Chapter 13
Built-in Functions

This chapter lists and describes many of Mathcad's built-in functions. Functions associated with Mathcad's statistical and data analysis features are described in Chapter 14, “Statistical Functions.” Functions used for working with vectors are described in Chapter 10, “Vectors and Matrices.” And for functions to solve differential equations, see Chapter 16, “Solving Differential Equations.”

The following sections make up this chapter:

Inserting built-in functions

Using the **Insert Function** dialog box to see all available functions and get help on what they do.

Transcendental functions

Basic trigonometric, exponential, hyperbolic, and Bessel functions.

Truncation and round-off functions

Functions which extract something from a number, including the real or imaginary part, the mantissa, or the modulo function.

Discrete transform functions

Functions for discrete complex Fourier transforms and wavelet transforms.

Sorting functions

Functions to sort elements of vectors and matrices.

Piecewise continuous functions

Using piecewise continuous functions to perform conditional branching and iteration.

Pro

String functions

Functions for manipulating strings, converting strings to and from numbers and vectors, and creating customized error messages.

Inserting built-in functions

This section describes how to see a list of all functions available to you together with a brief description of each function. Mathcad's set of built-in functions can change depending on whether you've installed additional function packs or whether you've written your own built-in functions. These functions can come from four sources:

Built-in Mathcad functions

This is the core set of functions that come with Mathcad. These functions are all documented here and in other chapters of this *User's Guide*.

Mathcad Function Packs and Electronic Books

A Function Pack consists of a collection of advanced functions geared to a particular area of application. Documentation for these functions comes with the Function Pack itself. In addition, some but not all Electronic Books come with additional functions. Documentation for any of these functions is in the Electronic Book itself. The list of available Function Packs and Electronic Books is constantly expanding and includes collections for image processing, numerical analysis and advanced statistical analysis. To find out more about these products, contact MathSoft or your local distributor.

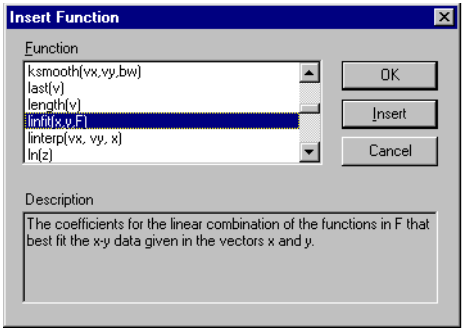
Pro

Built-in functions you write yourself

If you have Mathcad Professional and a supported 32-bit C compiler, you can write your own built-in functions. For details see Appendix C, “Creating a User DLL.”

To see the list of built-in functions available with your copy of Mathcad, choose **Function** from the **Insert** menu. Although built-in function names are not font sensitive, they are case sensitive. You must type the names of built-in functions exactly as shown in the following tables: uppercase, lowercase, or mixed, as indicated. Alternatively, you can use the Insert Function dialog box to insert a function together with placeholders for its arguments. To do so:

- Click in a blank area of your document or on a placeholder.
- Choose **Function** from the **Insert** menu. Mathcad opens the Insert Function dialog box shown on the following page.
- Double-click on the function you want to insert from the left-hand scrolling list, or click the “Insert” button.
- Close the dialog box if you no longer need it by clicking the “Cancel” button.



The scrolling list at the top of the Insert Function dialog box shows all of Mathcad's built-in functions along with their arguments. The box below gives a brief description of the currently selected function.

To apply a function to an expression you have already entered, place the expression between the two editing lines and follow the steps given on the preceding page.

Transcendental functions

This section describes Mathcad's trigonometric, hyperbolic, and exponential functions together with all their inverses. It also describes Mathcad's built-in cylindrical Bessel functions.

Trigonometric functions and their inverses

Mathcad's trig functions and their inverses accept any scalar argument: real, complex, or imaginary. They also return complex numbers wherever appropriate. Complex arguments and results are computed using the following identities:

$$\sin(z) = \frac{e^{i \cdot z} - e^{-i \cdot z}}{2 \cdot i}$$

$$\cos(z) = \frac{e^{i \cdot z} + e^{-i \cdot z}}{2 \cdot i}$$

$$e^{i \cdot z} = \cos(z) + i \cdot \sin(z)$$

If you want to apply one of these functions to every element of a vector or matrix, use the vectorize operator as described in the section “Doing calculations in parallel” in Chapter 10.

Note that all of these trig functions expect their arguments in *radians*. To pass degrees, use the built-in unit *deg*. For example, to evaluate the sine of 45 degrees, type **`sin(45*deg)`**.

Keep in mind that because of round-off errors inherent in a computer, Mathcad may return a very large number where you would ordinarily expect a singularity. In general, you should be cautious whenever you encounter any such singularity.

sin(*z*)	Returns the sine of *z*. In a right triangle, this is the ratio of the length of the side *opposite* the angle over the length of the hypotenuse.
cos(*z*)	Returns the cosine of *z*. In a right triangle, this is the ratio of the length of the side *adjacent* to the angle over the length of the hypotenuse.
tan(*z*)	Returns $\sin(z)/\cos(z)$, the tangent of *z*. In a right triangle, this is the ratio of the length of the side *opposite* the angle over the length of the side *adjacent* to the angle. *z* should not be an odd multiple of $\pi/2$.
csc(*z*)	Returns $1/\sin(z)$, the cosecant of *z*. *z* should not be an even multiple of π.
sec(*z*)	Returns $1/\cos(z)$, the secant of *z*. *z* should not be an odd multiple of $\pi/2$.
cot(*z*)	Returns $1/\tan(z)$, the cotangent of *z*. *z* should not be an even multiple of π.

The inverse trigonometric functions below all return an angle in radians between 0 and $2 \cdot \pi$. To convert this result into degrees, you can either divide the result by the built-in unit *deg* or type *deg* in the units placeholder as described in the section "Displaying units of results" in Chapter 9.

Because of roundoff error inherent in computers, you may find that *atan* of a very large number returns $\pi/2$. As a general rule, it's best to avoid numerical computations near such singularities.

asin(*z*)	Returns the angle (in radians) whose sine is *z*.
acos(*z*)	Returns the angle (in radians) whose cosine is *z*.
atan(*z*)	Returns the angle (in radians) whose tangent is *z*.

Hyperbolic functions

The hyperbolic functions *sinh* and *cosh* are given by:

$$\sinh(z) = \frac{e^z - e^{-z}}{2}$$

$$\cosh(z) = \frac{e^z + e^{-z}}{2}$$

Both these functions will accept and return complex arguments. As the above identities indicate, when you use complex arguments, the hyperbolic functions behave very much like trigonometric functions. In fact:

$$\sinh(i \cdot z) = i \cdot \sin(z)$$
$$\cosh(i \cdot z) = \cos(z)$$

sinh(*z*)	Returns the hyperbolic sine of *z*.
cosh(*z*)	Returns the hyperbolic cosine of *z*.
tanh(*z*)	Returns $\sinh(z) / \cosh(z)$, the hyperbolic tangent of *z*.
csch(*z*)	Returns $1 / \sinh(z)$, the hyperbolic cosecant of *z*.
sech(*z*)	Returns $1 / \cosh(z)$, the hyperbolic secant of *z*.
coth(*z*)	Returns $1(z) / \tanh(z)$, the hyperbolic cotangent of *z*.
asinh(*z*)	Returns the number whose hyperbolic sine is *z*.
acosh(*z*)	Returns the number whose hyperbolic cosine is *z*.
atanh(*z*)	Returns the number whose hyperbolic tangent is *z*.

Log and exponential functions

Mathcad's exponential and logarithmic functions will accept and return complex arguments. Complex arguments to the exponential are given by:

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

In general, a complex argument to the natural log function returns:

$$\ln(x + i \cdot y) = \ln|x + i \cdot y| + \operatorname{atan}(y / x) \cdot i + 2 \cdot n \cdot \pi \cdot i$$

Mathcad's *ln* function returns the value corresponding to $n = 0$. Namely:

$$\ln(x + i \cdot y) = \ln|x + i \cdot y| + \operatorname{atan}(y / x) \cdot i$$

This is called the *principal branch* of the natural log function. Figure 13-1 illustrates some of the basic properties of log functions.

exp(*z*)	Returns *e* raised to the power *z*.
ln(*z*)	Returns the natural log of *z*. ($z \neq 0$).
log(*z*)	Returns the base 10 log of *z*. ($z \neq 0$).

Figure 13-1 shows how you can use these functions to easily find the log to any base.

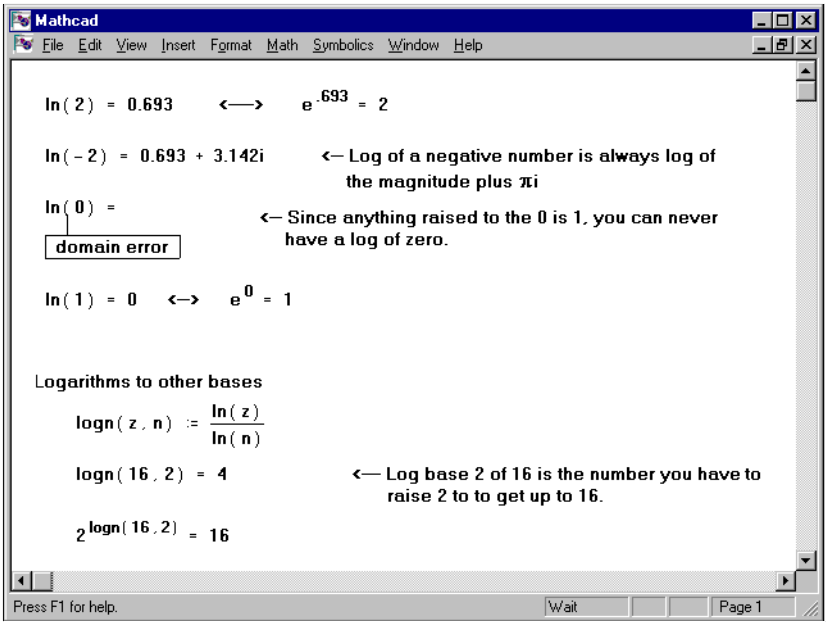


Figure 13-1: Using logarithmic functions.

Bessel functions

These functions typically arise as solutions to the wave equation subject to cylindrical boundary conditions.

Bessel functions of the first kind and second kind, $J_n(x)$ and $Y_n(x)$, are solutions to the following differential equation:

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} + (x^2 - n^2) \cdot y = 0$$

Modified Bessel functions of the first and second kind, $I_n(x)$ and $K_n(x)$, are solutions

to the slightly different differential equation:

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} - ((x^2 - n^2) \cdot y) = 0$$

J0(*x*)	Returns $J_0(x)$. *x* real.
J1(*x*)	Returns $J_1(x)$. *x* real.
$J_n(m, x)$	Returns $J_m(x)$. *x* real, $0 \le m \le 100$.
Y0(*x*)	Returns $Y_0(x)$. *x* real, $x > 0$.
Y1(*x*)	Returns $Y_1(x)$. *x* real, $x > 0$.
$Y_n(m, x)$	Returns $Y_m(x)$. $x > 0$, $0 \le m \le 100$.
I0(*x*)	Returns $I_0(x)$. *x* real.
I1(*x*)	Returns $I_1(x)$. *x* real.
$I_n(m, x)$	Returns $I_m(x)$. *x* real, $0 \le m \le 100$.
K0(*x*)	Returns $K_0(x)$. *x* real, $x > 0$.
K1(*x*)	Returns $K_1(x)$. *x* real, $x > 0$.
$K_n(m, x)$	Returns $K_m(x)$. $x > 0$, $0 \le m \le 100$.

Special functions

The following functions arise in a wide variety of problems.

erf(*x*) Returns the value of the error function at *x*:

$$\mathrm{erf}(x) = \int_0^x \frac{2}{\sqrt{\pi}} e^{-t^2} dt$$

x must be real.

$\Gamma(z)$ Returns the value of the Euler gamma function at z. For real z, the values of this function coincide with the following integral:

$$\Gamma(x) = \int_0^\infty t^{z-1} e^{-t} dt$$

For complex z, the values are the analytic continuation of the real function. Euler's gamma function is undefined for $z = 0, -1, -2, \ldots$

Euler's gamma function satisfies the recurrence relationship:

$$\Gamma(z+1) = z\Gamma(z)$$

which means that when z is a positive integer:

$$\Gamma(z+1) = z!$$

The error function arises frequently in statistics. You can also use it to define the complementary error function as:

$$\mathrm{erfc}(x) := 1 - \mathrm{erf}(x)$$

Truncation and round-off functions

These functions all have in common the fact that they extract something from their arguments.

The functions *Re*, *Im*, and *arg* extract the corresponding part of a complex number. For more information on these functions, see Chapter 8, "Variables and Constants."

The functions *ceil* and *floor* will return the next integer above and below their arguments respectively. You can use these functions to create a function that returns just the mantissa of a number:

```
mantissa(x) := x - floor(x)
```

Figure 13-2 shows how you can use the *floor* and *ceil* functions to round off numbers.

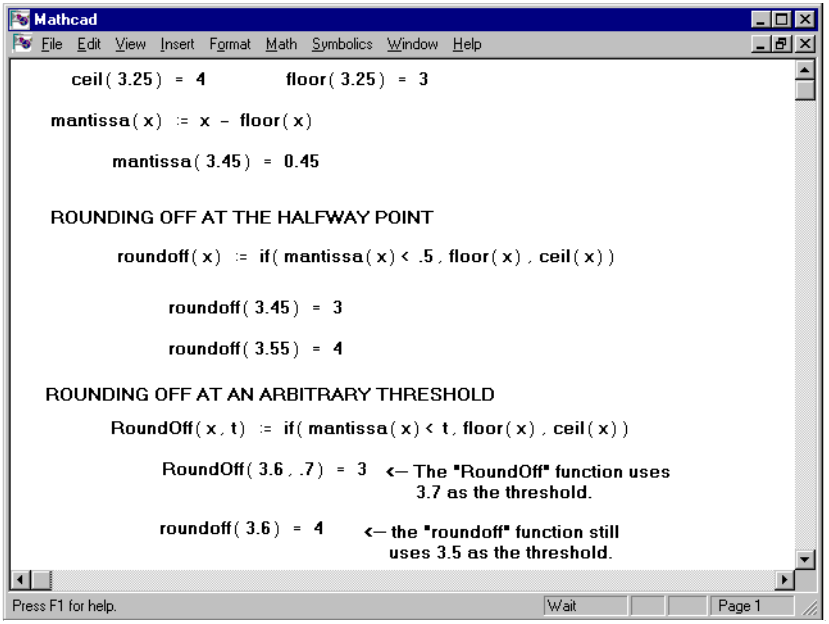


Figure 13-2: Creating a round-off function.

Re(z)	Real part of z.
Im(z)	Imaginary part of z.
arg(z)	Argument of z: the value of θ when z is written as $r \cdot e^{i \cdot \theta}$. Result is between $-\pi$ and π.
floor(x)	Greatest integer $\leq x$ (x real).
ceil(x)	Least integer $\geq x$ (x real).
mod(x, y)	Remainder on dividing x by y. Result has same sign as x.
angle(x, y)	Angle (in radians) from positive x-axis to the point (x, y) in the x-y plane. Arguments must be real. Returns a value between zero and 2π.

Discrete transform functions

Mathcad contains a variety of functions for performing discrete transforms. All these functions require vectors as arguments. When you define a vector **v** for use with Fourier or wavelet transforms, be sure to start with v_0. If you do not define v_0, Mathcad automatically sets it to zero. This can distort the results of the transform functions.

Introduction to Discrete Fourier transforms

Mathcad comes with two types of Fourier transform pairs: $fft/ifft$ and $cfft/icfft$. These functions are discrete: they apply to and return vectors and matrices only. You cannot use them with other functions.

Use the *fft* and *ifft* functions if:

- The data values in the time domain are real, and
- the data vector has 2^m elements.

Use the *cfft* and *icfft* functions in all other cases.

The first condition is required because the *fft/ifft* pair takes advantage of the fact that, for real data, the second half of the transform is just the conjugate of the first. Mathcad discards the second half of the result vector. This saves both time and memory. The *cfft/icfft* pair does not assume symmetry in the transform. For this reason, you *must* use this pair for complex valued data. Since the real numbers are just a subset of the complex numbers, you can use the *cfft/icfft* pair for real numbers as well.

The second condition is required because the *fft/ifft* Fourier transform pair uses a highly efficient fast Fourier transform algorithm. In order to do so, the vector you use with *fft* must have 2^m elements. The *cfft/icfft* Fourier transform pair uses an algorithm that permits vectors as well as matrices of arbitrary size. When you use this transform pair with a matrix, you get back a two-dimensional Fourier transform.

Note that if you used *fft* to get to the frequency domain, you *must* use *ifft* to get back to the time domain. Similarly, if you used *cfft* to get to the frequency domain, you *must* use *icfft* to get back to the time domain.

Different sources use different conventions concerning the initial factor of the Fourier transform and whether to conjugate the results of either the transform or the inverse transform. The functions *fft*, *ifft*, *cfft*, and *icfft* use $1/\sqrt{N}$ as a normalizing factor and a positive exponent in going from the time to the frequency domain. The functions *FFT*, *IFFT*, *CFFT*, and *ICFFT* use $1/N$ as a normalizing factor and a negative exponent in going from the time to the frequency domain. Be sure to use these functions in pairs. For example, if you used *CFFT* to go from the time domain to the frequency domain, you *must* use *ICFFT* to transform back to the time domain.

Fourier transforms on real data

With 2^m real-valued data points, you can use the *fft/ifft* Fourier transform pair. These functions take advantage of symmetry conditions present only when the data is real. This saves both time and memory.

---

fft(**v**) This function returns the Fourier transform of a 2^m element vector of real data representing measurements at regular intervals in the time domain.

The vector **v** must have 2^m elements. The result is a vector of $1 + 2^{m-1}$ complex coefficients representing values in the frequency domain. If **v** contains other than 2^m elements, Mathcad returns an error message.

The elements of the vector returned by *fft* satisfy the following equation:

$$c_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} v_k e^{2\pi i(j/n)k}$$

In this formula, *n* is the number of elements in **v** and *i* is the imaginary unit.

The elements in the vector returned by the *fft* function correspond to different frequencies. To recover the actual frequency, you must know the sampling frequency of the original signal. If **v** is an *n* element vector passed to the *fft* function, and the sampling frequency is f_s, the frequency corresponding to c_k is

$$f_k = \frac{k}{n} \cdot f_s$$

Note that this makes it impossible to detect frequencies above the sampling frequency. This is a limitation not of Mathcad, but of the underlying mathematics itself. In order to correctly recover a signal from the Fourier transform of its samples, you must sample the signal with a frequency of at least twice its bandwidth. A thorough discussion of this phenomenon is outside the scope of this manual but within that of any textbook on digital signal processing.

---

ifft(**v**) This function returns the inverse Fourier transform of a vector of data representing values in the frequency domain. The inverse transform will be pure real.

The vector **v** must have $1 + 2^m$ elements for *m* integer. The result is a vector of 2^{m+1} complex coefficients representing values in the frequency domain. If **v** contains other than $1 + 2^m$ elements, Mathcad returns an error message.

The argument **v** is a vector similar to those generated by the *fft* function. To compute the result, Mathcad first creates a new vector **w** by taking the conjugates of the elements of **v** and appending them to the vector **v**. Then Mathcad computes a vector **d** whose elements satisfy this formula:

$$d_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} w_k e^{-2\pi i(j/n)k}$$

This is the same formula as the *fft* formula, except for the minus sign in the *exp* function. The *fft* and *ifft* functions are exact inverses. For all real **v**, $\mathrm{ifft}(\mathrm{fft}(\mathbf{v})) = \mathbf{v}$.

Fourier transforms on complex data

There are two reasons why you may not be able to use the *fft/ifft* transform pair discussed in the previous section:

- The data may be complex valued. This means that Mathcad can no longer exploit the symmetry present in the real valued case.
- The data vector might not have 2^m data points in it. This means Mathcad cannot take advantage of the highly efficient FFT algorithm used by the *fft/ifft* Fourier transform pair.

Complex Fourier transforms require the following functions:

cfft(**A**)	Returns the fast Fourier transform of a vector or matrix of complex data representing equally spaced measurements in the time domain. The array returned is the same size as the array you used as an argument.
icfft(**A**)	Returns the inverse Fourier transform of a vector or matrix of data representing values in the frequency domain. The result is an array representing values in the time domain. The *icfft* function is the inverse of the *cfft* function. Like *cfft*, this function returns an array of the same size as its argument.

Although the *cfft/icfft* Fourier transform pair will work on arrays of any size, they work significantly faster when the number of rows and columns contains many smaller factors. Vectors with length 2^m fall into this category. So do vectors having lengths like 100 or 120. On the other hand, a vector whose length is a large prime number will slow down the Fourier transform algorithm.

The *cfft* and *icfft* functions are exact inverses. That is, $\mathrm{icfft}(\mathrm{cfft}(\mathbf{v})) = \mathbf{v}$. Figure 13-3 shows an example of Fourier transforms in Mathcad.

When you use the *cfft* with a matrix, the result is the two-dimensional Fourier transform of the input matrix

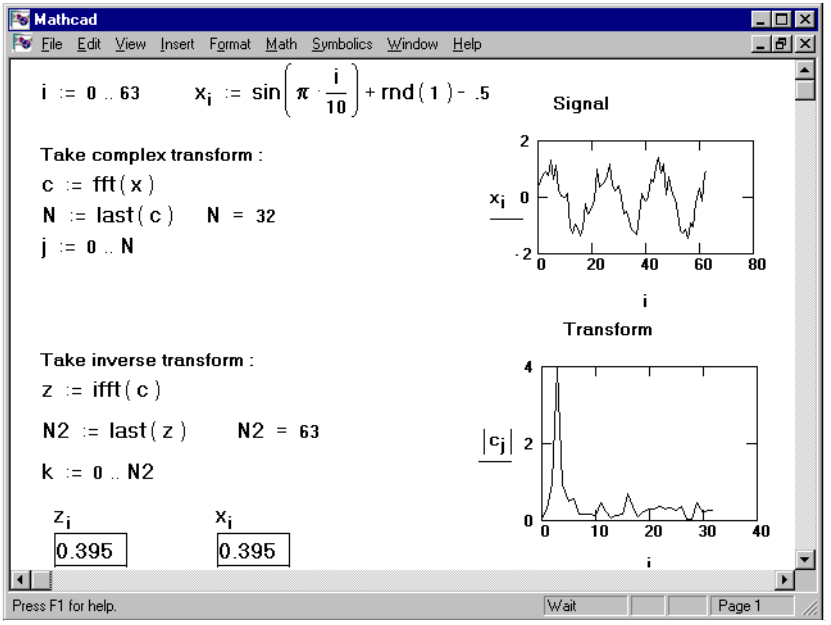


Figure 13-3: Use of fast Fourier transforms in Mathcad.

Alternate forms of the Fourier transform

The definitions for the Fourier transform discussed earlier are not the only ones used. For example, the following definitions for the discrete Fourier transform and its inverse appear in Ronald Bracewell's *The Fourier Transform and Its Applications* (McGraw-Hill, 1986):

$$F(\upsilon) = \frac{1}{n}\sum_{\tau=1}^{n} f(\tau)e^{-2\pi i(\upsilon/n)\tau}$$

$$f(\tau) = \sum_{\upsilon=1}^{n} F(\upsilon)e^{-2\pi i(\tau/n)\upsilon}$$

These definitions are very common in the engineering literature. To use these definitions rather than those presented in the last section, use the functions *FFT*, *IFFT*, *CFFT*, and *ICFFT*. These differ from those discussed in the last section as follows:

- Instead of a factor of $1/\sqrt{n}$ in front of both forms, there is a factor of $1/n$ in front of the transform and no factor in front of the inverse.
- The minus sign appears in the exponent of the transform instead of in its inverse.

The functions *FFT*, *IFFT*, *CFFT*, and *ICFFT* are used in exactly the same way as the functions discussed in the previous section.

Wavelet transforms

Mathcad Professional includes two wavelet transforms for performing the one-dimensional discrete wavelet transform and its inverse. The transform is performed using the Daubechies four-coefficient wavelet basis.

Pro	wave(v)	Returns the discrete wavelet transform of **v**, a 2^m element vector containing real data. The vector returned is the same size as **v**.
Pro	iwave(**v**)	Returns the inverse discrete wavelet transform of **v**, a 2^m element vector containing real data. The vector returned is the same size as **v**.

Sorting functions

Mathcad includes three functions shown in Figure 13-4 for sorting arrays and one for reversing the order of their elements:

sort(**v**)	Returns the elements of the vector **v** sorted in ascending order.
csort(**A**, *n*)	Sorts the rows of the matrix **A** so as to place the elements in column *n* in ascending order. The result has the same size as **A**.
rsort(**A**, *n*)	Sorts the columns of the matrix **A** so as to place the elements in row *n* in ascending order. The result has the same size as **A**.
reverse(**v**) reverse(**A**)	Reverses the order of the elements of the vector **v** or the rows of the matrix **A**.

The above sorting functions accept matrices and vectors with complex elements. However in sorting them, Mathcad ignores the imaginary part.

To sort a vector or matrix in descending order, first sort in ascending order, then use *reverse*. For example, *reverse*(*sort*(**v**)) returns the elements of **v** sorted in descending order.

Unless you change the value of ORIGIN, matrices are numbered starting with row zero and column zero. If you forget this, it's easy to make the error of sorting a matrix on the wrong row or column by specifying an incorrect *n* argument for *rsort* and *csort*. To sort on the first column of a matrix, for example, you must use *csort*(**A**, 0).

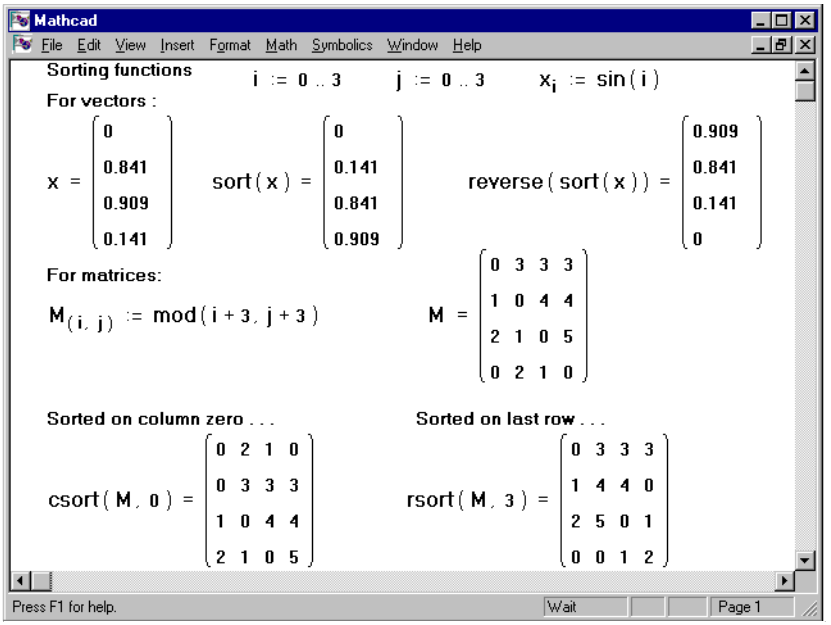


Figure 13-4: Sorting functions.

Piecewise continuous functions

Piecewise continuous functions are useful for branching and iteration. There are five Mathcad functions falling into this category. The *if* function is useful for choosing one of two values based on a condition. The Heaviside step function, $\Phi(x)$, and the Kronecker Delta function, $\delta(m, n)$, are special cases of the *if* function.

The *until* function is used to drive iteration. It is unique among Mathcad functions because it is designed to work only with range variables. This is the only Mathcad function which can actually halt iteration upon the occurrence of a condition. Mathcad Professional, however, also includes specialized programming operators that allow you to control iteration, as described in Chapter 18, “Programming.”

The last function is ε, the completely anti-symmetric tensor function. This returns a 0, 1, or -1 depending on the permutation of its arguments. Although this function is of limited applicability, it would be difficult to perform this function using any other combination of Mathcad functions.

The if function

Use *if* to define a function that behaves one way below a certain number and behaves completely differently above that number. That point of discontinuity is specified by its first argument, *cond*. The remaining two arguments let you specify the behavior of the function on either side of that discontinuity.

if(*cond, tval, fval*)	Returns *tval* if *cond* is nonzero (true) Returns *fval* if *cond* is zero (false).

Although the argument *cond* can be any expression at all, it is usually more convenient to use a boolean expression from the table below. The following table shows the meaning of boolean expressions with numbers (*x* and *y* can be real scalars, while *w* and *z* can be complex scalars). As explained in Chapter 12, "Operators," boolean operators can also be used to order string expressions character by character based on ASCII codes.

Condition	How to type	Description
$w=z$	**[Ctrl]=**	Boolean equals. Returns 1 if expressions are equal; otherwise 0.
$x>y$	**>**	Greater than.
$x<y$	**<**	Less than.
$x\geq y$	**[Ctrl]0**	Greater than or equal to.
$x\leq y$	**[Ctrl]9**	Less than or equal to.
$w\neq z$	**[Ctrl]3**	Not equal to.

Note that boolean expressions involving inequalities cannot be used with complex numbers. This is because it is meaningless to speak of one complex number being "larger" or "smaller" than another.

To save time, Mathcad only evaluates those arguments it has to. For example, if *cond* is false, there is no need to evaluate *tval* since it will not be returned anyway. Because of this, errors in the unevaluated argument can escape detection. For example, Mathcad will never detect the fact that *ln*(0) is undefined in the expression below:

```
if(|x| < 0,ln(0),ln(x))
```

Figure 13-5 shows several equations using the *if* function. You can combine boolean operators to create more complicated conditions. For example, the condition

$$(x < 1) \cdot (x > 0)$$

acts like an "and" gate, returning 1 only if x is between 0 and 1. Similarly, the expression

$$(x > 1) + (x < 0)$$

acts like an "or" gate, returning a 1 if either $x > 1$ or $x < 0$, but not if x is between 0 and 1.

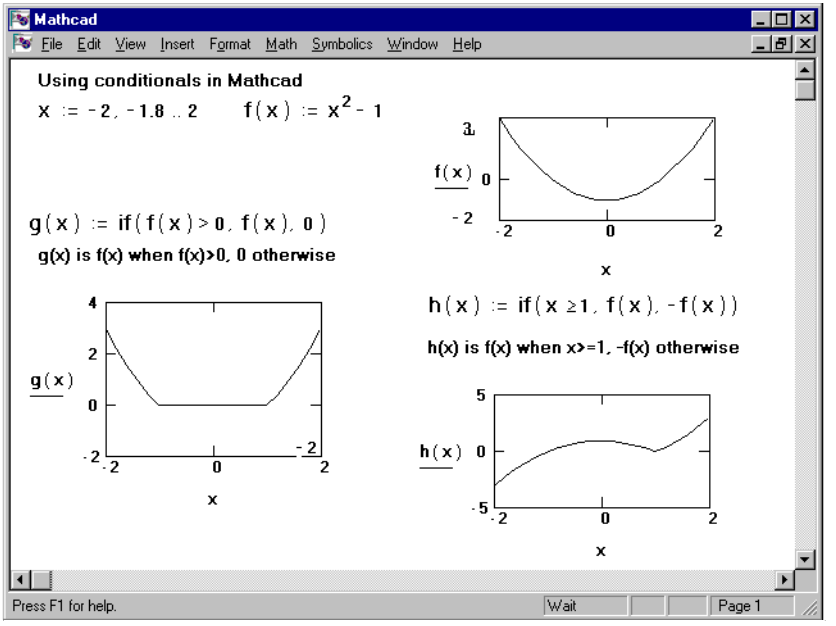


Figure 13-5: Conditionals in Mathcad.

The until function

Mathcad's *until* function allows you to halt an iteration when a particular condition is met. The *until* function has no effect when its first argument involves no range variables. When the first argument does involve a range variable, Mathcad will iterate until the first argument evaluates to a negative value. When this happens, Mathcad halts iteration.

until(x, z)	Returns z until the test expression x becomes negative. x should be an expression involving a range variable.

Do not use the *until* function in equations with more than one range variable (for example, multiple summations). Mathcad will halt all iteration on all range variables the first time that the first argument of *until* is negative. This usually does not produce the desired result.

The *until* function is useful in iterative processes with a specified convergence condition. For example, Figure 13-6 shows how to use the *until* function to test an iterative process for convergence. The iteration in the equation for x_i continues until x_i is within *err* of *a*. Figure 13-6 also shows how to use the *last* function to detect when iteration has halted and to compute the size of the resulting array.

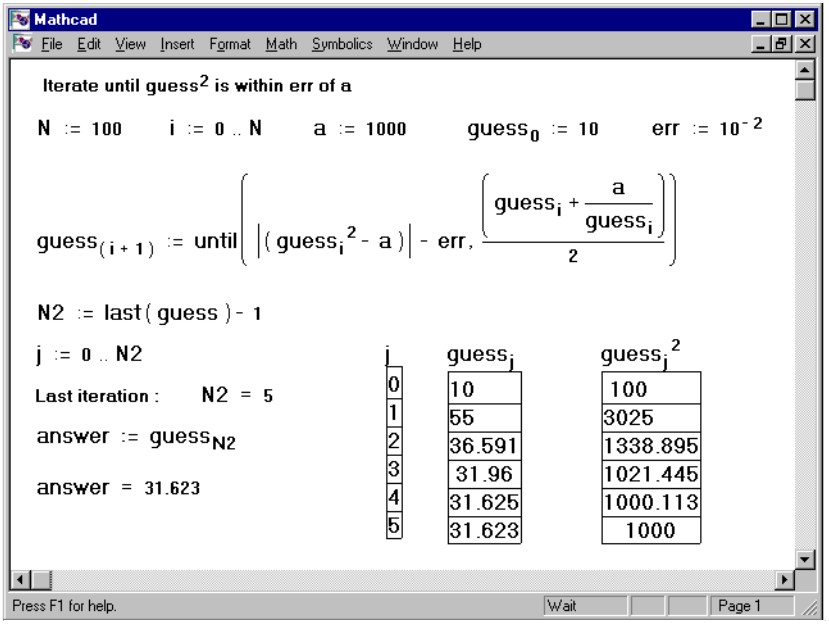


Figure 13-6: Using the until function to halt an iteration.

When you use the *until* function, be sure that the value of the test expression does in fact change somewhere in the iteration. Otherwise, you may find yourself in an infinite loop. If this does occur, press the [**Esc**] key to interrupt calculation.

Impulse and step functions

These two functions are special cases of the *if* function. The Heaviside step function is equivalent to:

$$\Phi(x) := \text{if}(x < 0, 0, 1)$$

For integer *m* and *n*, the Kronecker delta function is equivalent to

$$\delta(m,n) := \text{if}(m = n, 1, 0)$$

$\Phi(x)$	Heaviside step function. 1 if $x \geq 0$; otherwise, 0.
$\delta(m, n)$	Kronecker's delta function. Returns 1 if $m = n$; otherwise, 0. Both arguments must be integer.

You can use the Heaviside step function to define a pulse of width w by defining:

```
pulse(x,w) := Φ(x) - Φ(x-w)
```

A lowpass and highpass filter having width $2 \cdot w$ could then be defined as:

```
lowpass(x,w) := pulse(x+w, 2*w)

highpass(x,w) := 1 - pulse(x+w, 2*w)
```

Figure 13-7 illustrates the use of the Heaviside step function for creating filters.

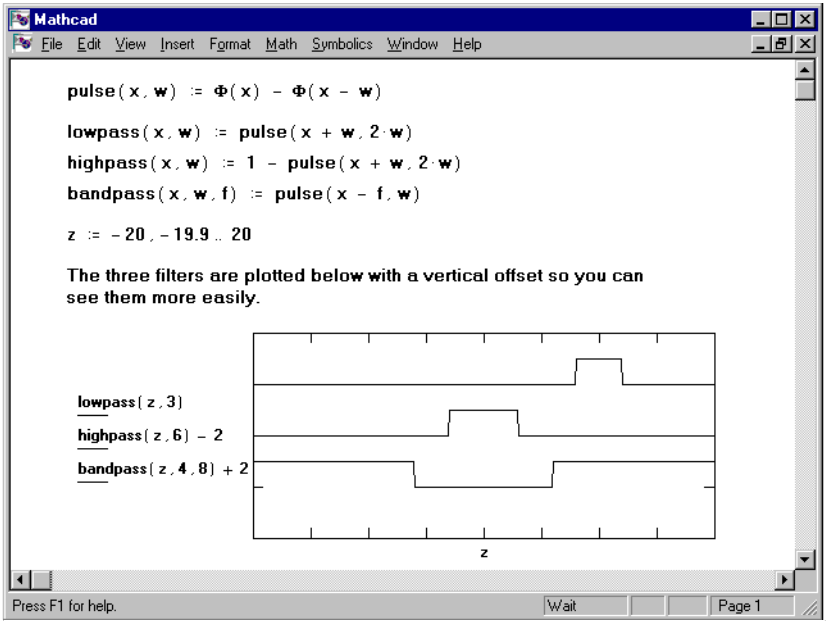


Figure 13-7: Using the step function for filtering.

Antisymmetric tensor function

The arguments to this function are three integers between 0 and 2 inclusive. It basically determines how many times you have to swap two numbers in order to get back to the sequence $[0, 1, 2]$ from whatever sequence $[i, j, k]$ you passed to it.

More generally, $\varepsilon(i, j, k) = 1$ if $[i, j, k]$ is an even permutation of $[0, 1, 2]$ (an even number of swaps), and $\varepsilon(i, j, k) = -1$ if $[i, j, k]$ is an odd permutation of $[0, 1, 2]$ (an odd number of swaps). This explains why $\varepsilon(0, 1, 2) = 1$.

For example, $\varepsilon(2, 0, 1) = 1$ because to get from $[2, 0, 1]$ back to $[0, 1, 2]$ you'll have to swap twice. On the other hand, $\varepsilon(0, 2, 1) = -1$ because to get from $[0, 2, 1]$ back to $[0, 1, 2]$ you only have to swap once. If two numbers are the same, for example $\varepsilon(0, 1, 1)$, you can never get back to $[0, 1, 2]$, so the function just returns 0.

Although this function is not used very often, it is truly indispensable when you need it. It is very difficult to perform this same feat using any other combination of Mathcad functions.

$\varepsilon(i, j, k)$	Completely antisymmetric tensor of rank 3. *i*, *j*, and *k* must be integers between 0 and 2 inclusive (or between ORIGIN and ORIGIN + 2 inclusive if ORIGIN≠0). Result is 0 if any two are the same, 1 for even permutations, −1 for odd permutations.

String functions

Pro

Most built-in functions and operators do not make sense when used with strings and will issue error messages if you use string arguments with them. Mathcad Professional, however, includes several specialized functions for working with strings.

The strings used and returned by most of the functions described below are typed in a math placeholder by pressing the double-quote key (**"**) in a placeholder and entering any combination of letters, numbers, or other ASCII characters. Mathcad automatically places double quotes around the string expression, and will display quotes around a string whenever one is returned as a result.

See Chapter 8, “Variables and Constants,” for more information on strings.

Manipulating strings

The following four functions are useful for combining strings, determining the length of a string, and locating and returning substrings.

When evaluating the functions *search* and *substr*, Mathcad assumes that the first character in a string is at position 0.

Pro	concat(*S1*, *S2*)	Returns a string formed by appending string *S2* to the end of string *S1*.

Pro	strlen(*S*)	Returns the number of characters in string *S*.
Pro	search(*S1*, *SubS*, *m*)	Returns the starting position of the substring *SubS* in string *S1* beginning from position *m*, or –1 if no substring is found. The argument *m* must be a nonnegative integer.
Pro	substr(*S*, *m*, *n*)	Returns a substring of *S* beginning with the character in the *m*th position and having at most *n* characters. The arguments *m* and *n* must be nonnegative integers.

Converting to and from strings

Use *num2str* and *str2num* to convert between numeric values and strings. These functions are useful when it is more convenient to manipulate a number as a string rather than mathematically, or when you need to perform numerical calculations on a string of numbers.

The *str2num* function requires a string consisting of characters which constitute an integer, a floating-point or complex number, or an e-format number such as 4.51e-3 (for $4.51 \cdot 10^{-3}$). For example, valid string arguments for *str2num* are "-16.5," "2.1+6i," "3.241 - 9.234j," and "2.418e3." Spaces are ignored.

Use *str2vec* and *vec2str* if you want to convert between the ASCII codes for characters and the characters themselves. For a list of ASCII codes see Appendix A, "Reference." For example, the ASCII code for letter "a" is 97, that for letter "b" is 98, and that for letter "c" is 99.

Pro	str2num(*S*)	Returns a constant formed by converting the characters in string *S* to a number. *S* must contain only characters which constitute an integer, a floating-point or complex number, or an e-format number such as 4.51e-3 (for $4.51 \cdot 10^{-3}$). Spaces are ignored.
Pro	num2str(*z*)	Returns a string formed by converting the real or complex number *z* into a decimal-valued string.
Pro	str2vec(*S*)	Returns a vector of ASCII codes corresponding to the characters in string *S*.
Pro	vec2str(**v**)	Returns a string formed by converting a vector **v** of ASCII codes to characters. The elements of **v** must be integers between 0 and 255.

Customized error messages

Just as Mathcad's built-in error messages appear as "error tips" when a built-in function is used incorrectly or could not return a result, you may want specialized error messages to appear when your user-defined functions are used improperly or cannot return answers.

Mathcad allows you to define your own error messages using the string function *error*. This function is especially useful for trapping erroneous inputs to Mathcad programs you write. See Chapter 18, "Programming," for details on using *error* in programs.

Pro error(*S*) Returns the string *S* as an error message.

When Mathcad encounters the *error* function in an expression, it highlights the expression in red. When you click on the expression, the error message appears in a tool tip that hovers over the expression. The text of the message is the string argument you supply to the *error* function.

Figure 13-8 shows examples of Mathcad's string manipulation, string conversion, and custom error message functions.

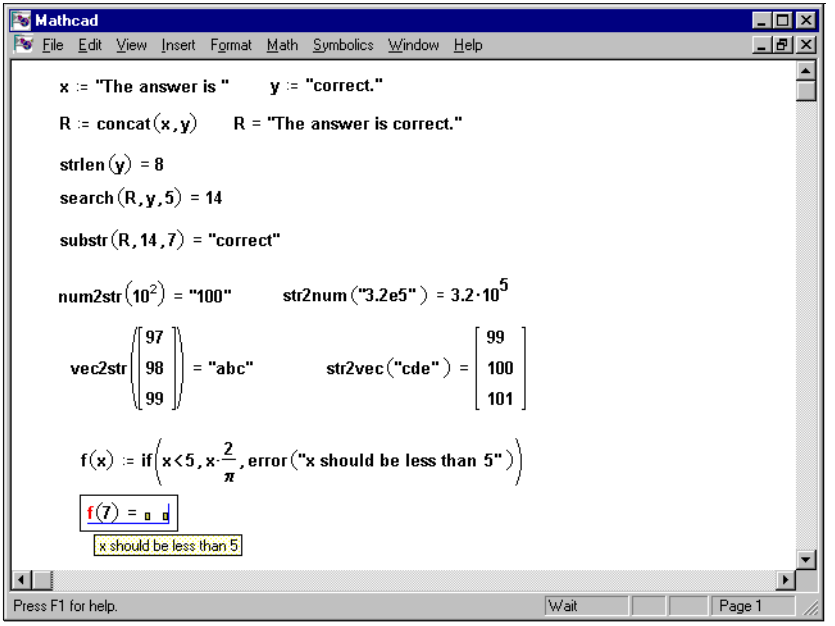


Figure 13-8: Using string functions to manipulate strings, convert to and from strings, and define error messages.