

# Chapter 1

## Functions

This chapter lists and describes Mathcad's built-in mathematical and statistical functions. The functions are listed alphabetically.

Functions labeled *Professional* are available only in Mathcad Professional. Certain features labeled *Expert* require Mathcad Professional and are available for sale separately (in Mathcad Expert Solver).

Function names are case-sensitive, but not font-sensitive. Type them in any font, but use the same capitalization as shown in the syntax section.

Many functions described here as accepting scalar arguments will, in fact, accept vector arguments. For example, while the input  $z$  for the `ACOS` function is specified as a "real or complex number," `acos` will in fact evaluate correctly at each of a vector input of real or complex numbers.

Some functions don't accept input arguments with units. For such a function  $f$ , an error message "must be dimensionless" will arise when evaluating  $f(x)$ , if  $x$  has units.

### Function Categories

Each function falls within one of the following categories:

- Bessel
- Complex numbers
- Differential equation solving
- Expression type
- File access
- Fourier transform
- Hyperbolic
- Interpolation and prediction
- Log and exponential
- Number theory/combinatorics
- Piecewise continuous
- Probability density
- Probability distribution
- Random number
- Regression and smoothing
- Solving
- Sorting
- Special
- Statistics

- String
- Trigonometric
- Truncation and round-off
- Vector and matrix
- Wavelet transform

The category name is indicated in the upper right corner of each entry. To see all the functions that belong to a given category, check the index of this book.

## Finding More Information

You can also find information about functions using either of these methods:

- To quickly see a short description of each function from within Mathcad, choose **Function** from the **Insert** menu. Select a function in the Function field, then read the description in the Description field. Click on the **Help** button to see the Help topic on a selected function.
- Refer to the Resource Center QuickSheets for more detailed information about functions, categories, and related topics. Select **Resource Center** from the **Help** menu. Then click on the QuickSheets icon and select a specific topic.

## About the References

References are provided in Appendix B for you to learn more about the numerical algorithm underlying a given Mathcad function or operator. References are not intended to give a description of the actual underlying source code. Some references (such as *Numerical Recipes*) do contain actual C code for the algorithms discussed therein, but the use of the reference does not necessarily imply that the code is what is implemented in Mathcad. The references are cited for background information only.

# Functions

---

## acos Trigonometric

Syntax	$\text{acos}(z)$
Description	Returns the inverse cosine of $z$ (in radians). The result is between $0$ and $\pi$ if $z$ is real. For complex $z$ , the result is the principal value.
Arguments	
$z$	real or complex number

---

## acosh Hyperbolic

Syntax	$\text{acosh}(z)$
Description	Returns the inverse hyperbolic cosine of $z$ . The result is the principal value for complex $z$ .
Arguments	
$z$	real or complex number

---

## acot Trigonometric

Syntax	$\text{acot}(z)$
Description	Returns the inverse cotangent of $z$ (in radians). The result is between $0$ and $\pi$ if $z$ is real. For complex $z$ , the result is the principal value.
Arguments	
$z$	real or complex number

---

## acoth Hyperbolic

Syntax	$\text{acoth}(z)$
Description	Returns the inverse hyperbolic cotangent of $z$ . The result is the principal value for complex $z$ .
Arguments	
$z$	real or complex number

---

## acsc Trigonometric

Syntax	$\text{acsc}(z)$
Description	Returns the inverse cosecant of $z$ (in radians). The result is the principal value for complex $z$ .
Arguments	
$z$	real or complex number

---

**acsch**

Hyperbolic

Syntax      `acsch(z)`Description      Returns the inverse hyperbolic cosecant of  $z$ . The result is the principal value for complex  $z$ .Arguments  
 $z$       real or complex number

---

**Ai**      *(Professional)*

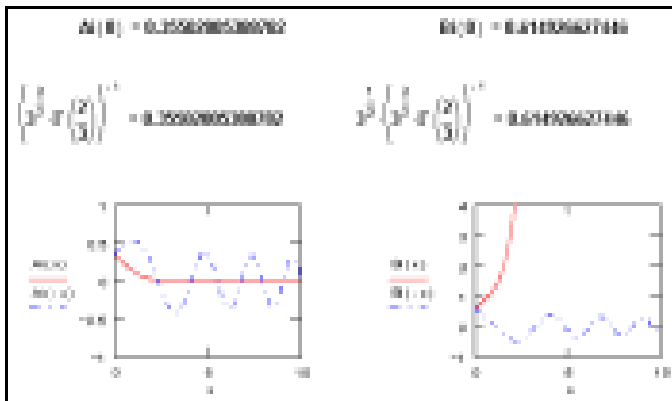
Bessel

Syntax      `Ai(x)`

Description      Returns the value of the Airy function of the first kind.

Arguments  
 $x$       real number

Example

Comments      This function is a solution of the differential equation:  $\frac{d^2}{dx^2}y - x \cdot y = 0$ .

Algorithm      Asymptotic expansion (Abramowitz and Stegun, 1972)

See also      Bi

---

**angle**

Trigonometric

Syntax      `angle(x, y)`Description      Returns the angle (in radians) from positive  $x$ -axis to point  $(x, y)$  in  $x$ - $y$  plane. The result is between  $0$  and  $2\pi$ .Arguments  
 $x, y$       real numbersSee also      `arg`, `atan`, `atan2`

---

**APPENDPRN**

File Access

Syntax	<code>APPENDPRN(<i>file</i>) := <b>A</b></code>
Description	Appends a matrix <b>A</b> to an existing structured ASCII data file. Each row in the matrix becomes a new line in the data file. Existing data must have as many columns as <b>A</b> . The function must appear alone on the left side of a definition.
Arguments	
<i>file</i>	string variable corresponding to structured ASCII data filename or path
See also	WRITEPRN for more details

---

**arg**

Complex Numbers

Syntax	<code>arg(<i>z</i>)</code>
Description	Returns the angle (in radians) from the positive real axis to point $z$ in the complex plane. The result is between $-\pi$ and $\pi$ . Returns the same value as that of $\theta$ when $z$ is written as $r \cdot e^{i \cdot \theta}$ .
Arguments	
<i>z</i>	real or complex number
See also	angle, atan, atan2

---

**asec**

Trigonometric

Syntax	<code>asec(<i>z</i>)</code>
Description	Returns the inverse secant of $z$ (in radians). The result is the principal value for complex $z$ .
Arguments	
<i>z</i>	real or complex number

---

**asech**

Hyperbolic

Syntax	<code>asech(<i>z</i>)</code>
Description	Returns the inverse hyperbolic secant of $z$ . The result is the principal value for complex $z$ .
Arguments	
<i>z</i>	real or complex number

---

<b>asin</b>	Trigonometric
Syntax	$\text{asin}(z)$
Description	Returns the inverse sine of $z$ (in radians). The result is between $-\pi/2$ and $\pi/2$ if $z$ is real. For complex $z$ , the result is the principal value.
Arguments	
$z$	real or complex number

---

<b>asinh</b>	Hyperbolic
Syntax	$\text{asinh}(z)$
Description	Returns the inverse hyperbolic sine of $z$ . The result is the principal value for complex $z$ .
Arguments	
$z$	real or complex number

---

<b>atan</b>	Trigonometric
Syntax	$\text{atan}(z)$
Description	Returns the inverse tangent of $z$ (in radians). The result is between $-\pi/2$ and $\pi/2$ if $z$ is real. For complex $z$ , the result is the principal value.
Arguments	
$z$	real or complex number
See also	angle, arg, atan2

---

<b>atan2</b>	Trigonometric
Syntax	$\text{atan2}(x, y)$
Description	Returns the angle (in radians) from positive $x$ -axis to point $(x, y)$ in $x$ - $y$ plane. The result is between $-\pi$ and $\pi$ .
Arguments	
$x, y$	real numbers
See also	angle, arg, atan

---

<b>atanh</b>	Hyperbolic
Syntax	$\text{atanh}(z)$
Description	Returns the inverse hyperbolic tangent of $z$ . The result is the principal value for complex $z$ .
Arguments	
$z$	real or complex number

---

---

**augment**

Vector and Matrix

**Syntax**      `augment(A, B)`

**Description**      Returns a matrix formed by placing the matrices **A** and **B** side by side.

**Arguments**  
**A, B**      two matrices or vectors; **A** and **B** must have the same number of rows

**Example**

$$\begin{array}{l} A = \begin{bmatrix} \sqrt{2} \\ a \\ b \end{bmatrix} \quad B = \text{identity}(3) \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \\ \text{augment}(A, B) = \begin{bmatrix} 1.41421 & 0 & 0 & 0 \\ 2.71828 & 0 & 1 & 0 \\ 3.14159 & 0 & 0 & 1 \end{bmatrix} \quad \text{stack}(A, B) = \begin{bmatrix} 1.41421 & 2.71828 & 3.14159 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

**See also**      `stack`

---

**bei***(Professional)*

Bessel

**Syntax**      `bei(n, x)`

**Description**      Returns the value of the imaginary Bessel Kelvin function of order  $n$ .

**Arguments**  
 $n$       integer,  $n \geq 0$   
 $x$       real number

**Comments**      The function  $\text{ber}(n, x) + i \cdot \text{bei}(n, x)$  is a solution of the differential equation:  
$$x^2 \frac{d^2}{dx^2} y + x \cdot \frac{d}{dx} y - (i \cdot x^2 + n^2) \cdot y = 0.$$

**Algorithm**      Series expansion (Abramowitz and Stegun, 1972)

**See also**      `ber`

---

**ber***(Professional)*

Bessel

**Syntax**      `ber(n, x)`

**Description**      Returns the value of the real Bessel Kelvin function of order  $n$ .

**Arguments**  
 $n$       integer,  $n \geq 0$   
 $x$       real number

Comments The function  $\text{ber}(n, x) + i \cdot \text{bei}(n, x)$  is a solution of the differential equation:

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

Algorithm Series expansion (Abramowitz and Stegun, 1972)

See also `bei`

---

**Bi** *(Professional)* Bessel

Syntax `Bi(x)`

Description Returns the value of the Airy function of the second kind.

Arguments  
`x` real number

Comments This function is a solution of the differential equation:

$$\frac{d^2}{dx^2} y - x \cdot y = 0.$$

Algorithm Asymptotic expansion (Abramowitz and Stegun, 1972)

See also `Ai` for example

---

**bspline** Interpolation and Prediction

Syntax `bspline(vx, vy, u, n)`

Description Returns the vector of coefficients of a B-spline of degree `n`, given the knot locations indicated by the values in `u`. The output vector becomes the first argument of the `interp` function.

Arguments  
`vx, vy` real vectors of the same size; elements of `vx` must be in ascending order  
`u` real vector with `n - 1` fewer elements than `vx`; elements of `u` must be in ascending order  
`n` integer equal to 1, 2, or 3; represents the degree of the individual piecewise linear, quadratic, or cubic polynomial fits

Comments The knots, those values where the pieces fit together, are contained in the input vector `u`. This is unlike traditional splines (`lspline`, `cspline`, and `pspline`) where the knots are forced to be the values contained in the vector `vx`. The fact that knots are chosen or modified by the user gives `bspline` more flexibility than the other splines.

See also `lspline` for more details

---



---

<b>bulstoer</b>	<i>(Professional)</i>	Differential Equation Solving
Syntax	bulstoer(y, x1, x2, acc, <b>D</b> , kmax, save)	
Description	Solves a differential equation using the smooth Bulirsch-Stoer method. Provides DE solution estimate at x2.	
Arguments	<i>Several arguments for this function are the same as described for rkfixed.</i>	
y	real vector of initial values	
x1, x2	real endpoints of the solution interval	
acc	real $acc > 0$ controls the accuracy of the solution; a small value of <i>acc</i> forces the algorithm to take smaller steps along the trajectory, thereby increasing the accuracy of the solution. Values of <i>acc</i> around 0.001 will generally yield accurate solutions.	
<b>D</b> (x, y)	real vector-valued function containing the derivatives of the unknown functions	
kmax	integer $kmax > 0$ specifies maximum number of intermediate points at which the solution is approximated; places an upper bound on the number of rows of the matrix returned by these functions	
save	real $save > 0$ specifies the smallest allowable spacing between values at which the solutions are approximated; places a lower bound on the difference between any two numbers in the first column of the matrix returned by the function	
Comments	The specialized DE solvers <b>Bulstoer</b> , <b>Rkadapt</b> , <b>Stiffb</b> , and <b>StiffR</b> provide the solution $y(x)$ over a number of uniformly spaced $x$ -values in the integration interval bounded by $x1$ and $x2$ . When you want the value of the solution at only the endpoint, $y(x2)$ , use <b>bulstoer</b> , <b>rkadapt</b> , <b>stiffb</b> , and <b>stiffR</b> instead.	
Algorithm	Adaptive step Bulirsch-Stoer method (Press <i>et al.</i> , 1992)	
See also	rkfixed, a more general differential equation solver, for information on output and arguments.	

---

<b>Bulstoer</b>	<i>(Professional)</i>	Differential Equation Solving
Syntax	Bulstoer(y, x1, x2, npts, <b>D</b> )	
Description	Solves a differential equation using the smooth Bulirsch-Stoer method. Provides DE solution at equally spaced $x$ -values by repeated calls to <b>bulstoer</b> .	
Arguments	<i>All arguments for this function are the same as described for rkfixed.</i>	
y	real vector of initial values	
x1, x2	real endpoints of the solution interval	
npts	integer $npts > 0$ specifies the number of points beyond initial point at which the solution is to be approximated; controls the number of rows in the matrix output	
<b>D</b> (x,y)	real vector-valued function containing the derivatives of the unknown functions	

Comments	When you know the solution is smooth, use the <b>Bulstoer</b> function instead of <b>rkfixed</b> . The <b>Bulstoer</b> function uses the Bulirsch-Stoer method which is slightly more accurate under these circumstances than the Runge-Kutta method used by <b>rkfixed</b> .
Algorithm	Fixed step Bulirsch-Stoer method with adaptive intermediate steps (Press <i>et al.</i> , 1992)
See also	<b>rkfixed</b> , a more general differential equation solver, for information on output and arguments.

**bvalfit** (Professional) Differential Equation Solving

Syntax **bvalfit(v1, v2, x1, x2, xf, D, load1, load2, score)**

Description Converts a boundary value differential equation to initial/terminal value problems. Useful when derivatives have a single discontinuity at an intermediate point *xf*.

Arguments

- v1** real vector containing guesses for initial values left unspecified at *x1*
- v2** real vector containing guesses for initial values left unspecified at *x2*
- x1, x2* real endpoints of the interval on which the solution to the DEs are evaluated
- xf* point between *x1* and *x2* at which the trajectories of the solutions beginning at *x1* and those beginning at *x2* are constrained to be equal
- D(x, y)** real *n*-element vector-valued function containing the derivatives of the unknown functions
- load1(x1, v1)** real vector-valued function whose *n* elements correspond to the values of the *n* unknown functions at *x1*. Some of these values are constants specified by your initial conditions. If a value is unknown, you should use the corresponding guess value from **v1**
- load2(x2, v2)** analogous to **load1** but for values taken by the *n* unknown functions at *x2*
- score(xf, y)** real *n*-element vector-valued function used to specify how you want the solutions to match at *xf*. One usually defines **score(xf, y) := y** to make the solutions to all unknown functions match up at *xf*

Example

```

Solve  $y'' = \begin{pmatrix} y \\ -y \end{pmatrix}$  for  $x \in [0, 1]$  where  $y(0) = 1$  and  $y(1) = 2$ 
for  $x \in [1, 2]$  where  $y(1) = 1$  and  $y(2) = 2$ 

D(x, y) =  $\begin{pmatrix} y_1 \\ (x \in [0, 1]) ? y_0 + (x \cdot 2) - y_0 \end{pmatrix}$ 

v1_0 = 1 ← guess value for y'(0)
v2_0 = 1 ← guess value for y'(1)  af = 1 ← point of discontinuity

load1(x1, v1) =  $\begin{pmatrix} 1 \\ v1_0 \end{pmatrix}$  ← y(0)
← guess value for y'(0)

load2(x2, v2) =  $\begin{pmatrix} 2 \\ v2_0 \end{pmatrix}$  ← y(1)
← guess value for y'(1)

score(af, y) = y ← tells Method to match the two halves of the solution at x=af

E = bvalfit(v1, v2, 0, 1, 1, 2, D, load1, load2, score)
S = [ 0.000 -0.678 ] ← contains ( y'(0) y'(1) )

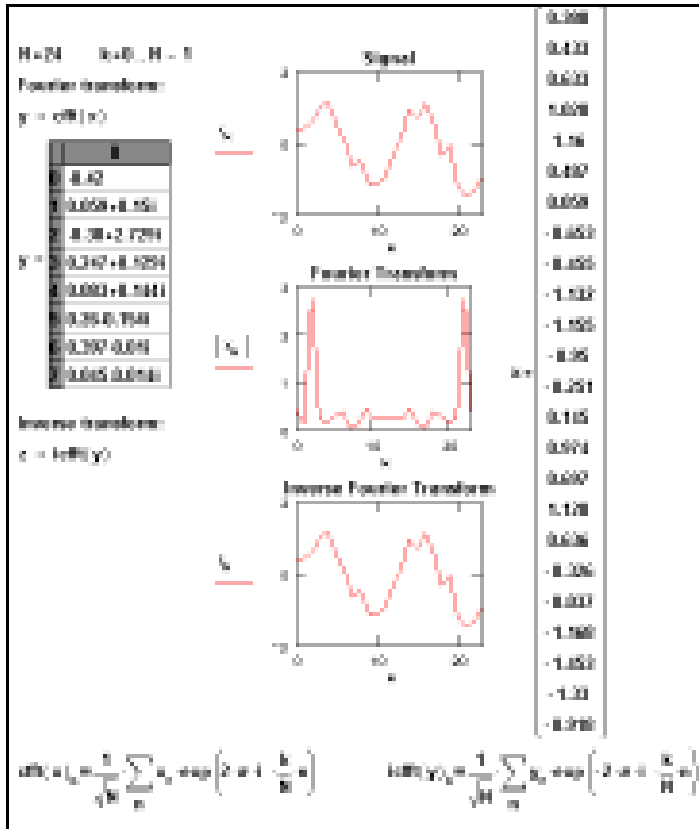
```

Comments	<p>If you have information at the initial and terminal points, then use <code>sbval</code>. If, instead, you know something about the solution and its first <math>n - 1</math> derivatives at some intermediate value <math>x_f</math>, then use <code>bvalfit</code>.</p> <p><code>bvalfit</code> solves a two-point boundary value problem of this type by shooting from the endpoints and matching the trajectories of the solution and its derivatives at the intermediate point. <code>bvalfit</code> is especially useful when a derivative has a discontinuity somewhere in the integration interval, as the above example illustrates. <code>bvalfit</code> does not return a solution to a differential equation. It merely computes the initial values the solution must have in order for the solution to match the final values you specify. You must then take the initial values returned by <code>bvalfit</code> and solve the resulting initial value problem using <code>rkfixed</code> or any of the other more specialized DE solvers.</p>
Algorithm	Shooting method with 4th order Runge-Kutta method (Press <i>et al.</i> , 1992)
See also	<code>rkfixed</code> , for more information on output and arguments.

<b>ceil</b>	Truncation and Round-off
Syntax	<code>ceil(x)</code>
Description	Returns the least integer $\geq x$ .
Arguments	
$x$	real number
See also	<code>floor</code> for more details, <code>round</code> , <code>trunc</code>

<b>cfft</b>	Fourier Transform
Syntax	<code>cfft(A)</code>
Description	Returns the fast discrete Fourier transform of complex data (representing measurements at regular intervals in the time domain). Returns an array of the same size as its argument.
Arguments	
<b>A</b>	real or complex matrix or vector

## Example



## Comments

There are two reasons why you may not be able to use the `fft/ifft` Fourier transform pair discussed elsewhere:

- The data may be complex-valued, hence Mathcad can no longer exploit the symmetry present in the real-valued case.
- The data vector might not have exactly  $2^m$  data points in it, hence Mathcad cannot take advantage of the efficient FFT algorithm used by the `fft/ifft` pair.

Although the `cfft/icfft` pair works on arrays of any size, the functions work significantly faster when the number of rows and columns contains many smaller factors. Vectors with length  $2^m$  fall into this category, as do vectors having lengths like 100 or 120. Conversely, a vector whose length is a large prime number slows down the Fourier transform algorithm.

## Algorithm

Singleton method (Singleton, 1986)

## See also

`fft` for more details

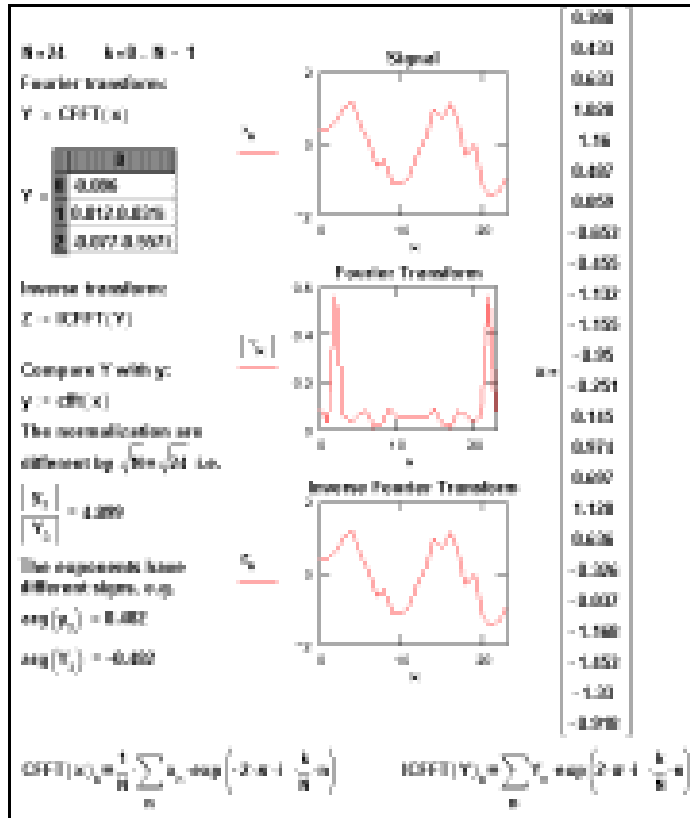
Syntax CFFT(A)

Description Returns the fast discrete Fourier transform of complex data (representing measurements at regular intervals in the time domain). Returns an array of the same size as its argument. Identical to `fft(A)`, except uses a different normalizing factor and sign convention (see example).

Arguments

A real or complex matrix or vector

Example



Algorithm Singleton method (Singleton, 1986)

See also `fft` for more details

---

**cholesky** *(Professional)* Vector and MatrixSyntax `cholesky(M)`Description Returns a lower triangular matrix **L** satisfying the equation  $\mathbf{L} \cdot \mathbf{L}^T = \mathbf{M}$ .Arguments  
**M** real, symmetric, positive definite, square matrixComments `cholesky` takes **M** to be symmetric, in the sense that it uses only the upper triangular part of **M** and assumes it to match the lower triangular part.

---

**cnorm** Probability DistributionSyntax `cnorm(x)`Description Returns the cumulative standard normal distribution. Same as `pnorm(x, 0, 1)`.Arguments  
*x* real numberComments `cnorm` is provided mainly for compatibility with documents created in earlier versions of Mathcad.

---

**cols** Vector and MatrixSyntax `cols(A)`Description Returns the number of columns in array **A**.Arguments  
**A** matrix or vector

Example



The screenshot shows a Mathcad interface with a matrix **M** defined as:

$$\mathbf{M} := \begin{pmatrix} 0 & 1 \\ 5 & 3 \\ 6 & -2 \end{pmatrix}$$

To the right of the matrix, the following text is displayed:

`cols(M) = 2`      *n* - Number, esp. of rows and columns in **M**  
`rows(M) = 3`

See also `rows`

---

**combin**

Number Theory/Combinatorics

Syntax	<code>combin(<math>n</math>, <math>k</math>)</code>
Description	Returns the number of subsets each of size $k$ that can be formed from $n$ objects.
Arguments	$n$ , $k$ integers, $0 \leq k \leq n$
Comments	Each such subset is known as a combination. The number of combinations is $C_k^n = \frac{n!}{k! \cdot (n-k)!}$ .
See also	<code>permut</code>

---

**concat***(Professional)*

String

Syntax	<code>concat(<math>S1</math>, <math>S2</math>, <math>S3</math>, ...)</code>
Description	Appends string $S2$ to the end of string $S1$ , string $S3$ to the end of string $S2$ , and so on.
Arguments	$S1$ , $S2$ , $S3$ , ... string expressions

---

**cond1***(Professional)*

Vector and Matrix

Syntax	<code>cond1(<math>\mathbf{M}</math>)</code>
Description	Returns the condition number of the matrix $\mathbf{M}$ based on the $L_1$ norm.
Arguments	$\mathbf{M}$ real or complex square matrix

---

**cond2***(Professional)*

Vector and Matrix

Syntax	<code>cond2(<math>\mathbf{M}</math>)</code>
Description	Returns the condition number of the matrix $\mathbf{M}$ based on the $L_2$ norm.
Arguments	$\mathbf{M}$ real or complex square matrix
Algorithm	Singular value computation (Wilkinson and Reinsch, 1971)

---

**conde***(Professional)*

Vector and Matrix

Syntax	<code>conde(<math>\mathbf{M}</math>)</code>
Description	Returns the condition number of the matrix $\mathbf{M}$ based on the Euclidean norm.
Arguments	$\mathbf{M}$ real or complex square matrix

---

<b>condi</b>	<i>(Professional)</i>	Vector and Matrix
Syntax	condi( <b>M</b> )	
Description	Returns the condition number of the matrix <b>M</b> based on the infinity norm.	
Arguments		
<b>M</b>	real or complex square matrix	

---

<b>corr</b>		Statistics
Syntax	corr( <b>A</b> , <b>B</b> )	
Description	Returns the Pearson correlation coefficient for the elements in two $m \times n$ arrays <b>A</b> and <b>B</b> :	
	$\text{corr}(\mathbf{A}, \mathbf{B}) = \frac{\text{cvar}(\mathbf{A}, \mathbf{B})}{\text{stdev}(\mathbf{A}) \cdot \text{stdev}(\mathbf{B})}$	
Arguments		
<b>A</b> , <b>B</b>	real or complex $m \times n$ matrices or vectors of the same size	
See also	cvar	

---

<b>cos</b>		Trigonometric
Syntax	cos( $z$ ), for $z$ in radians; cos( $z$ -deg), for $z$ in degrees	
Description	Returns the cosine of $z$ .	
Arguments		
$z$	real or complex number	

---

<b>cosh</b>		Hyperbolic
Syntax	cosh( $z$ )	
Description	Returns the hyperbolic cosine of $z$ .	
Arguments		
$z$	real or complex number	

---

<b>cot</b>		Trigonometric
Syntax	cot( $z$ ), for $z$ in radians; cot( $z$ -deg), for $z$ in degrees	
Description	Returns the cotangent of $z$ .	
Arguments		
$z$	real or complex number	

---



---

**coth**

Hyperbolic

Syntax	$\text{coth}(z)$
Description	Returns the hyperbolic cotangent of $z$ .
Arguments	
$z$	real or complex number

---

**csc**

Trigonometric

Syntax	$\text{csc}(z)$ , for $z$ in radians; $\text{csc}(z:\text{deg})$ , for $z$ in degrees
Description	Returns the cosecant of $z$ .
Arguments	
$z$	real or complex number

---

**csch**

Hyperbolic

Syntax	$\text{csch}(z)$
Description	Returns the hyperbolic cosecant of $z$ .
Arguments	
$z$	real or complex number

---

**csgn**

Complex Numbers

Syntax	$\text{csgn}(z)$
Description	Returns 0 if $z=0$ , 1 if $\text{Re}(z)>0$ or ( $\text{Re}(z)=0$ and $\text{Im}(z)>0$ ), $-1$ otherwise.
Arguments	
$z$	real or complex number
See also	sign, signum

---

**csort**

Sorting

Syntax	<code>csort(A, j)</code>
Description	Sorts the rows of the matrix <b>A</b> by placing the elements in column $j$ in ascending order. The result is the same size as <b>A</b> .
Arguments	
<b>A</b>	$m \times n$ matrix or vector
<b>j</b>	integer, $0 \leq j \leq n - 1$
Algorithm	Heap sort (Press <i>et al.</i> , 1992)
See also	<code>sort</code> for more details, <code>rsort</code>

---

**cspline**

Interpolation and Prediction

*One-dimensional Case*

Syntax	<code>cspline(vx, vy)</code>
Description	Returns the vector of coefficients of a cubic spline with cubic ends. This vector becomes the first argument of the <code>interp</code> function.
Arguments	
<b>vx, vy</b>	real vectors of the same size; elements of <b>vx</b> must be in ascending order

*Two-dimensional Case*

Syntax	<code>cspline(Mxy, Mz)</code>
Description	Returns the vector of coefficients of a two-dimensional cubic spline, constrained to be cubic at region boundaries spanned by <b>Mxy</b> . This vector becomes the first argument of the <code>interp</code> function.
Arguments	
<b>Mxy</b>	$n \times 2$ matrix whose elements, $Mxy_{i,0}$ and $Mxy_{i,1}$ , specify the $x$ - and $y$ -coordinates along the <i>diagonal</i> of a rectangular grid. This matrix plays exactly the same role as <b>vx</b> in the one-dimensional case described above. Since these points describe a diagonal, the elements in each column of <b>Mxy</b> must be in ascending order ( $Mxy_{i,k} < Mxy_{j,k}$ whenever $i < j$ ).
<b>Mz</b>	$n \times n$ matrix whose $ij$ th element is the $z$ -coordinate corresponding to the point $x = Mxy_{i,0}$ and $y = Mxy_{j,1}$ . <b>Mz</b> plays exactly the same role as <b>vy</b> does in the one-dimensional case above.
Algorithm	Tridiagonal system solving (Press <i>et al.</i> , 1992; Lorzczak)
See also	<code>lspline</code> for more details

---

**cvar**

Statistics

Syntax	<code>cvar(A, B)</code>
Description	Returns the covariance of the elements in two $m \times n$ arrays <b>A</b> and <b>B</b> : $\text{cvar}(\mathbf{A}, \mathbf{B}) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [A_{i,j} - \text{mean}(\mathbf{A})] \overline{[B_{i,j} - \text{mean}(\mathbf{B})]}$ , where the bar indicates complex conjugation.
Arguments	
<b>A, B</b>	real or complex $m \times n$ matrices or vectors
See also	<code>corr</code>

---

**dbeta**

Probability Density

Syntax	<code>dbeta(x, s1, s2)</code>
Description	Returns the probability density for a beta distribution: $\frac{\Gamma(s_1 + s_2)}{\Gamma(s_1) \cdot \Gamma(s_2)} \cdot x^{s_1 - 1} \cdot (1 - x)^{s_2 - 1}$ .
Arguments	
<i>x</i>	real number, $0 < x < 1$
<i>s1, s2</i>	real shape parameters, $s_1 > 0, s_2 > 0$

---

**dbinom**

Probability Density

Syntax	<code>dbinom(k, n, p)</code>
Description	Returns $\Pr(X = k)$ when the random variable $X$ has the binomial distribution: $\frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$ .
Arguments	
<i>k, n</i>	integers, $0 \leq k \leq n$
<i>p</i>	real number, $0 \leq p \leq 1$

---

**dcauchy**

Probability Density

Syntax	<code>dcauchy(x, l, s)</code>
Description	Returns the probability density for the Cauchy distribution: $(\pi s (1 + ((x - l)/s)^2))^{-1}$ .
Arguments	
<i>x</i>	real number
<i>l</i>	real location parameter
<i>s</i>	real scale parameter, $s > 0$

---

**dchisq**

Probability Density

Syntax `dchisq(x, d)`Description Returns the probability density for the chi-squared distribution:  $\frac{e^{-x/2}}{2\Gamma(d/2)}\left(\frac{x}{2}\right)^{(d/2-1)}$ .

Arguments

$x$  real number,  $x \geq 0$   
 $d$  integer degrees of freedom,  $d > 0$

---

**dexp**

Probability Density

Syntax `dexp(x, r)`Description Returns the probability density for the exponential distribution:  $re^{-rx}$ .

Arguments

$x$  real number,  $x \geq 0$   
 $r$  real rate,  $r > 0$

---

**dF**

Probability Density

Syntax `dF(x, d1, d2)`

Description Returns the probability density for the F distribution:

$$\frac{d_1^{d_1/2} d_2^{d_2/2} \Gamma((d_1 + d_2)/2)}{\Gamma(d_1/2) \Gamma(d_2/2)} \cdot \frac{x^{(d_1-2)/2}}{(d_2 + d_1 x)^{(d_1+d_2)/2}}$$

Arguments

$x$  real number,  $x \geq 0$   
 $d1, d2$  integer degrees of freedom,  $d_1 > 0, d_2 > 0$

---

**dgamma**

Probability Density

Syntax `dgamma(x, s)`Description Returns the probability density for the gamma distribution:  $\frac{x^{s-1} e^{-x}}{\Gamma(s)}$ .

Arguments

$x$  real number,  $x \geq 0$   
 $s$  real shape parameter,  $s > 0$

---

**dgeom**

Probability Density

Syntax	$dgeom(k, p)$
Description	Returns $\Pr(X = k)$ when the random variable $X$ has the geometric distribution: $p(1 - p)^k$ .
Arguments	
$k$	integer, $k \geq 0$
$p$	real number, $0 < p \leq 1$

---

**dhypergeom**

Probability Density

Syntax	$dhypergeom(m, a, b, n)$
Description	Returns $\Pr(X = m)$ when the random variable $X$ has the hypergeometric distribution: $\binom{a}{m} \cdot \binom{b}{n-m} / \binom{a+b}{n}$ where $\max\{0, n-b\} \leq m \leq \min\{n, a\}$ ; 0 for $m$ elsewhere.
Arguments	
$m, a, b, n$	integers, $0 \leq m \leq a$ , $0 \leq n - m \leq b$ , $0 \leq n \leq a + b$

---

**diag***(Professional)*

Vector and Matrix

Syntax	$diag(\mathbf{v})$
Description	Returns a diagonal matrix containing, on its diagonal, the elements of $\mathbf{v}$ .
Arguments	
$\mathbf{v}$	real or complex vector

---

**dlnorm**

Probability Density

Syntax	$dlnorm(x, \mu, \sigma)$
Description	Returns the probability density for the lognormal distribution: $\frac{1}{\sqrt{2\pi}\sigma x} \exp\left(-\frac{1}{2\sigma^2}(\ln(x) - \mu)^2\right)$ .
Arguments	
$x$	real number, $x \geq 0$
$\mu$	real logmean
$\sigma$	real logdeviation, $\sigma > 0$

---

**dlogis**

Probability Density

Syntax `dlogis(x, l, s)`Description Returns the probability density for the logistic distribution:  $\frac{\exp(-(x-l)/s)}{s(1 + \exp(-(x-l)/s))^2}$ .

Arguments

*x* real number  
*l* real location parameter  
*s* real scale parameter,  $s > 0$

---

**dnbinom**

Probability Density

Syntax `dnbinom(k, n, p)`Description Returns  $\Pr(X = k)$  when the random variable  $X$  has the negative binomial distribution:

$$\binom{n+k-1}{k} p^n (1-p)^k$$

Arguments

*k, n* integers,  $n > 0$  and  $k \geq 0$   
*p* real number,  $0 < p \leq 1$

---

**dnorm**

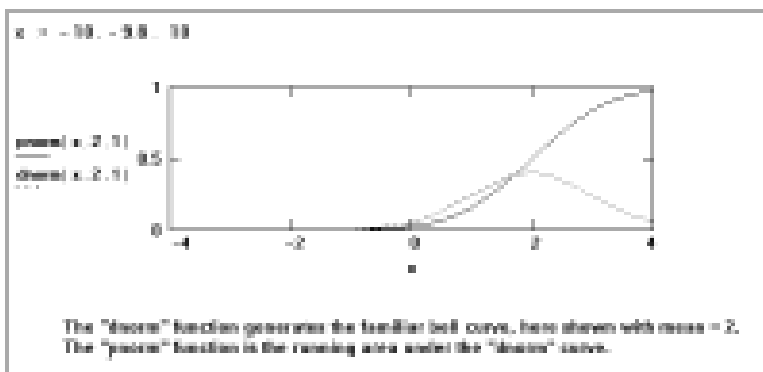
Probability Density

Syntax `dnorm(x,  $\mu$ ,  $\sigma$ )`Description Returns the probability density for the normal distribution:  $\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$ .

Arguments

*x* real number  
 *$\mu$*  real mean  
 *$\sigma$*  real standard deviation,  $\sigma > 0$

Example



---

**dpois**

Probability Density

Syntax	<code>dpois(<math>k</math>, <math>\lambda</math>)</code>
Description	Returns $\Pr(X = k)$ when the random variable $X$ has the Poisson distribution: $\frac{\lambda^k}{k!} e^{-\lambda}$ .
Arguments	
$k$	integer, $k \geq 0$
$\lambda$	real mean, $\lambda > 0$

---

**dt**

Probability Density

Syntax	<code>dt(<math>x</math>, <math>d</math>)</code>
Description	Returns the probability density for Student's $t$ distribution: $\frac{\Gamma((d+1)/2)}{\Gamma(d/2)\sqrt{\pi d}} \left(1 + \frac{x^2}{d}\right)^{-(d+1)/2}$ .
Arguments	
$x$	real number
$d$	integer degrees of freedom, $d > 0$

---

**dunif**

Probability Density

Syntax	<code>dunif(<math>x</math>, <math>a</math>, <math>b</math>)</code>
Description	Returns the probability density for the uniform distribution: $\frac{1}{b-a}$ .
Arguments	
$x$	real number, $a \leq x \leq b$
$a$ , $b$	real numbers, $a < b$

---

**dweibull**

Probability Density

Syntax	<code>dweibull(<math>x</math>, <math>s</math>)</code>
Description	Returns the probability density for the Weibull distribution: $sx^{s-1} \exp(-x^s)$ .
Arguments	
$x$	real number, $x \geq 0$
$s$	real shape parameter, $s > 0$

---

**eigenvals**

Vector and Matrix

Syntax	<code>eigenvals(<math>\mathbf{M}</math>)</code>
Description	Returns a vector of eigenvalues for the matrix $\mathbf{M}$ .
Arguments	
$\mathbf{M}$	real or complex square matrix

Example

$$A = \begin{pmatrix} 8 & -7 & 6 \\ 3 & 0 & 10 \\ 12 & 5 & -1 \end{pmatrix} \quad c = \text{eigenvals}(A) \quad c = \begin{pmatrix} 3.885 + 1.194i \\ 3.885 - 1.194i \\ -7.669 \end{pmatrix}$$

Algorithm Reduction to Hessenberg form coupled with QR decomposition (Press *et al.*, 1992)

See also `eigenvec`, `eigenvecs`

---

## **eigenvec**

Vector and Matrix

Syntax `eigenvec(M, z)`

Description Returns a vector containing the normalized eigenvector corresponding to the eigenvalue  $z$  of the square matrix  $\mathbf{M}$ .

Arguments

$\mathbf{M}$  real or complex square matrix

$z$  real or complex number

Algorithm Inverse iteration (Press *et al.*, 1992; Lorczak)

See also `eigenvals`, `eigenvecs`

---

## **eigenvecs**

(*Professional*)

Vector and Matrix

Syntax `eigenvecs(M)`

Description Returns a matrix containing the normalized eigenvectors corresponding to the eigenvalues of the matrix  $\mathbf{M}$ . The  $n$ th column of the matrix is the eigenvector corresponding to the  $n$ th eigenvalue returned by `eigenvals`.

Arguments

$\mathbf{M}$  real or complex square matrix

Algorithm Reduction to Hessenberg form coupled with QR decomposition (Press *et al.*, 1992)

See also `eigenvals`, `eigenvec`



## Example

Finding eigenvalues and eigenvectors of a real matrix . .

$$A = \begin{pmatrix} 1 & -2 & 6 \\ 3 & 0 & 10 \\ 2 & 5 & -1 \end{pmatrix} \quad c = \text{eigenvals}(A) \quad c = \begin{pmatrix} 0.185 \\ 7.467 \\ -7.602 \end{pmatrix}$$

To find all the corresponding eigenvectors at once  
(Medical Professionals)

$$v = \text{eigenvecs}(A) \quad v = \begin{pmatrix} 0.873 & 0.244 & -0.564 \\ -0.408 & 0.81 & -0.574 \\ -0.266 & 0.524 & 0.698 \end{pmatrix}$$

The first column of  $v$  is the eigenvector corresponding to 0.185, the first element of  $c$ . Similarly, the second column of  $v$  is the eigenvector corresponding to 7.467, the second element of  $c$ .

---

### erf

Special

Syntax	$\text{erf}(x)$
Description	Returns the error function $\text{erf}(x) = \int_0^x \frac{2}{\sqrt{\pi}} e^{-t^2} dt$ .
Arguments	
$x$	real number
Algorithm	Continued fraction expansion (Abramowitz and Stegun, 1972; Lorzczak)
See also	erfc

---

### erfc

Special

Syntax	$\text{erfc}(x)$
Description	Returns the complementary error function $\text{erfc}(x) := 1 - \text{erf}(x)$ .
Arguments	
$x$	real number
Algorithm	Continued fraction expansion (Abramowitz and Stegun, 1972; Lorzczak)
See also	erf

---

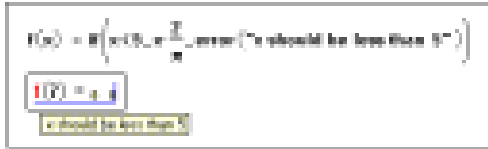
<b>error</b>	<i>(Professional)</i>	String
--------------	-----------------------	--------

Syntax	error( <i>S</i> )
--------	-------------------

Description	Returns the string <i>S</i> as an error message.
-------------	--

Arguments	
<i>S</i>	string

Example
---------



Comments	Mathcad’s built-in error messages appear as “error tips” when a built-in function is used incorrectly or could not return a result.
----------	---

Use the string function **error** to define specialized error messages that will appear when your user-defined functions are used improperly or cannot return answers. This function is especially useful for trapping erroneous inputs to Mathcad programs you write.

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.

---

<b>exp</b>	Log and Exponential
------------	---------------------

Syntax	exp( <i>z</i> )
--------	-----------------

Description	Returns the value of the exponential function $e^z$ .
-------------	---

Arguments	
<i>z</i>	real or complex number

---

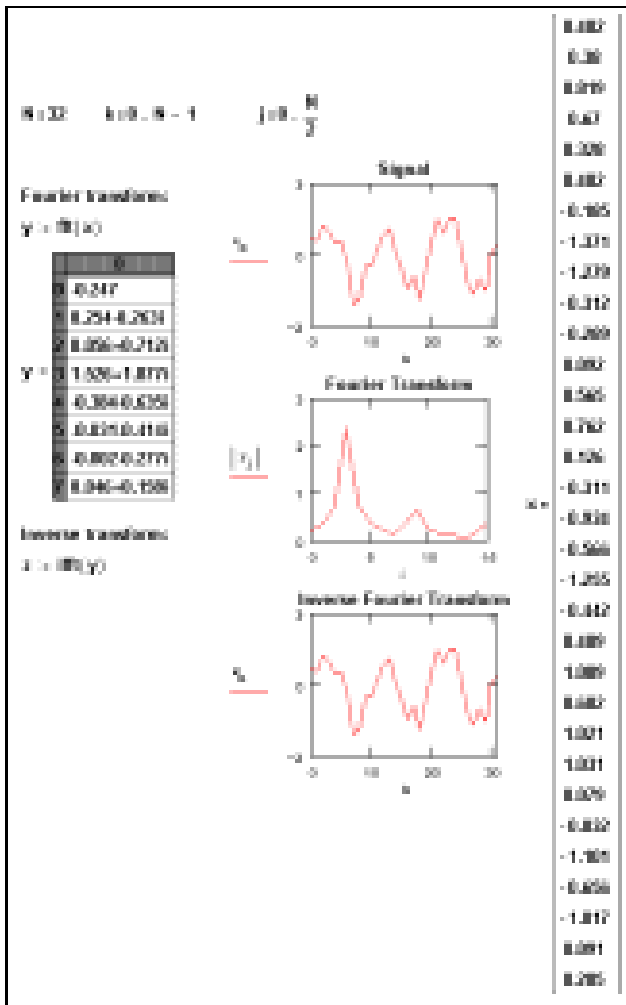
<b>fft</b>	Fourier Transform
------------	-------------------

Syntax	fft( <b>v</b> )
--------	-----------------

Description	Returns the fast discrete Fourier transform of real data. Returns a vector of size $2^{n-1} + 1$ .
-------------	--

Arguments	
<b>v</b>	real vector with $2^n$ elements (representing measurements at regular intervals in the time domain), where $n$ is an integer, $n > 0$ .

Example



Comments

When you define a vector  $v$  for use with Fourier or wavelet transforms, be sure to start with  $v_0$  (or change the value of ORIGIN). If you do not define  $v_0$ , Mathcad automatically sets it to zero. This can distort the results of the transform functions.

Mathcad comes with two types of Fourier transform pairs: `fft/ifft` and `cfft/icfft`. These functions can be applied only to discrete data (i.e., the inputs and outputs are vectors and matrices only). You cannot apply them to continuous data.

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 to save time and memory. The `cfft/icfft` pair does not assume symmetry in the transform; therefore you *must* use this pair for complex valued data. Because 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` 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.

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  $1/N$  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.

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  $\mathbf{v}$  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  $\mathbf{v}$  is an  $n$ -element vector passed to the `fft` function, and the sampling frequency is  $f_s$ , the frequency corresponding to  $c_k$  is

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

Therefore, it is 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.

Algorithm      Cooley-Tukey (Press *et al.*, 1992)

Syntax	FFT(v)
Description	Identical to <code>fft(v)</code> , except uses a different normalizing factor and sign convention. Returns a vector of size $2^{n-1} + 1$ .
Arguments	v real vector with $2^n$ elements (representing measurements at regular intervals in the time domain), where $n$ is an integer, $n > 0$ .
Comments	<p>The definitions for the Fourier transform discussed in the <code>fft</code> entry are not the only ones used. For example, the following definitions for the discrete Fourier transform and its inverse appear in Ronald Bracewell's <i>The Fourier Transform and Its Applications</i> (McGraw-Hill, 1986):</p> $F(v) = \frac{1}{n} \sum_{\tau=1}^n f(\tau) e^{-2\pi i(v/n)\tau} \quad f(\tau) = \sum_{v=1}^n F(v) e^{2\pi i(\tau/n)v}$ <p>These definitions are very common in engineering literature. To use these definitions rather than those presented in the last section, use the functions <code>FFT</code>, <code>IFFT</code>, <code>CFFT</code>, and <code>ICFFT</code>. These differ from those discussed in the last section as follows:</p> <ul style="list-style-type: none"> <li>• Instead of a factor of <math>1/\sqrt{n}</math> in front of both forms, there is a factor of <math>1/n</math> in front of the transform and no factor in front of the inverse.</li> <li>• The minus sign appears in the exponent of the transform instead of in its inverse.</li> </ul> <p>The functions <code>FFT</code>, <code>IFFT</code>, <code>CFFT</code>, and <code>ICFFT</code> are used in exactly the same way as the functions <code>fft</code>, <code>ifft</code>, <code>cfft</code>, and <code>icfft</code>.</p>
Algorithm	Cooley-Tukey (Press <i>et al.</i> , 1992)
See also	<code>fft</code> for more details

---

<b>fhyper</b>	<i>(Professional)</i>	Special
Syntax	fhyper(a, b, c, x)	
Description	Returns the value of the Gauss hypergeometric function ${}_2F_1(a, b; c; x)$ .	
Arguments	a, b, c, x real numbers, $-1 < x < 1$	

Comments The hypergeometric function is a solution of the differential equation

$$x \cdot (1-x) \cdot \frac{d^2}{dx^2}y + (c - (a+b+1) \cdot x) \cdot \frac{d}{dx}y - a \cdot b \cdot y = 0 \quad .$$

Many functions are special cases of the hypergeometric function, e.g., elementary ones like

$$\ln(1+x) = x \cdot \text{fhyper}(1, 1, 2, -x), \quad \text{asin}(x) = x \cdot \text{fhyper}\left(\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, x^2\right),$$

and more complicated ones like Legendre functions.

Algorithm Series expansion (Abramowitz and Stegun, 1972)

---

## Find

Solving

Syntax Find(*var1*, *var2*, ...)

Description Returns values of *var1*, *var2*, ... which solve a prescribed system of equations, subject to prescribed inequalities. The number of arguments matches the number of unknowns. Output is a scalar if only one argument; otherwise it is a vector of answers.

Arguments

*var1*, *var2*, ... real or complex variables; *var1*, *var2*,... must be assigned guess values before using Find.

Examples

Solve the equation:	$a^2 + 10 = e^a$
Guess values:	$a = 2$
Given	$a^2 + 10 = e^a$ $a = \text{Find}(x)$
Result is:	$a = 2.919$
Verify result:	$a^2 + 10 = 18.52$ $e^a = 18.52$

*Example 1: A solve block with one equation in one unknown.*

Intersection of Circle and line:

Given values:  $x := 1$   
 $y := 1$

Given

$x^2 + y^2 = 6$	Circle
$x + y = 2$	Line
$x \leq 1$	Inequality constraints
$y > 2$	

$\begin{pmatrix} xval \\ yval \end{pmatrix} := \text{Find}(x, y)$

Results:  $xval = -0.414$   
 $yval = 2.414$

Check that point is an actual solution:

$xval^2 + yval^2 = 6$	$xval + yval = 2$
-----------------------	-------------------

Example 2: A solve block with both equations and inequalities.

$f := 0.02$

Given

$$\frac{1}{\sqrt{f}} = -2.0 \cdot \log \left( \frac{e \cdot D^{-1}}{3.7} + \frac{2.51}{R \cdot \sqrt{f}} \right)$$

$\text{FricFac}(e, D, R) := \text{Find}(f)$

Same problem, solved for a vector of answers ...

$D := 2.5 \cdot \text{in}$   
 $e := .00005 \cdot \text{in}$

$i := 0 \dots 10$   
 $R_i := 10000 + 10000 \cdot i$   
 $f_i := \text{FricFac}(e, D, R_i)$

$f_i$
0.031
0.027
0.024
0.023
0.022
0.021

Example 3: Solving an equation repeatedly (by defining the Reynolds number  $R$  to be a range variable).

Two methods for computing a matrix square root

$$M = \begin{bmatrix} 13 & 4 & 4 \\ 4 & 9 & -3 \\ 4 & -3 & 12 \end{bmatrix}$$

$Vec = \text{eigenvals}(M)$      $Mat = \text{diag}(\text{eigenvals}(M))$      $S = \text{Vec} \cdot \sqrt{Mat} \cdot \text{Vec}^T$

$$S = \begin{bmatrix} 1.528 & 0.629 & 0.28 \\ 0.629 & 2.915 & -0.31 \\ 0.28 & -0.31 & 7.524 \end{bmatrix} \quad S^2 = \begin{bmatrix} 13 & 4 & 4 \\ 4 & 9 & -3 \\ 4 & -3 & 12 \end{bmatrix} \quad \text{using eigenanalysis}$$

$X = M$  (initial guess)  
Given  
 $X^2 = M$

$$\text{Find}(X) = \begin{bmatrix} 1.528 & 0.629 & 0.28 \\ 0.629 & 2.915 & -0.31 \\ 0.28 & -0.31 & 7.524 \end{bmatrix} \quad \text{using solve block}$$

Example 4: A solve block for computing the square root of a matrix.

$A = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$      $B = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$      $C = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$      $P = \text{identity}(2)$   
 (initial guess)

Given

$$-P \cdot A \cdot P + P \cdot B + B^T \cdot P + C = 0$$

$\text{Find}(P) = \begin{bmatrix} 1.732051 & 1 \\ 1 & 0.732051 \end{bmatrix}$     Solution of algebraic (Riccati) equation from system and control theory

Example 5: A solve block for computing the solution of a matrix equation.

## Comments

Mathcad Professional lets you numerically solve a system of up to 200 simultaneous equations in 200 unknowns. (For Mathcad Standard, the upper limit is 50 equations in 50 unknowns.) If you aren't sure that a given system possesses a solution but need an approximate answer which minimizes error, use **Minerr** instead. To solve an equation symbolically, that is, to find an exact numerical answer in terms of elementary functions, choose **Solve for Variable** from the **Symbolic** menu or use the **solve** keyword.

There are four steps to solving a system of simultaneous equations:

1. Provide initial guesses for all the unknowns you intend to solve for. These give Mathcad a place to start searching for solutions. Use complex guess values if you anticipate complex solutions; use real guess values if you anticipate real solutions.
2. Type the word **Given**. This tells Mathcad that what follows is a system of equality or inequality constraints. You can type **Given** or **given** in any style. Just don't type it while in a text region.
3. Type the equations and inequalities in any order below the word **Given**. Use **[Ctrl]=** to type “=”.



4. Finally, type the Find function with your list of unknowns. You can't put numerical values in the list of unknowns: for example, Find(2) in Example 1 isn't permitted. Like given, you can type Find or find in any style.

The word Given, the equations and inequalities that follow, and the Find function form a *solve block*.

Example 1 shows a worksheet that contains a solve block for one equation in one unknown. For one equation in one unknown, you can also use the root or polyroots functions.

Mathcad is very specific about the types of expressions that can appear between Given and Find. See Example 2. The types of allowable constraints are  $z=w$ ,  $x>y$ ,  $x<y$ ,  $x\geq y$  and  $x\leq y$ . Mathcad does not allow the following inside a solve block:

- Constraints with “ $\neq$ ”
- Range variables or expressions involving range variables of any kind
- Inequalities of the form  $a < b < c$
- Any kind of assignment statement (statements like  $\mathbf{x} := \mathbf{1}$ )

If you want to include the outcome of a solve block in an iterative calculation, see Example 3.

Solve blocks cannot be nested inside each other. Each solve block can have only one Given and one Find. You can however, define a function like  $f(x) := \text{Find}(x)$  at the end of one solve block and use this same function in another solve block.

If the solver cannot make any further improvements to the solution but the constraints are *not* all satisfied, then the solver stops and marks Find with an error message. This happens whenever the difference between successive approximations to the solution is greater than TOL *and*:

- The solver reaches a point where it cannot reduce the error any further.
- The solver reaches a point from which there is no preferred direction. Because of this, the solver has no basis on which to make further iterations.
- The solver reaches the limit of its accuracy. Roundoff errors make it unlikely that further computation would increase accuracy of the solution. This often happens if you set TOL to a value below  $10^{-15}$ .

The following problems may cause this sort of failure:

- There may actually be no solution.
- You may have given real guesses for an equation with no real solution. If the solution for a variable is complex, the solver will not find it unless the starting value for that variable is also complex.
- The solver may have become trapped in a local minimum for the error values. To find the actual solution, try using different starting values or add an inequality to keep Mathcad from being trapped in the local minimum.
- The solver may have become trapped on a point that is not a local minimum, but from which it cannot determine where to go next. Again, try changing the initial guesses or adding an inequality to avoid the undesirable stopping point.
- It may not be possible to solve the constraints to within the desired tolerance. Try defining TOL with a larger value somewhere above the solve block. Increasing the tolerance changes what Mathcad considers close enough to call a solution.

In Mathcad Professional, the context menu (available via right mouse click) associated with Find contains the following options:

- AutoSelect – chooses an appropriate algorithm
- Linear option – indicates that the problem is linear (and thus applies linear programming methods to the problem); guess values for  $var1$ ,  $var2$ ,... are immaterial (can all be zero)
- Nonlinear option – indicates that the problem is nonlinear (and thus applies these general methods to the problem: the conjugate gradient solver; if that fails to converge, the Levenberg-Marquadt solver; if that too fails, the quasi-Newton solver) – guess values for  $var1$ ,  $var2$ ,... greatly affect the solution
- Quadratic option (appears only if the Mathcad Expert Solver product is installed) – indicates that the problem is quadratic (and thus applies quadratic programming methods to the problem); guess values for  $var1$ ,  $var2$ ,... are immaterial (can all be zero)
- Advanced options – applies only to the nonlinear conjugate gradient and the quasi-Newton solvers

These options provide you more control in trying different algorithms for testing and comparison. You may also adjust the values of the built-in variables CTOL and TOL. The *constraint tolerance* CTOL controls how closely a constraint must be met for a solution to be acceptable; if CTOL were 0.001, then a constraint such as  $x < 2$  would be considered satisfied if the value of  $x$  satisfied  $x < 2.001$ . This can be defined or changed in the same way as the *convergence tolerance* TOL. The default value for CTOL is 0.

**Algorithm** For the non-linear case: Levenberg-Marquardt, Quasi-Newton, Conjugate Gradient  
 For the linear case: simplex method with branch/bound techniques  
 (Press *et al.*, 1992; Polak, 1997; Winston, 1994)

**See also** Minerr, Maximize, Minimize

## floor

Truncation and Round-off

**Syntax** floor( $x$ )

**Description** Returns the greatest integer  $\leq x$ .

**Arguments**  
 $x$  real number

**Example**

```

ceil(3.96) = 4      floor(3.96) = 3
mantissa(a) := x - floor(x)
mantissa(3.45) = 0.45
  
```

**Comments** Can be used to define the positive fractional part of a number:  $\text{mantissa}(x) := x - \text{floor}(x)$ .

**See also** ceil, round, trunc

---

**gcd**

Number Theory/Combinatorics

Syntax	<code>gcd(A)</code>
Description	Returns the largest positive integer that is a divisor of all the values in the array <b>A</b> . This integer is known as the greatest common divisor of the elements in <b>A</b> .
Arguments	
<b>A</b>	integer matrix or vector; all elements of <b>A</b> are greater than zero
Algorithm	Euclid's algorithm (Niven and Zuckerman, 1972)
See also	<code>lcm</code>

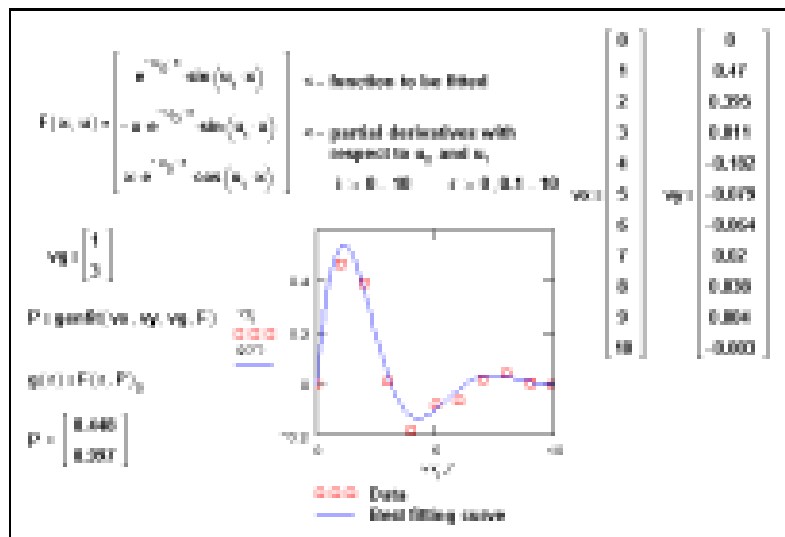
---

**genfit**

Regression and Smoothing

Syntax	<code>genfit(vx, vy, vg, F)</code>
Description	Returns a vector containing the parameters that make a function $f$ of $x$ and $n$ parameters $u_0, u_1, \dots, u_{n-1}$ best approximate the data in <b>vx</b> and <b>vy</b> .
Arguments	
<b>vx, vy</b>	real vectors of the same size
<b>vg</b>	real vector of guess values for the $n$ parameters
<b>F</b>	a function that returns an $n+1$ element vector containing $f$ and its partial derivatives with respect to its $n$ parameters

## Example



Comments	<p>The functions <code>linfit</code> and <code>genfit</code> are closely related. Anything you can do with <code>linfit</code> you can also do, albeit less conveniently, with <code>genfit</code>. The difference between these two functions is analogous to the difference between solving a system of linear equations and solving a system of nonlinear equations. The former is easily done using the methods of linear algebra. The latter is far more difficult and generally must be solved by iteration. This explains why <code>genfit</code> needs a vector of guess values as an argument and <code>linfit</code> does not.</p> <p>The example above uses <code>genfit</code> to find the exponent that best fits a set of data. By decreasing the value of the built-in TOL variable, higher accuracy in <code>genfit</code> might be achieved.</p>
Algorithm	Levenberg-Marquardt (Press <i>et al.</i> , 1992)
See also	<code>linfit</code>

---

<b>geninv</b>	<i>(Professional)</i>	Vector and Matrix
Syntax	<code>geninv(A)</code>	
Description	Returns the left inverse of a matrix <b>A</b> .	
Arguments	<b>A</b>	real $m \times n$ matrix, where $m \geq n$ .
Comments	If <b>L</b> denotes the left inverse, then $\mathbf{L} \cdot \mathbf{A} = \mathbf{I}$ where <b>I</b> is the identity matrix with $\text{cols}(\mathbf{I}) = \text{cols}(\mathbf{A})$ .	
Algorithm	SVD-based construction (Nash, 1979)	

---

<b>genvals</b>	<i>(Professional)</i>	Vector and Matrix
Syntax	<code>genvals(M, N)</code>	
Description	Returns a vector <b>v</b> of eigenvalues each of which satisfies the generalized eigenvalue equation $\mathbf{M} \cdot \mathbf{x} = v_j \cdot \mathbf{N} \cdot \mathbf{x}$ for nonzero eigenvectors <b>x</b> .	
Arguments	<b>M, N</b>	real square matrices of the same size

Example

```

M = [-2 0 0
      2 0 -1
      6 0 -5]
N = [-5 0 -1
      0 0 -3
      -2 10 0]

Vector of generalized eigenvalues:
v = genvals(M, N)
v = [ 3.177
      0.803
      0.285]

Matrix of generalized eigenvectors
which correspond to the generalized
eigenvalues in vector v:
x = genvecs(M, N)
x = [ 0.839 0.562 -0.587
      0.515 0.725 -0.21
      0.175 0.297 -0.774]
x1 = subspace(x, [1, 2, 0, 0])
x2 = subspace(x, [1, 2, 1, 1])
x3 = subspace(x, [1, 2, 2, 2])

Compare
M*x1 = [ 0.578
         1.818
         7.28 ]
M*x2 = [ 2.686
         8.5
         3.744 ]
M*x3 = [ 0.534
         1.308
         -0.969 ]
v1 (M*x1) = [ 0.578
              1.818
              7.28 ]
v2 (M*x2) = [ 2.686
              8.5
              3.744 ]
v3 (M*x3) = [ 0.534
              1.308
              -0.969 ]

```

Comments To compute the eigenvectors, use `genvec`.

Algorithm Stable QZ method (Golub and Van Loan, 1989)

**genvecs** *(Professional)* Vector and Matrix

Syntax `genvecs(M, N)`

Description Returns a matrix of normalized eigenvectors corresponding to the eigenvalues in `v`, the vector returned by `genvals`. The *j*th column of this matrix is the eigenvector `x` satisfying the generalized eigenvalue problem  $M \cdot x = v_j \cdot N \cdot x$ .

Arguments `M, N` real square matrices of the same size

Algorithm Stable QZ method (Golub and Van Loan, 1989)

See also `genvals` for example

**gmean** Statistics

Syntax `gmean(A)`

Description Returns the geometric mean of the elements of `A`:  $gmean(A) = \left( \prod_{i=0}^{m-1} \prod_{j=0}^{n-1} A_{i,j} \right)^{1/(mn)}$ .

Arguments `A` real  $m \times n$  matrix or vector with all elements greater than zero

See also `hmean`, `mean`, `median`, `mode`

---

**Her***(Professional)*

Special

Syntax

Her(*n*, *x*)

Description

Returns the value of the Hermite polynomial of degree *n* at *x*.

Arguments

*n* integer,  $n \geq 0$ *x* real number

Comments

The *n*th degree Hermite polynomial is a solution of the differential equation:

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

Algorithm

Recurrence relation (Abramowitz and Stegun, 1972)

---

**hist**

Statistics

Syntax

hist(**intervals**, **A**)

Description

Returns a vector containing the frequencies with which values in **A** fall in the intervals represented by the **intervals** vector. The resulting histogram vector is one element shorter than **intervals**.

Arguments

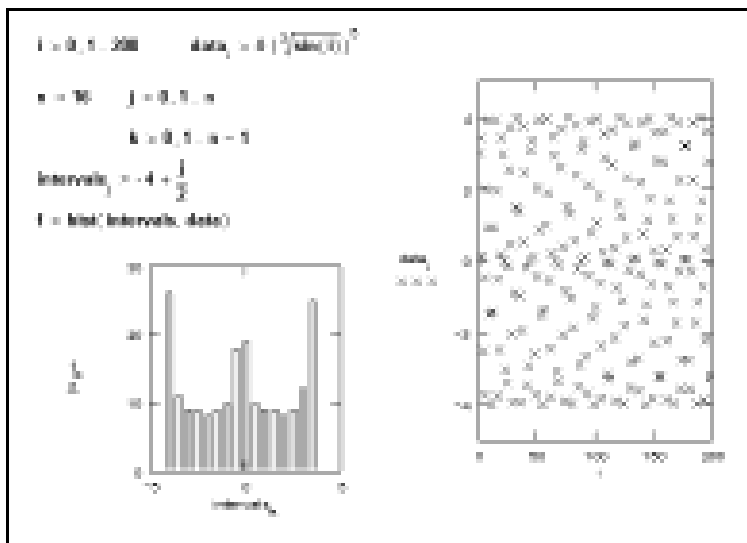
**intervals**

real vector with elements in ascending order

**A**

real matrix

Example



**Comments** The **intervals** vector contains the endpoints of subintervals constituting a partition of the data. The result of the **hist** function is a vector **f**, in which  $f_i$  is the number of values in **A** satisfying the condition  $intervals_i \leq value < intervals_{i+1}$ .

Mathcad ignores data points less than the first value in **intervals** or greater than the last value in **intervals**.

**hmean** Statistics

**Syntax** hmean(**A**)

**Description** Returns the harmonic mean of the elements of **A**: 
$$hmean(\mathbf{A}) = \left( \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \frac{1}{A_{i,j}} \right)^{-1}.$$

**Arguments**

**A** real or complex  $m \times n$  matrix or vector with all elements nonzero

**See also** gmean, mean, median, mode

**I0** Bessel

**Syntax** I0( $x$ )

**Description** Returns the value of the modified Bessel function  $I_0(x)$  of the first kind. Same as  $ln(0, x)$ .

**Arguments**

$x$  real number

**Algorithm** Small order approximation (Abramowitz and Stegun, 1972)

**I1** Bessel

**Syntax** I1( $x$ )

**Description** Returns the value of the modified Bessel function  $I_1(x)$  of the first kind. Same as  $ln(1, x)$ .

**Arguments**

$x$  real number

**Algorithm** Small order approximation (Abramowitz and Stegun, 1972)

---

<b>ibeta</b>	<i>(Professional)</i>	Special
Syntax	ibeta( <i>a</i> , <i>x</i> , <i>y</i> )	
Description	Returns the value of the incomplete beta function with parameter <i>a</i> , at ( <i>x</i> , <i>y</i> ).	
Arguments		
<i>a</i>	real number, $0 \leq a \leq 1$	
<i>x</i> , <i>y</i>	real numbers, $x > 0, y > 0$	
Comments	The incomplete beta function often arises in probabilistic applications. It is defined by the following formula: $\text{ibeta}(a, x, y) = \frac{\Gamma(x+y)}{\Gamma(x) \cdot \Gamma(y)} \cdot \int_0^a t^{x-1} \cdot (1-t)^{y-1} dt .$	
Algorithm	Continued fraction expansion (Abramowitz and Stegun, 1972)	

---

<b>icfft</b>		Fourier Transform
Syntax	icfft( <b>A</b> )	
Description	Returns the inverse Fourier transform corresponding to <b>cfft</b> . Returns an array of the same size as its argument.	
Arguments		
<b>A</b>	real or complex matrix or vector	
Comments	The <b>cfft</b> and <b>icfft</b> functions are exact inverses; $\text{icfft}(\text{cfft}(\mathbf{A})) = \mathbf{A}$ .	
Algorithm	Singleton method (Singleton, 1986)	
See also	fft for more details and <b>cfft</b> for example	

---

<b>ICFFT</b>		Fourier Transform
Syntax	ICFFT( <b>A</b> )	
Description	Returns the inverse Fourier transform corresponding to <b>CFFT</b> . Returns an array of the same size as its argument.	
Arguments		
<b>A</b>	real or complex matrix or vector	
Comments	The <b>CFFT</b> and <b>ICFFT</b> functions are exact inverses; $\text{ICFFT}(\text{CFFT}(\mathbf{A})) = \mathbf{A}$ .	
Algorithm	Singleton method (Singleton, 1986)	
See also	fft for more details and <b>CFFT</b> for example	

---



---

**identity**

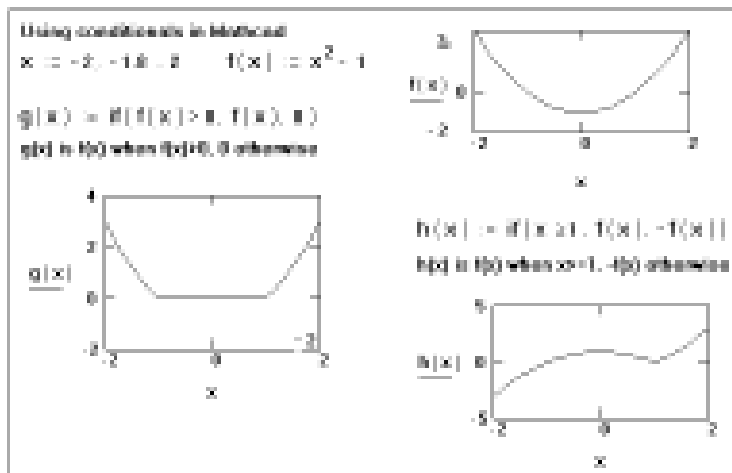
Syntax	<code>identity(<i>n</i>)</code>
Description	Returns the identity matrix of size <i>n</i> .
Arguments	
<i>n</i>	integer, $n > 0$

---

**if**

Syntax	<code>if(<i>cond</i>, <i>x</i>, <i>y</i>)</code>
Description	Returns <i>x</i> or <i>y</i> depending on the value of <i>cond</i> . If <i>cond</i> is true (non-zero), returns <i>x</i> . If <i>cond</i> is false (zero), returns <i>y</i> .
Arguments	
<i>cond</i>	arbitrary expression (usually a Boolean expression)
<i>x</i> , <i>y</i>	real or complex numbers

## Example



**Comments** Use if to define a function that behaves one way below a certain number and a different way 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. The argument *cond* is usually a Boolean expression (made up using the Boolean operators  $\mathbf{=}$ ,  $\mathbf{>}$ ,  $\mathbf{<}$ ,  $\mathbf{\geq}$ ,  $\mathbf{\leq}$  or  $\mathbf{\neq}$ ).

To save time, Mathcad evaluates only the necessary arguments. For example, if *cond* is false, there is no need to evaluate *x* because it will not be returned anyway. Therefore, errors in the unevaluated argument can escape detection. For example, Mathcad will never detect the fact that  $\ln(0)$  is undefined in the expression  $\text{if}(|z| < 0, \ln(0), \ln(z))$ .

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 if and only if *x* is between 0 and 1. Similarly, the expression  $(x > 1) + (x < 0)$  acts like an “or” gate, returning a 1 if and only if  $x > 1$  or  $x < 0$ .

**ifft** Fourier Transform

**Syntax** `ifft(v)`

**Description** Returns the inverse Fourier transform corresponding to `fft`. Returns a real vector of size  $2^n$ .

**Arguments**  
**v** real or complex vector of size  $1 + 2^{n-1}$ , where *n* is an integer.

**Comments** 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 exponent. The `fft` and `ifft` functions are exact inverses. For all real **v**,  $\text{ifft}(\text{fft}(\mathbf{v})) = \mathbf{v}$ .

**Algorithm** Cooley-Tukey (Press *et al.*, 1992)

**See also** `fft` for more details

**IFFT** Fourier Transform

**Syntax** `IFFT(v)`

**Description** Returns the inverse transform corresponding to `FFT`. Returns a real vector of size  $2^n$ .

**Arguments**  
**v** real or complex vector of size  $1 + 2^{n-1}$ , where *n* is an integer.

**Algorithm** Cooley-Tukey (Press *et al.*, 1992)

**See also** `fft` for more details

---

**Im**

Complex Numbers

Syntax	$\text{Im}(z)$
Description	Returns the imaginary part of $z$ .
Arguments	
$z$	real or complex number
See also	Re

---

**In**

Bessel

Syntax	$\text{In}(m, x)$
Description	Returns the value of the modified Bessel function $I_m(x)$ of the first kind.
Arguments	
$m$	integer, $0 \leq m \leq 100$
$x$	real number
Comments	Solution of the differential equation $x^2 \cdot \frac{d^2}{dx^2}y + x \cdot \frac{d}{dx}y - (x^2 + n^2) \cdot y = 0$ .
Algorithm	Small order approximation, upward recurrence relation (Abramowitz and Stegun, 1972; Press <i>et al.</i> , 1992)
See also	Kn

---

**intercept**

Regression and Smoothing

Syntax	$\text{intercept}(\mathbf{vx}, \mathbf{vy})$
Description	Returns the y-intercept of the least-squares regression line.
Arguments	
$\mathbf{vx}, \mathbf{vy}$	real vectors of the same size
See also	slope for more details, stderr

*One-dimensional Case*

Syntax	<code>interp(vs, vx, vy, x)</code>
Description	Interpolates the value from spline coefficients or regression coefficients. Takes three vector arguments <b>vx</b> , <b>vy</b> (of the same size) and <b>vs</b> . Returns the interpolated $y$ value corresponding to the point $x$ .
Arguments	
<b>vs</b>	real vector output from interpolation routines <code>bspline</code> , <code>cspline</code> , <code>lspline</code> , or <code>pspline</code> or regression routines <code>regress</code> or <code>loess</code>
<b>vx, vy</b>	real vectors of the same size
<b>x</b>	real number
Comments	<p>To find the interpolated value for a particular <math>x</math>, Mathcad finds the two points which <math>x</math> falls between. It then returns the <math>y</math> value on the cubic section enclosed by these two points. For <math>x</math> values less than the smallest point in <b>vx</b>, Mathcad extrapolates the cubic section connecting the smallest two points of <b>vx</b>. Similarly, for <math>x</math> values greater than the largest point in <b>vx</b>, Mathcad extrapolates the cubic section connecting the largest two points of <b>vx</b>.</p> <p>For best results, do not use the <code>interp</code> function on values of <math>x</math> far from the fitted points. Splines are intended for interpolation, not extrapolation. Consequently, computed values for such <math>x</math> values are unlikely to be useful. See <code>predict</code> for an alternative.</p>

*Two-dimensional Case*

Syntax	<code>interp(vs, Mxy, Mz, v)</code>
Description	Interpolates the value from spline coefficients or regression coefficients. Takes two matrix arguments <b>Mxy</b> and <b>Mz</b> (with the same number of rows) and one vector argument <b>vs</b> . Returns the interpolated $z$ value corresponding to the point $x = v_0$ and $y = v_1$ .
Arguments	
<b>vs</b>	real vector output from interpolation routines <code>bspline</code> , <code>cspline</code> , <code>lspline</code> , or <code>pspline</code> or regression routines <code>regress</code> or <code>loess</code>
<b>Mxy, Mz</b>	real matrices (with the same number of rows)
<b>v</b>	real two-dimensional vector
Comments	<p>For best results, do not use the <code>interp</code> function on values of <math>x</math> and <math>y</math> far from the grid points. Splines are intended for interpolation, not extrapolation. Consequently, computed values for such <math>x</math> and <math>y</math> values are unlikely to be useful. See <code>predict</code> for an alternative.</p>
See also	<code>lspline</code> for example, <code>bspline</code> , <code>cspline</code> , <code>pspline</code> , <code>regress</code> , <code>loess</code>

<b>IsArray</b>	<i>(Professional)</i>	Expression Type
Syntax	IsArray( $x$ )	
Description	Returns 1 if $x$ is a matrix or vector; 0 otherwise.	
Arguments	$x$ arbitrary real or complex number, array, or string	
<b>IsScalar</b>	<i>(Professional)</i>	Expression Type
Syntax	IsScalar( $x$ )	
Description	Returns 1 if $x$ is a real or complex number; 0 otherwise.	
Arguments	$x$ arbitrary real or complex number, array, or string	
<b>IsString</b>	<i>(Professional)</i>	Expression Type
Syntax	IsString( $x$ )	
Description	Returns 1 if $x$ is a string; 0 otherwise.	
Arguments	$x$ arbitrary real or complex number, array, or string	
<b>iwave</b>	<i>(Professional)</i>	Wavelet Transform
Syntax	iwave( $\mathbf{v}$ )	
Description	Returns the inverse wavelet transform corresponding to <b>wave</b> .	
Arguments	$\mathbf{v}$ real vector of $2^n$ elements, where $n$ is an integer, $n > 0$ .	
Algorithm	Pyramidal Daubechies 4-coefficient wavelet filter (Press <i>et al.</i> , 1992)	
See also	<b>wave</b> for example	
<b>J0</b>		Bessel
Syntax	J0( $x$ )	
Description	Returns the value of the Bessel function $J_0(x)$ of the first kind. Same as $J_n(0, x)$ .	
Arguments	$x$ real number	
Algorithm	Steed's method (Press <i>et al.</i> , 1992)	

---

<b>J1</b>		Bessel
Syntax	J1( <i>x</i> )	
Description	Returns the value of the Bessel function $J_1(x)$ of the first kind. Same as $J_n(1, x)$ .	
Arguments		
<i>x</i>	real number	
Algorithm	Steed's method (Press <i>et al.</i> , 1992)	

---

<b>Jac</b>	( <i>Professional</i> )	Special
Syntax	Jac( <i>n</i> , <i>a</i> , <i>b</i> , <i>x</i> )	
Description	Returns the value of the Jacobi polynomial of degree <i>n</i> with parameters <i>a</i> and <i>b</i> , at <i>x</i> .	
Arguments		
<i>n</i>	integer, $n \geq 0$	
<i>a</i> , <i>b</i>	real numbers, $a > -1$ , $b > -1$	
<i>x</i>	real number	
Comments	The Jacobi polynomial is a solution of the differential equation: $(1 - x^2) \cdot \frac{d^2}{dx^2}y + (b - a - (a + b + 2) \cdot x) \cdot \frac{d}{dx}y + n \cdot (n + a + b + 1) \cdot y = 0$ and includes the Chebyshev and Legendre polynomials as special cases.	
Algorithm	Recurrence relation (Abramowitz and Stegun, 1972)	

---

<b>Jn</b>		Bessel
Syntax	Jn( <i>m</i> , <i>x</i> )	
Description	Returns the value of the Bessel function $J_m(x)$ of the first kind.	
Arguments		
<i>m</i>	integer, $0 \leq m \leq 100$ .	
<i>x</i>	real number	
Comments	Solution of the differential equation $x^2 \cdot \frac{d^2}{dx^2}y + x \cdot \frac{d}{dx}y + (x^2 - n^2) \cdot y = 0$ .	
Algorithm	Steed's method (Press <i>et al.</i> , 1992)	
See also	Yn	

---

---

<b>js</b>	<i>(Professional)</i>	Bessel
Syntax	$js(n, x)$	
Description	Returns the value of the spherical Bessel function of the first kind, of order $n$ , at $x$ .	
Arguments		
$x$	real number, $x > 0$ ; $x = 0$ is permitted for $js$ if $n \geq 0$	
$n$	integer	
Comments	Solution of the differential equation: $x^2 \cdot \frac{d^2}{dx^2}y + 2 \cdot x \cdot \frac{d}{dx}y + (x^2 - n \cdot (n + 1))y = 0$ .	
Algorithm	Small order approximation, upward recurrence relation (Abramowitz and Stegun, 1972; Press <i>et al.</i> , 1992)	
See also	$ys$	

---

<b>K0</b>		Bessel
Syntax	$K0(x)$	
Description	Returns the value of the modified Bessel function $K_0(x)$ of the second kind. Same as $Kn(0, x)$ .	
Arguments		
$x$	real number, $x > 0$	
Algorithm	Small order approximation (Abramowitz and Stegun, 1972)	

---

<b>K1</b>		Bessel
Syntax	$K1(x)$	
Description	Returns the value of the modified Bessel function $K_1(x)$ of the second kind. Same as $Kn(1, x)$ .	
Arguments		
$x$	real number, $x > 0$	
Algorithm	Small order approximation (Abramowitz and Stegun, 1972)	

---

<b>Kn</b>		Bessel
Syntax	$Kn(m, x)$	
Description	Returns the value of the modified Bessel function $K_m(x)$ of the second kind.	
Arguments		
$m$	integer, $0 \leq m \leq 100$ .	
$x$	real number, $x > 0$	

---

Comments	Solution of the differential equation $x^2 \cdot \frac{d^2}{dx^2}y + x \cdot \frac{d}{dx}y - (x^2 + n^2) \cdot y = 0$ .
See also	In
Algorithm	Small order approximation, upward recurrence relation (Abramowitz and Stegun, 1972; Press <i>et al.</i> , 1992)

**ksmooth** *(Professional)* Regression and Smoothing

Syntax	<code>ksmooth(vx, vy, b)</code>
Description	Creates a new vector, of the same size as <b>vy</b> , by using a Gaussian kernel to return weighted averages of <b>vy</b> .
Arguments	<p><b>vx, vy</b> real vectors of the same size; elements of <b>vx</b> must be in ascending order</p> <p><b>b</b> real bandwidth <math>b &gt; 0</math>; controls the smoothing window and should be set to a few times the spacing between your data points on the <math>x</math>-axis, depending on how big of a window you want to use when smoothing</p>
Comments	The <code>ksmooth</code> function uses a Gaussian kernel to compute local weighted averages of the input vector <b>vy</b> . This smoother is most useful when your data lies along a band of relatively constant width. If your data lies scattered along a band whose width fluctuates considerably, you should use an adaptive smoother like <code>supsmooth</code> .

For each  $vy_i$  in the  $n$ -element vector **vy**, the `ksmooth` function returns a new  $vy'_i$  given by:

$$vy'_i = \frac{\sum_{j=1}^n K\left(\frac{vx_i - vx_j}{b}\right)vy_j}{\sum_{j=1}^n K\left(\frac{vx_i - vx_j}{b}\right)} \quad \text{where: } K(t) = \frac{1}{\sqrt{2\pi} \cdot (0.37)} \cdot \exp\left(-\frac{t^2}{2 \cdot (0.37)^2}\right)$$

and  $b$  is a bandwidth which you supply to the `ksmooth` function. The bandwidth is usually set to a few times the spacing between data points on the  $x$  axis, depending on how big a window you want to use when smoothing.

Algorithm	Moving window Gaussian kernel smoothing (Lorczak)
See also	<code>medsmooth</code> for more details, <code>supsmooth</code>

**kurt** Statistics

Syntax	<code>kurt(A)</code>
Description	Returns the kurtosis of the elements of <b>A</b> : $\text{kurt}(\mathbf{A}) = \left( \frac{mn(mn+1)}{(mn-1)(mn-2)(mn-3)} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left( \frac{\mathbf{A}_{i,j} - \text{mean}(\mathbf{A})}{\text{Stdev}(\mathbf{A})} \right)^4 \right) - \frac{3(mn-1)^2}{(mn-2)(mn-3)}$
Arguments	<p><b>A</b> real or complex <math>m \times n</math> matrix or vector; <math>m \cdot n \geq 4</math></p>



---

<b>Lag</b>	<i>(Professional)</i>	Special
Syntax	Lag( <i>n</i> , <i>x</i> )	
Description	Returns the value of the Laguerre polynomial of degree <i>n</i> at <i>x</i> .	
Arguments		
<i>n</i>	integer, $n \geq 0$	
<i>x</i>	real number	
Comments	The Laguerre polynomial is a solution of the differential equation	
	$x \cdot \frac{d^2}{dx^2}y + (1 - x) \cdot \frac{d}{dx}y + n \cdot y = 0.$	
Algorithm	Recurrence relation (Abramowitz and Stegun, 1972)	

---

<b>last</b>		Vector and Matrix
Syntax	last( <b>v</b> )	
Description	Returns the index of the last element in vector <b>v</b> .	
Arguments		
<b>v</b>	vector	
Comments	last( <b>v</b> ) = length( <b>v</b> ) - 1 + ORIGIN	
See also	rows	

---

<b>lcm</b>		Number Theory/Combinatorics
Syntax	lcm( <b>A</b> )	
Description	Returns the smallest positive integer that is a multiple of all the values in the array <b>A</b> . This integer is known as the least common multiple of the elements in <b>A</b> .	
Arguments		
<b>A</b>	integer matrix or vector; all elements of <b>A</b> are greater than zero	
Algorithm	Euclid's algorithm (Niven and Zuckerman, 1972)	
See also	gcd	

---

---

<b>Leg</b>	<i>(Professional)</i>	Special
Syntax	Leg( <i>n</i> , <i>x</i> )	
Description	Returns the value of the Legendre polynomial of degree <i>n</i> at <i>x</i> .	
Arguments		
<i>n</i>	integer, $n \geq 0$	
<i>x</i>	real number	
Comments	The Legendre polynomial is a solution of the differential equation	
	$(1 - x^2) \cdot \frac{d^2}{dx^2}y - 2 \cdot x \cdot \frac{d}{dx}y + n \cdot (n + 1) \cdot y = 0.$	
Algorithm	Recurrence relation (Abramowitz and Stegun, 1972)	

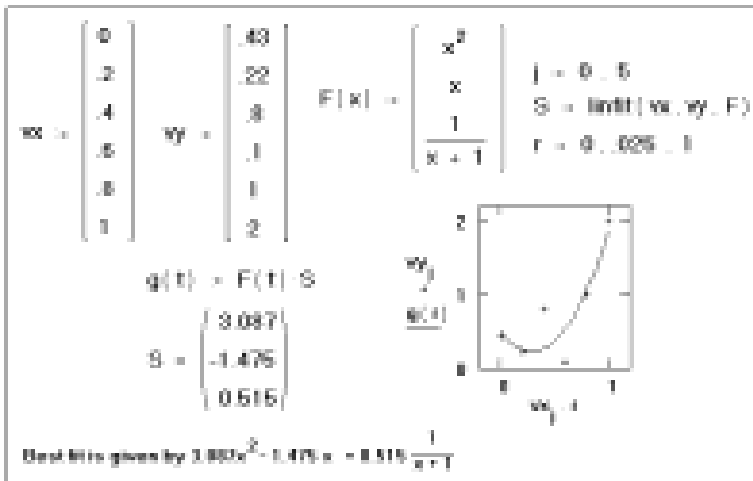
---

<b>length</b>		Vector and Matrix
Syntax	length( <b>v</b> )	
Description	Returns the number of elements in vector <b>v</b> .	
Arguments		
<b>v</b>	vector	
Comments	Same as rows( <b>v</b> )	

---

<b>linfit</b>		Regression and Smoothing
Syntax	linfit( <b>vx</b> , <b>vy</b> , <b>F</b> )	
Description	Returns a vector containing the coefficients used to create a linear combination of the functions in <b>F</b> which best approximates the data in <b>vx</b> and <b>vy</b> . See <b>genfit</b> for a more general technique.	
Arguments		
<b>vx</b> , <b>vy</b>	real vectors of the same size; elements of <b>vx</b> should be in ascending order	
<b>F</b>	a function that returns a vector of functions	

## Example



## Comments

Not all data sets can be modeled by lines or polynomials. There are times when you need to model your data with a linear combination of arbitrary functions, none of which represent terms of a polynomial. For example, in a Fourier series you try to approximate data using a linear combination of complex exponentials. Or you may believe your data can be modeled by a weighted combination of Legendre polynomials, but you just don't know what weights to assign.

The `linfit` function is designed to solve these kinds of problems. If you believe your data could be modeled by a linear combination of arbitrary functions:

$y = a_0 \cdot f_0(x) + a_1 \cdot f_1(x) + \dots + a_n \cdot f_n(x)$ , you should use `linfit` to evaluate the  $a_i$ . The example above shows a linear combination of three functions  $x$ ,  $x^2$ , and  $(x + 1)^{-1}$  to model some data.

There are times however when the flexibility of `linfit` is still not enough. Your data may have to be modeled not by a linear combination of data but by some function whose parameters must be chosen. For example, if your data can be modeled by the sum:

$f(x) = a_1 \cdot \sin(2x) + a_2 \cdot \tanh(3x)$  and all you need to do is solve for the unknown weights  $a_1$  and  $a_2$ , then the `linfit` function is sufficient. By contrast, if instead your data is to be modeled by the sum:  $f(x) = 2 \cdot \sin(a_1x) + 3 \cdot \tanh(a_2x)$  and you now have to solve for the unknown parameters  $a_1$  and  $a_2$ , you should use the `genfit` function.

## Algorithm

SVD-based least squares minimization (Press *et al.*, 1992)

## See also

`genfit`

---

## linterp

Interpolation and Prediction

### Syntax

`linterp(vx, vy, x)`

### Description

Returns a linearly interpolated value at  $x$ .

### Arguments

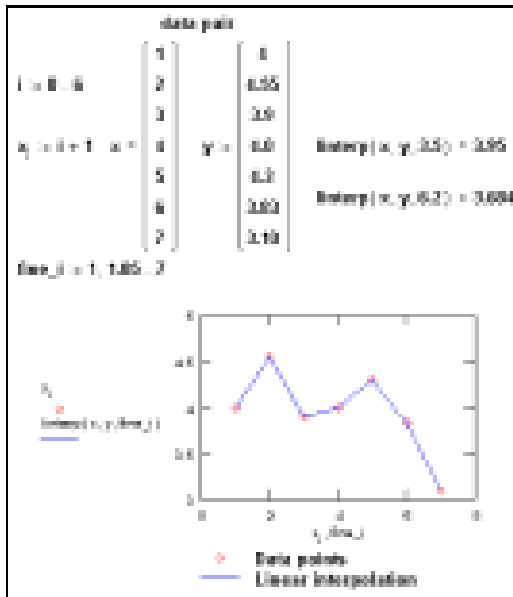
**vx, vy**

real vectors of the same size; elements of **vx** should be in ascending order

**x**

real number at which to interpolate

## Example



## Comments

Interpolation involves using existing data points to predict values between these data points. Mathcad allows you to either connect the data points with straight lines (linear interpolation) or to connect them with sections of a cubic polynomial (cubic spline interpolation).

Unlike the regression functions discussed elsewhere, these interpolation functions return a curve which must pass through the points you specify. Therefore, the resulting function is very sensitive to spurious data points. If your data is noisy, you should consider using the regression functions instead.

Be sure that every element in the  $\mathbf{vx}$  and  $\mathbf{vy}$  arrays contains a data value. Because every element in an array must have a value, Mathcad assigns 0 to any elements you have not explicitly assigned.

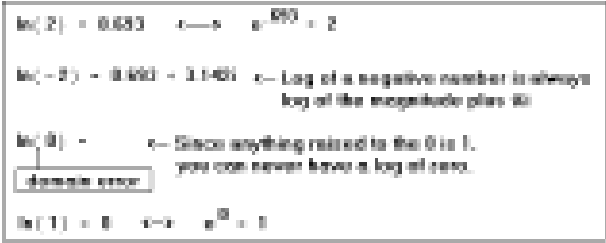
To find the interpolated value for a particular  $x$ ,  $linterp$  finds the two points between which the value falls and returns the corresponding  $y$  value on the straight line between the two points.

For  $x$  values before the first point in  $\mathbf{vx}$ ,  $linterp$  extrapolates the straight line between the first two data points. For  $x$  values beyond the last point in  $\mathbf{vx}$ ,  $linterp$  extrapolates the straight line between the last two data points.

For best results, the value of  $x$  should be between the largest and smallest values in the vector  $\mathbf{vx}$ . The  $linterp$  function is intended for interpolation, not extrapolation. Consequently, computed values for  $x$  outside this range are unlikely to be useful. See `predict` for an alternative.

---

## In

Syntax	$\ln(z)$
Description	Returns the natural logarithm of nonzero $z$ (to base $e$ ). It is the principal value (imaginary part between $\pi$ and $-\pi$ ) for complex $z$ .
Arguments	$z$ real or complex number
Example	
Comments	In general, a complex argument to the natural log function returns: $\ln(x + i \cdot y) = \ln x + i \cdot y  + \text{atan}(y/x) \cdot i + 2 \cdot n \cdot \pi \cdot i$ Mathcad's $\ln$ function returns the value corresponding to $n = 0$ , namely: $\ln(x + i \cdot y) = \ln x + i \cdot y  + \text{atan}(y/x) \cdot i$ (principal branch of the natural log function).
See also	$\log$

---

## LoadColormap

Syntax	$\text{LoadColormap}(\text{file})$
Description	Returns an array containing the values in the colormap <i>file</i> .
Arguments	<i>file</i> string variable corresponding to CMP filename
Comments	The file <i>file</i> is the name of a colormap located in the CMAPS subdirectory of your Mathcad directory. The function $\text{LoadColormap}$ is useful when you want to edit a colormap or use it to create a new colormap. See on-line Help for more information
See also	$\text{SaveColormap}$

One-dimensional Case

Syntax `loess(vx, vy, span)`

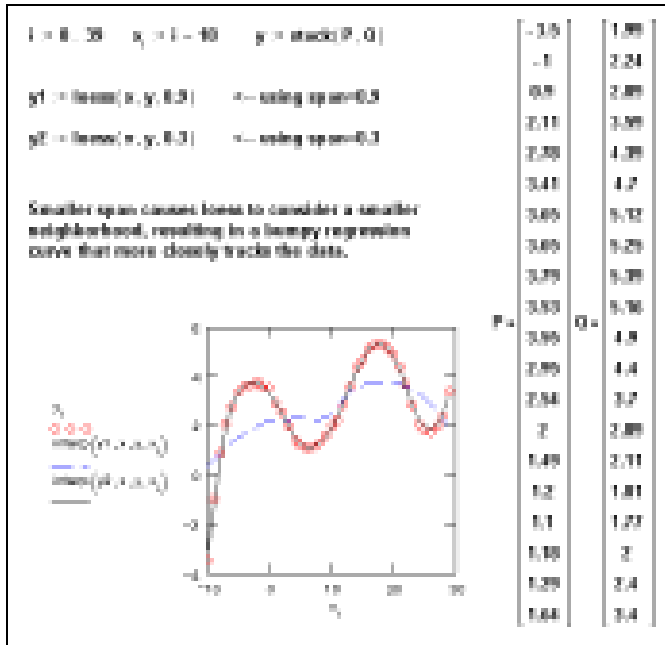
Description Returns the vector required by the `interp` function to find the set of second order polynomials that best fit particular neighborhoods of data points specified in arrays `vx` and `vy`.

Arguments

`vx, vy` real vectors of the same size

`span` real `span > 0` specifies how large a neighborhood `loess` will consider in performing this local regression

Example



Comments

Instead of generating a single polynomial the way `regress` does, `loess` generates a different second order polynomial depending on where you are on the curve. It does this by examining the data in a small neighborhood of the point you're interested in. The argument `span` controls the size of this neighborhood. As `span` gets larger, `loess` becomes equivalent to `regress` with `n = 2`. A good default value is `span = 0.75`.

The example above shows how `span` affects the fit generated by the `loess` function. A smaller value of `span` makes the fitted curve track fluctuations in data more effectively. A larger value of `span` tends to smear out fluctuations in data and thereby generates a smoother fit.

## Two-dimensional Case

Syntax	<code>loess(Mxy, vz, span)</code>
Description	Returns the vector required by the <code>interp</code> function to find the set of second order polynomials that best fit particular neighborhoods of data points specified in arrays <b>Mxy</b> and <b>vz</b> .
Arguments	
<b>Mxy</b>	real $m \times 2$ matrix containing $x$ - $y$ coordinates of the $m$ data points
<b>vz</b>	real $m$ -element vector containing the $z$ coordinates corresponding to the points specified in <b>Mxy</b>
<span><i>span</i></span>	real $span > 0$ specifies how large a neighborhood <code>loess</code> will consider in performing this local regression
Comments	Can be extended naturally to the three- and four-dimensional cases (that is, up to four independent variables).
Algorithm	Local polynomial estimation (Cleveland and Devlin, 1988)
See also	<code>regress</code> for more details

---

## log

Log and Exponential

### Classical Definition

Syntax	<code>log(z)</code>
Description	Returns the common logarithm of nonzero $z$ to base 10. The result is the principal value (imaginary part between $\pi$ and $-\pi$ ) for complex $z$ .
Arguments	
$z$	real or complex number

### Extended Definition

Syntax	<code>log(z, b)</code>
Description	Returns the logarithm of nonzero $z$ to base $b$ . The result is the principal value (imaginary part between $\pi$ and $-\pi$ ) for complex $z$ .
Arguments	
$z$	real or complex number
$b$	real number, $b > 1$
See also	<code>In</code>

---

**Isolve***(Professional)*

Vector and Matrix

Syntax

Isolve(**M**, **v**)

Description

Returns a solution vector **x** such that  $\mathbf{M} \cdot \mathbf{x} = \mathbf{v}$ .

Arguments

**M**

real or complex square matrix that is neither singular nor nearly singular

**v**

real or complex vector

Example

$3x + 6y = 9$  ← System of equations to be solved.  
 $2x + .54y = 4$

$\mathbf{M} = \begin{pmatrix} 3 & 6 \\ 2 & .54 \end{pmatrix}$     $\mathbf{v} = \begin{pmatrix} 9 \\ 4 \end{pmatrix}$  ← Create your matrix and vector.

$\text{Isolve}(\mathbf{M}, \mathbf{v}) = \begin{pmatrix} 1.344 \\ 0.576 \end{pmatrix}$  ← Values for x satisfying the system of equations.  
← Values for y satisfying the system of equations.

Note: The "Isolve" function is only available with Mathcad Professional.

Comments

A matrix is singular if its determinant is zero; it is nearly singular if it has a high condition number. Alternatively, you can solve a system of linear equations by using matrix inversion, via numeric or symbolic solve blocks.

Algorithm

LU decomposition and forward/backward substitution (Press *et al.*, 1992)

---

**Ispline**

Interpolation and Prediction

*One-dimensional Case*

Syntax

Ispline(**vx**, **vy**)

Description

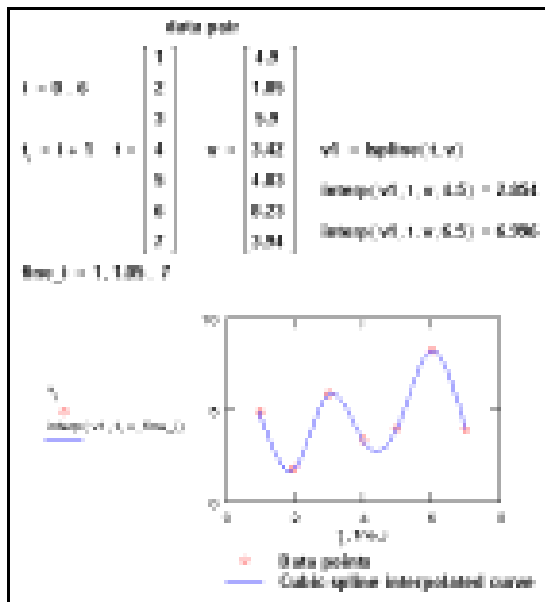
Returns the vector of coefficients of a cubic spline with linear ends. This vector becomes the first argument of the interp function.

Arguments

**vx**, **vy**real vectors of the same size; elements of **vx** must be in ascending order



## Example



## Comments

Cubic spline interpolation lets you pass a curve through a set of points so that the first and second derivatives of the curve are continuous across each point. This curve is assembled by taking three adjacent points and constructing a cubic polynomial passing through those points. These cubic polynomials are then strung together to form the completed curve.

To fit a cubic spline curve through a set of points:

1. Create the vectors  $\mathbf{vx}$  and  $\mathbf{vy}$  containing the  $x$  and  $y$  coordinates through which you want the cubic spline to pass. The elements of  $\mathbf{vx}$  should be in ascending order. (Although we use the names  $\mathbf{vx}$ ,  $\mathbf{vy}$ , and  $\mathbf{vs}$ , there is nothing special about these variable names; you can use whatever names you prefer.)
2. Generate the vector  $\mathbf{vs} := \text{lspline}(\mathbf{vx}, \mathbf{vy})$ . The vector  $\mathbf{vs}$  is a vector of intermediate results designed to be used with `interp`. It contains, among other things, the second derivatives for the spline curve used to fit the points in  $\mathbf{vx}$  and  $\mathbf{vy}$ .
3. To evaluate the cubic spline at an arbitrary point, say  $x0$ , evaluate `interp(vs, vx, vy, x0)` where  $\mathbf{vs}$ ,  $\mathbf{vx}$ , and  $\mathbf{vy}$  are the vectors described earlier.  
You could have accomplished the same task by evaluating:  
`interp(lspline(vx, vy), vx, vy, x0)`. As a practical matter, though, you'll probably be evaluating `interp` for many different points.

The call to `lspline` can be time-consuming and the result won't change from one point to the next, so it makes sense to do it just once and store the outcome in the  $\mathbf{vs}$  array.

Be sure that every element in the input arrays contains a data value. Because every element in a array must have a value, Mathcad assigns 0 to any elements you have not explicitly assigned.

In addition to `lspline`, Mathcad comes with three other cubic spline functions: `pspline`, `cspline`, and `bspline`. The `pspline` function generates a spline curve that approaches a parabola at the endpoints, while the `cspline` function generates a spline curve that can be fully cubic at the endpoints. `bspline`, on the other hand, allows the interpolation knots to be chosen by the user.

For `lspline`, the first three components of the output vector  $\mathbf{vs}$  are  $\mathbf{vs}_0=0$  (a code telling `interp` that  $\mathbf{vs}$  is the output of a spline function as opposed to a regression function),  $\mathbf{vs}_1=3$  (the index within  $\mathbf{vs}$  where the second derivative coefficients begin) and  $\mathbf{vs}_2=0$  (a code denoting `lspline`). The first three components for `pspline` and `cspline` are identical except  $\mathbf{vs}_2=1$  (the code denoting `pspline`) and  $\mathbf{vs}_2=2$  (the code denoting `cspline`), respectively.

## Two-dimensional Case

Syntax	<code>lspline(Mxy, Mz)</code>
Description	Returns the vector of coefficients of a two-dimensional cubic spline, constrained to be linear at region boundaries spanned by $\mathbf{Mxy}$ . This vector becomes the first argument of the <code>interp</code> function.
Arguments	
$\mathbf{Mxy}$	$n \times 2$ matrix whose elements, $Mxy_{i,0}$ and $Mxy_{i,1}$ , specify the $x$ - and $y$ -coordinates along the <i>diagonal</i> of a rectangular grid. This matrix plays exactly the same role as $\mathbf{vx}$ in the one-dimensional case described earlier. Since these points describe a diagonal, the elements in each column of $\mathbf{Mxy}$ must be in ascending order ( $Mxy_{i,k} < Mxy_{j,k}$ whenever $i < j$ ).
$\mathbf{Mz}$	$n \times n$ matrix whose $ij$ th element is the $z$ -coordinate corresponding to the point $x = Mxy_{i,0}$ and $y = Mxy_{j,1}$ . $\mathbf{Mz}$ plays exactly the same role as $\mathbf{vy}$ does in the one-dimensional case above.
Comments	Mathcad handles two-dimensional cubic spline interpolation in much the same way as the one-dimensional case. Instead of passing a curve through a set of points so that the first and second derivatives of the curve are continuous across each point, Mathcad passes a surface through a grid of points. This surface corresponds to a cubic polynomial in $x$ and $y$ in which the first and second partial derivatives are continuous in the corresponding direction across each grid point.  The first step in two-dimensional spline interpolation is exactly the same as that in the one-dimensional case: specify the points through which the surface is to pass. The procedure, however, is more complicated because you now have to specify a grid of points.

To perform two-dimensional spline interpolation, follow these steps:

1. Create  $\mathbf{Mxy}$ .
2. Create  $\mathbf{Mz}$ .
3. Generate the vector  $\mathbf{vs} := \text{lspline}(\mathbf{Mxy}, \mathbf{Mz})$ . The vector  $\mathbf{vs}$  is a vector of intermediate results designed to be used with `interp`.  
To evaluate the cubic spline at an arbitrary point, say  $(x_0, y_0)$ , evaluate

$$\text{interp}\left(\mathbf{vs}, \mathbf{Mxy}, \mathbf{Mz}, \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right), \text{ where } \mathbf{vs}, \mathbf{Mxy}, \text{ and } \mathbf{Mz} \text{ are as described earlier.}$$

The result is the value of the interpolating surface corresponding to the arbitrary point  $(x_0, y_0)$ . You could have accomplished exactly the same task by evaluating:

$$\text{interp}\left(\text{lspline}(\mathbf{Mxy}, \mathbf{Mz}), \mathbf{Mxy}, \mathbf{Mz}, \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right).$$