0$, and $cot(x) = 1/tan(x)$. The following MATLAB commands will produce a plot that avoids evaluating the function at $x = 0$:

```
» % Cotangent function
» x1=[-0.8:0.01:-0.01];x2=[0.01:0.01:0.8]; % two symmetric domains about x = 0
» x = [x1, x2]; %join the two domains into a single one
» plot(x,cot(x));title('cotangent');xlabel('x');ylabel('y');
```

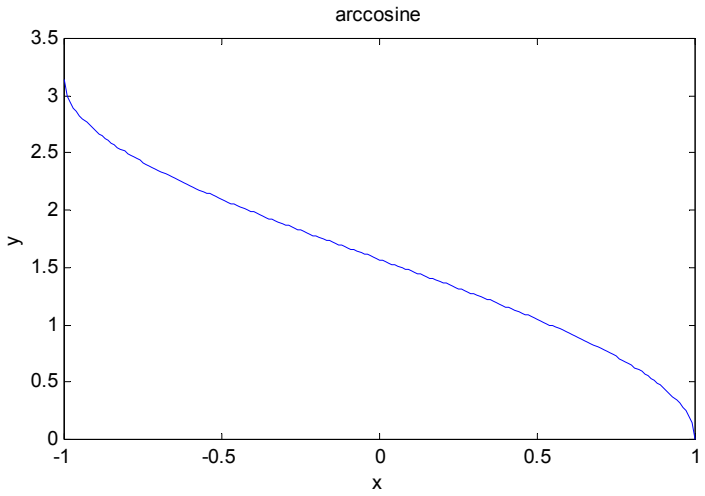


In evaluating inverse trigonometric function for plotting, it is necessary to keep in mind the range of values of the original trigonometric functions. For example, the range of values of *sine* and *cosine* is the interval [*-1,1*]. Thus, both the *asin(x)* and *acos(x)* functions will return real numbers if the argument is in the interval *-1* < *x* < *1*. On the other hand, the range of values for the *tangent* and *cotangent* are (-∞, ∞). Thus, the function *atan(x)* will return a real number for any value of *x*. Plots of the three inverse trigonometric functions are shown next.

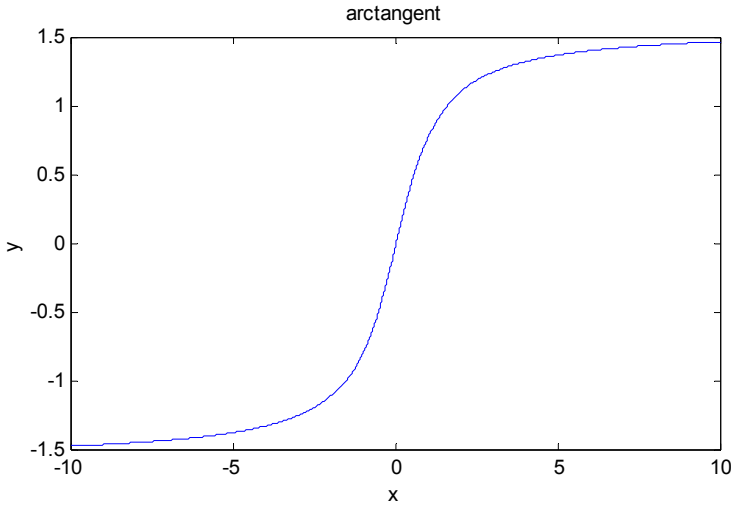
```
-->x = [-1:0.01:1];plot(x,asin(x)); //Arcsine function
-->xtitle('arcsine');xlabel('x');ylabel('y');
```




```
-->x = [-1:0.01:1];plot(x,acos(x)); //Arcosine function
-->xtitle('arccosine');xlabel('x');ylabel('y');
```




```
» x = [-10:0.01:10];plot(x,atan(x)); % Arctangent function
» title('arctangent');xlabel('x');ylabel('y');
```



The four trigonometric functions and three inverse trigonometric functions defined in MATLAB can take as arguments complex numbers returning complex numbers themselves. The way these functions are calculated with complex arguments requires the definition of the *hyperbolic functions* which will are introduced in the following section. Without defining these calculations, we present next some examples of trigonometric and inverse trigonometric functions with complex arguments:

```
» sin(3-5*i)
ans = 10.4725 +73.4606i

» cos(-2-8*i)
ans = -6.2026e+002 -1.3553e+003i

» tan(0.5-0.3*i)
ans = 0.4876 - 0.3689i

» cot(-0.3+0.86*i)
ans = -0.2746 - 1.3143i
```

```
» asin(1-i)
ans = 0.6662 - 1.0613i

» acos(5-3*i)
ans = 0.5470 + 2.4529i

» atan(2-3*i)
ans = 1.4099 - 0.2291i
```

When the argument of functions *asin(x)* and *acos(x)* is outside of the range *-1 < x < 1*, these functions will return complex numbers, e.g.,

```
» asin(10)
ans = 1.5708 + 2.9932i

» acos(20)
ans = 0 - 3.6883i
```

The latter result, in which the real part of the complex number returned is zero, is referred as a *purely imaginary number*. Thus, numbers of the form *yi*, where *i* is the unit imaginary number, are purely imaginary numbers, e.g., *5i, -10i, πi*.

Hyperbolic functions

Hyperbolic functions result from combining exponential functions. The following are the definitions of the *hyperbolic sine* (*sinh*), *hyperbolic cosine* (*cosh*), and *hyperbolic tangent* (*tanh*) functions:

$$\sinh(x) = \frac{e^x - e^{-x}}{2}, \quad \cosh(x) = \frac{e^x + e^{-x}}{2}, \quad \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The remaining hyperbolic functions, *hyperbolic cotangent* (*coth*), *hyperbolic secant* (*sech*), and *hyperbolic cosecant* (*csch*) are defined by

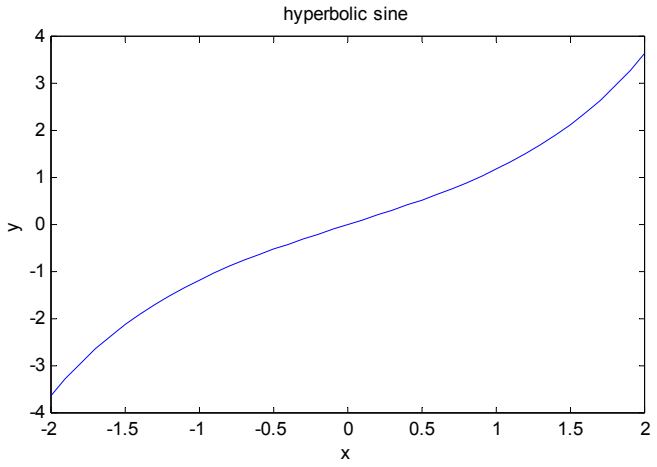
coth(x) = 1/tanh(x), *sech(x) = 1/cosh(x)*, and *csch(x) = 1/sinh(x)*.

The inverse hyperbolic functions are defined as

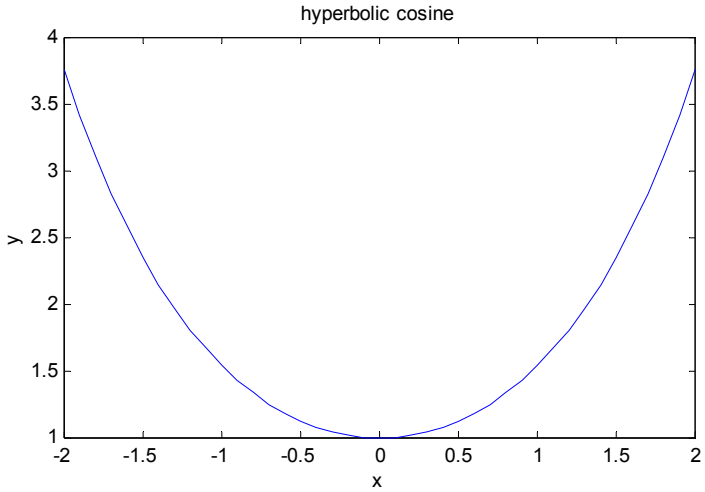
- If *sinh(y) = x*, then *y = sinh^{-1}(x) = asinh(x);* If *cosh(y) = x*, then *y = cosh^{-1}(x) = acosh(x)*
- If *tanh(y) = x*, then *y = tanh^{-1}(x) = atanh(x)*; If *coth(y) = x*, then *y = coth^{-1}(x) = acoth(x)*
- If *sech(y) = x*, then *y = sech^{-1}(x) = asech(x)*; If *csch(y) = x*, then *y = csch^{-1}(x) = acsch(x)*

MATLAB includes the following hyperbolic and inverse hyperbolic functions: *acosh, asinh, atanh, cosh, coth, sinh, tanh*. Plots of these functions for real arguments are shown next:

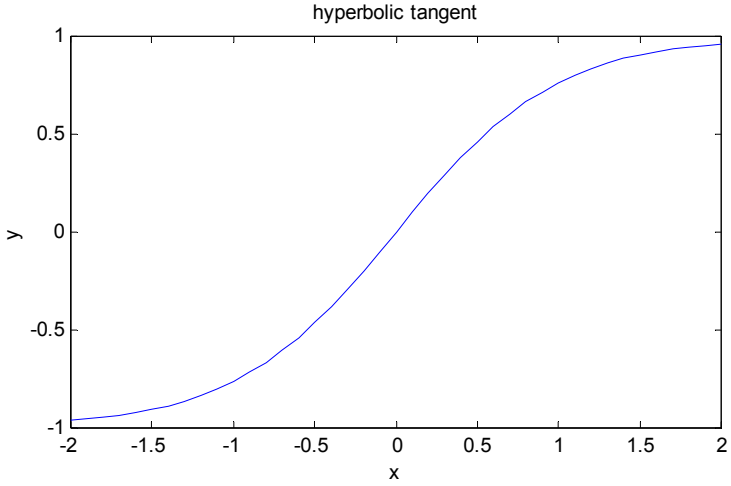
```
» % hyperbolic sine
» x = [-2:0.1:2]; plot(x,sinh(x));
» title('hyperbolic sine');xlabel('x');ylabel('y');
```



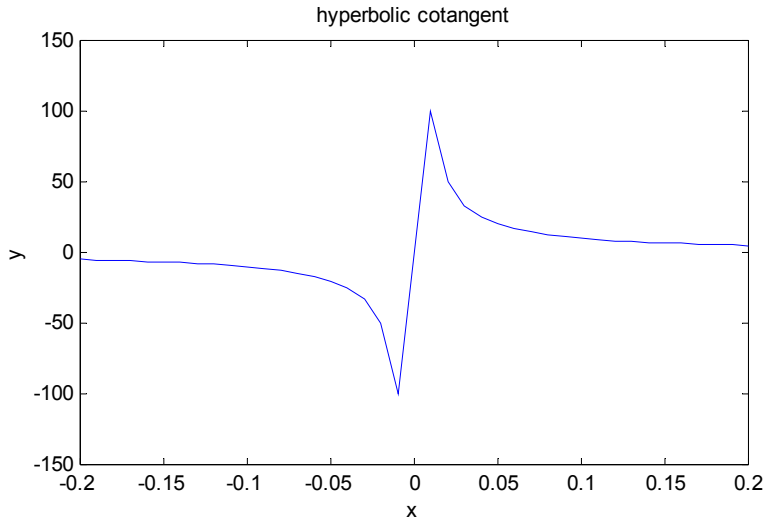

```
» % hyperbolic cosine
» x = [-2:0.1:2]; plot(x,cosh(x));
» title('hyperbolic cosine');xlabel('x');ylabel('y');
```



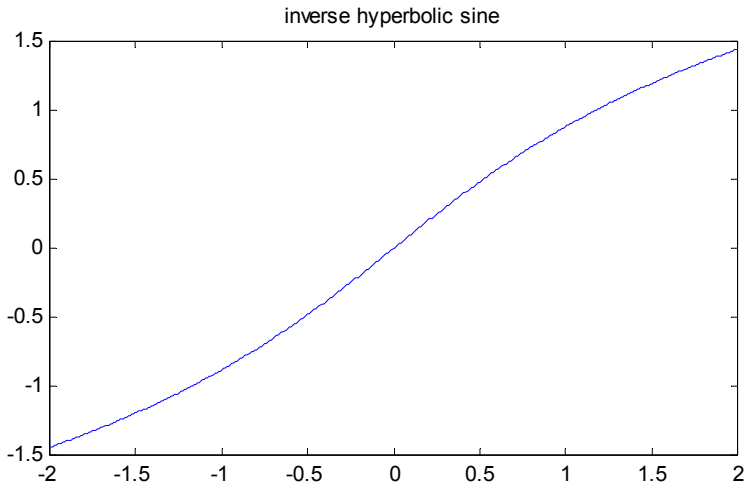

```
» % hyperbolic tangent
» x=[-2:0.1:2];plot(x,tanh(x));
» title('hyperbolic tangent');xlabel('x');ylabel('y');
```



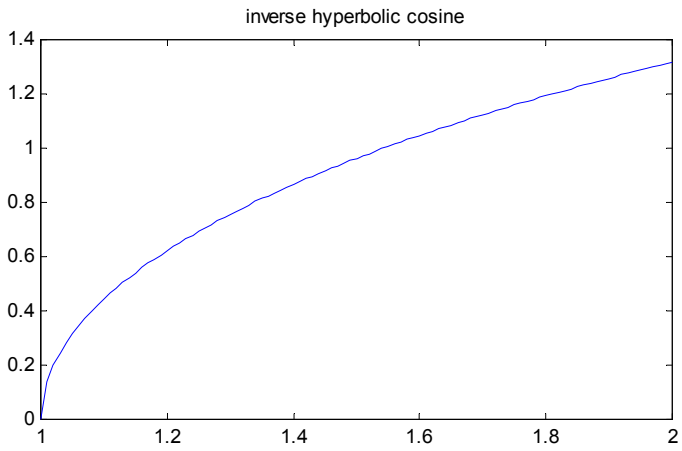
```
» %hyperbolic cotangent
» x1=[-0.2:0.01:-0.01];x2=[0.01:0.01:0.2];x=[x1,x2];
» plot(x,coth(x));title('hyperbolic cotangent');label('x');ylabel('y');
```



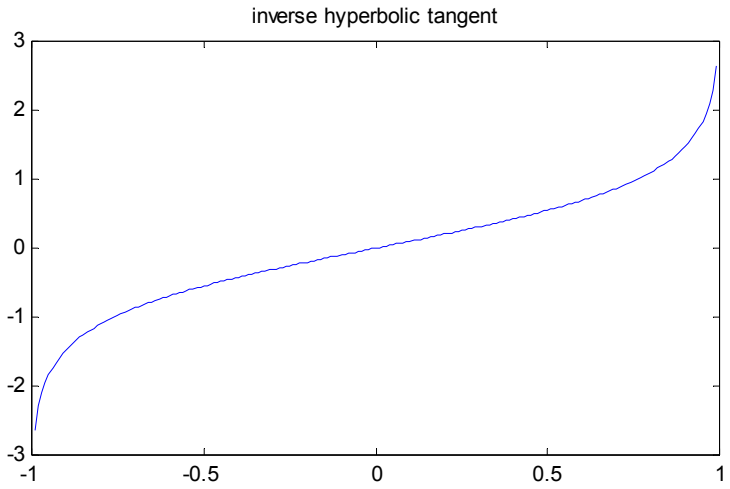

```
» % inverse hyperbolic sine
» x=[-2:0.01:2];plot(x,asinh(x));xtitle('inverse hyperbolic sine');
```




```
-->//inverse hyperbolic cosine
-->x=[1:0.01:2];plot(x,acosh(x));xtitle('inverse hyperbolic cosine');
```




```
-->//inverse hyperbolic tangent
-->x=[-0.99:0.01:0.99];plot(x,atanh(x));xtitle('inverse hyperbolic tangent');
```



Trigonometric and hyperbolic functions, and their inverses, in MATLAB, can use complex numbers as arguments. The rules for evaluation of these functions can be easily obtained by using their definitions in terms of exponential functions. Definitions of the hyperbolic functions in terms of exponential functions were presented above. To define trigonometric functions in terms of exponential functions we use Euler formula as follows:

$$e^{ix} = cos(x) + i\ sin(x)$$

$$e^{-ix} = cos(x) - i\ sin(x)$$

By subtracting and adding these two equations we arrive to the following definitions:

$$\sin(x) = -i\frac{e^{ix} - e^{-ix}}{2}, \quad \cos(x) = \frac{e^{ix} + e^{-ix}}{2}.$$

Also,

$$\tan(x) = \frac{\sin(x)}{\cos(x)} = -i\frac{e^{ix} - e^{-ix}}{e^{ix} - e^{-ix}}.$$

If you now replace the real variable *x* with the complex variable *z* = *x*+*iy* in the definitions for *sin(x)*, *cos(x)*, *tan(x)*, *sinh(x)*, *cosh(x)*, and *tanh(x)*, expressions for *sin(z)*, *cos(z)*, *tan(z)*, *sinh(z)*, *cosh(z)*, and *tanh(z)* can be obtained.

Examples of evaluation of trigonometric and inverse trigonometric functions for complex arguments were presented in the previous section. Following, we present the evaluation of hyperbolic and inverse hyperbolic functions with complex arguments:

```
» sinh(2-5*i)
ans = 1.0288 + 3.6077i

» cosh(-2+5*i)
ans = 1.0672 + 3.4779i

» tanh(1-6*i)
ans = 0.7874 + 0.1165i

» coth(3-2*i)
ans = 0.9968 - 0.0037i

» asinh(2.5-0.5*i)
ans =    1.6631 - 0.1840i

» acosh(2-6*i)
ans = 2.5426 - 1.2527i

» atanh(3-i)
ans = 0.3059 - 1.4615i
```

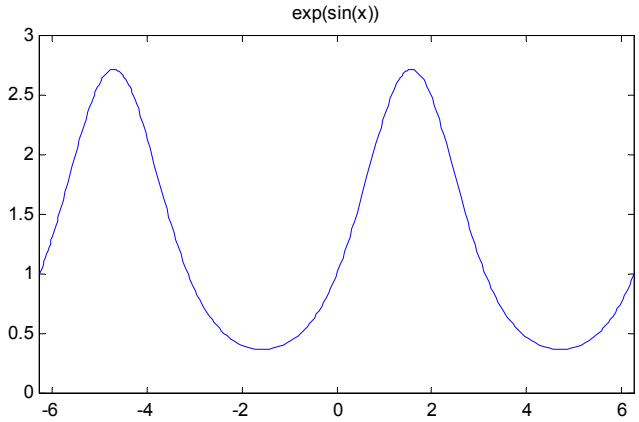
Combining elementary functions

Elementary mathematical functions as those described in this document can be combined in a variety of ways. Some examples are shown next:

- Composite functions

  In this example, we define the function *y* = *exp(sin(x))* and plot it in the range: $-2\pi<x<2\pi$:

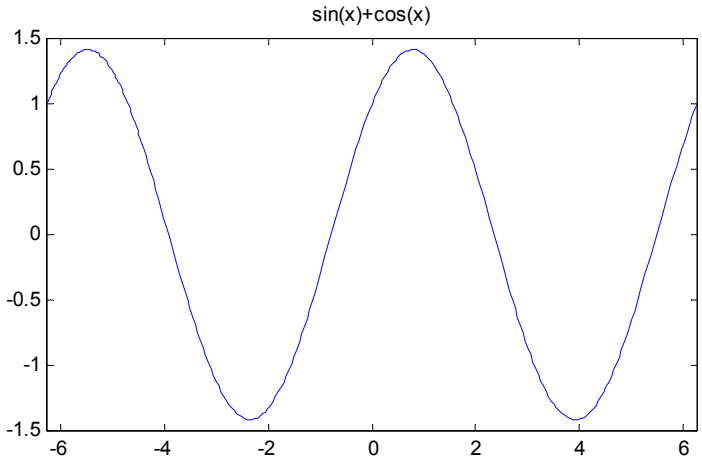
```
» f00 = inline('exp(sin(x))');
» xlim=[-2*pi,2*pi];fplot(f00,xlim);title('exp(sin(x))');
```



- Addition and subtraction

  In this example we define the function $y = sin(x) + cos(x)$, and plot it in the range: $-2\pi<x<2\pi$.

```
» f00 = inline('sin(x)+cos(x)');
» xlim=[-2*pi,2*pi];fplot(f00,xlim);title('sin(x)+cos(x)');
```



- Multiplication and division

  In this example we define the function $y = sin(x)* cos(x)$, and plot it in the range: $-2\pi<x<2\pi$. Just be careful to use a term-by-term multiplication symbol, i.e., .*, to ensure that evaluation of the function as a vector is accomplished properly.

```
» f00=inline('sin(x).*cos(x)');
» xlim=[-2*pi,2*pi];fplot(f00,xlim);title('sin(x)*cos(x)');
```