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 do it just once and store the outcome in the **vs** array.

In addition to lspline, Mathcad comes with two other cubic spline functions for the two-dimensional case: pspline and cspline. The pspline function generates a spline curve that approaches a second degree polynomial in x and y along the edges. The cspline function generates a spline curve that approaches a third degree polynomial in x and y along the edges.

Algorithm Tridiagonal system solving (Press *et al.*, 1992; Lorczak)

lu

(Professional) Vector and Matrix

Syntax lu(**M**)

Description Returns an $n \times (3 \cdot n)$ matrix whose first n columns contain an $n \times n$ permutation matrix **P**, whose next n columns contain an $n \times n$ lower triangular matrix **L**, and whose remaining n columns contain an $n \times n$ upper triangular matrix **U**. These matrices satisfy the equation $\mathbf{P} \cdot \mathbf{M} = \mathbf{L} \cdot \mathbf{U}$.

Arguments
M real or complex $n \times n$ matrix

Comments This is known as the LU decompostion (or factorization) of the matrix **M**, permuted by **P**.

Algorithm Crout's method with partial picoting (Press *et al.*, 1992; Golub and Van Loan, 1989)

matrix

Vector and Matrix

Syntax matrix(m, n, f)

Description Creates a matrix in which the ijth element is the value $f(i, j)$, where $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$.

Arguments
m, n integers
f scalar-valued function

max

Vector and Matrix

Syntax max(**A**)

Description Returns the largest element in **A**. If **A** is complex, returns max(Re(**A**)) + i·max(Im(**A**)).

Arguments
A real or complex $m \times n$ matrix or vector

See also min

Maximize

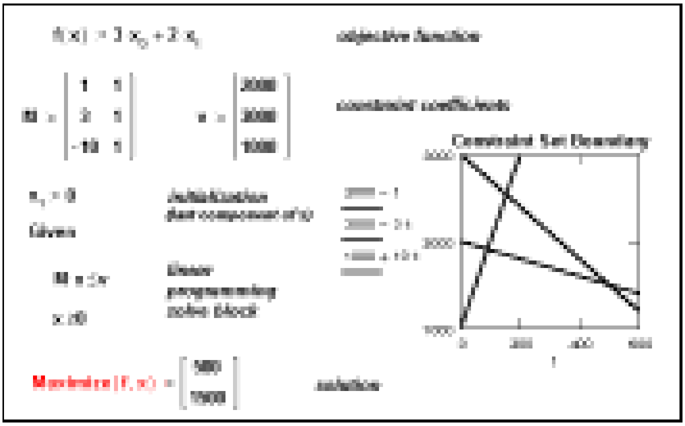
Syntax Maximize(*f*, *var1*, *var2*,...)

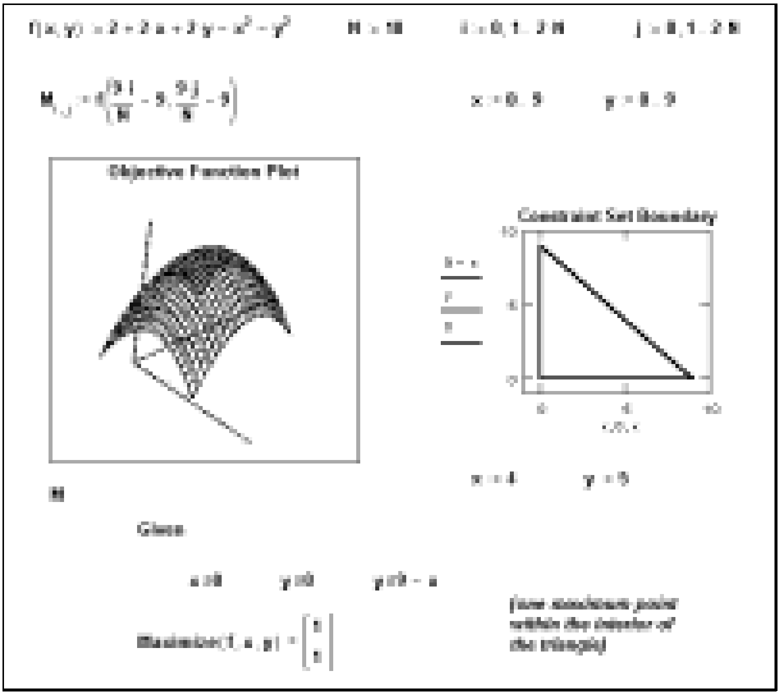
Description Returns values of *var1*, *var2*,... which solve a prescribed system of equations, subject to prescribed inequalities, and which make the function *f* take on its largest value. The number of arguments matches the number of unknowns, plus one. Output is a scalar if only one unknown; otherwise it is a vector of answers.

Arguments

f	real-valued objective function
var1, *var2*, ...	real or complex variables; *var1*, *var2*,.. must be assigned guess values before using Maximize

Examples





Comments There are five steps to solving a maximization problem:

1. Define the objective function *f*.
2. Provide an initial guess for all the unknowns you intend to solve for. This gives Mathcad a place to start searching for solutions.
3. 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 be sure you don't type it while in a text region.
4. Now type the equations and inequalities in any order below the word given. Use [**Ctrl**]**=** to type "=."
5. Finally, type the Maximize function with *f* and your list of unknowns. You can't put numerical values in the list of unknowns; for example, Maximize(*f*, 2) isn't permitted. Like given, you can type maximize or Maximize in any style.

The Maximize function returns values as follows:

- If there is one unknown, Maximize returns a scalar value that optimizes *f*.
- If there is more than one unknown, Maximize returns a vector of answers; for example, Maximize(*f*, *var1*, *var2*) returns a vector containing values of *var1* and *var2* that satisfy the constraints and optimize *f*.

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

By default, Mathcad examines your objective function and the constraints, and solves using an appropriate method. In Mathcad Professional, if you want to try different algorithms for testing and comparison, you can choose options from the context menu associated with Maximize (available via right mouse click), which include:

- AutoSelect – chooses an appropriate algorithm for you
- 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 more control for you to try 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, e.g., 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, which is discussed further in connection with the Find function. Since Maximize can be used without constraints, the value of CTOL will sometimes be irrelevant. Its default value is 0.

For an unconstrained maximization problem, the word Given and constraints are unnecessary.

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 Find for more details about solve blocks; Minerr, Minimize

---

mean

Statistics

Syntax mean(**A**)

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

Arguments

A real or complex $m \times n$ matrix or vector

See also gmean, hmean, median, mode

median

Syntax median(**A**)

Description Returns the median of the elements of **A**. The median is the value above and below which there are an equal number of values. If **A** has an even number of elements, median is the arithmetic mean of the two central values.

Arguments

A real or complex $m \times n$ matrix or vector

See also gmean, hmean, mean, mode

medsmooth

Syntax medsmooth(**vy**, *n*)

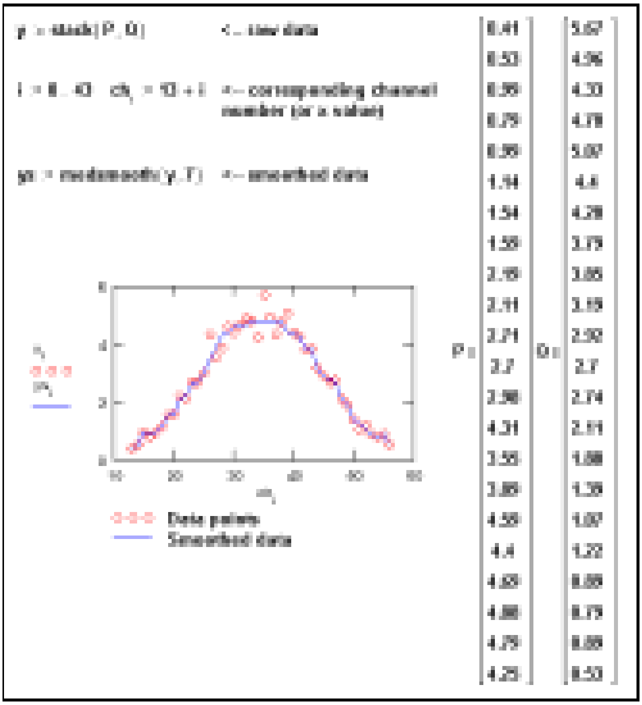
Description Creates a new vector, of the same size as **vy**, by smoothing **vy** with running medians.

Arguments

vy real vector

n odd integer, $n > 0$, the size of smoothing window

Example



Comments

Smoothing involves taking a set of *y* (and possibly *x*) values and returning a new set of *y* values that is smoother than the original set. Unlike the interpolation functions cspline, lspline, or pspline or regression functions regress or loess, smoothing results in a new set of *y* values, not a function that can be evaluated between the data points you specify. If you are interested in *y* values *between* the *y* values you specify, use an interpolation or regression function.

Whenever you use vectors in any of the functions described in this section, be sure that every element in the vector contains a data value. Because every element in a vector must have a value, Mathcad assigns 0 to any elements you have not explicitly assigned.

The medsmooth function is the most robust of Mathcad's three smoothing functions because it is least likely to be affected by spurious data points. This function uses a running median smoother, computes the residuals, smooths the residuals the same way, and adds these two smoothed vectors together.

medsmooth performs these steps:

1. Finds the running medians of the input vector **vy**. We'll call this $\mathbf{vy'}$. The *i*th element is given by: $vy'_i = \text{median}(vy_{i-(n-1/2)}, \ldots, vy_i, \ldots, vy_{i+(n-1/2)})$.
2. Evaluates the residuals: $\mathbf{vr} = \mathbf{vy} - \mathbf{vy'}$.
3. Smooths the residual vector, **vr**, using the same procedure described in step 1, to create a smoothed residual vector, $\mathbf{vr'}$.
4. Returns the sum of these two smoothed vectors: $\text{medsmooth}(\mathbf{vy}, n) = \mathbf{vy'} + \mathbf{vr'}$.

medsmooth will leave the first and last $(n-1)/2$ points unchanged. In practice, the length of the smoothing window, *n*, should be small compared to the length of the data set.

Algorithm

Moving window median method (Lorczak)

See also

ksmooth and supsmooth (alternative smoothing functions available in Mathcad Professional only)

mhyper

(Professional) Special

Syntax

mhyper(*a, b, x*)

Description

Returns the value of the confluent hypergeometric function, ${}_1F_1(a;b;x)$ or $\boldsymbol{M}(a;b;x)$.

Arguments

a, b, x real numbers

Comments

The confluent hypergeometric function is a solution of the differential equation:

$$x \cdot \frac{d^2}{dx^2}y + (b-x) \cdot \frac{d}{dx}y - a \cdot y = 0$$ and is also known as the Kummer function.

Many functions are special cases of this, e.g., elementary ones like

$\exp(x) = \text{mhyper}(1, 1, x)$ $\exp(x) \cdot \sinh(x) = x \cdot \text{mhyper}(1, 2, 2 \cdot x)$

and more complicated ones like Hermite functions.

Algorithm

Series expansion, asymptotic approximations (Abramowitz and Stegun, 1972)

min

Vector and Matrix

Syntax min(**A**)

Description Returns the smallest element in **A**. If **A** is complex, returns min(Re(**A**)) + $i\cdot$min(Im(**A**)).

Arguments

A real or complex $m \times n$ matrix or vector

See also max

Minerr

Solving

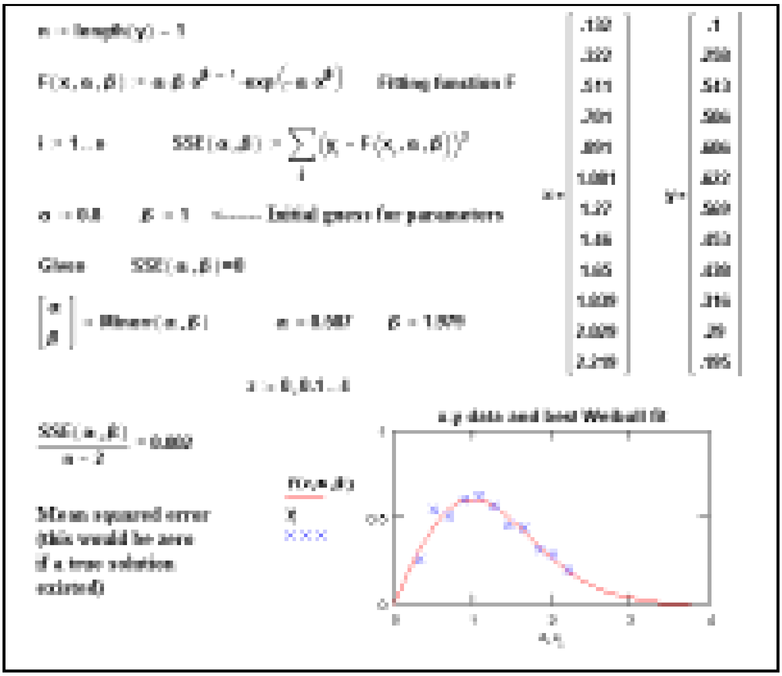
Syntax Minerr(*var1*, *var2*,...)

Description Returns values of *var1*, *var2*, ... which come closest to solving 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 Minerr.

Example



Comments The Minerr function is very similar to Find and uses exactly the same algorithm. The difference is that even if a system has no solutions, Minerr will attempt to find values which come closest to solving the system. The Find function, on the other hand, will return an error message indicating that it could not find a solution. You use Minerr exactly the way you use Find.

Like Find, type the Minerr function with your list of unknowns. You can't put numerical values in the list of unknowns; e.g., in the example above, Minerr(0.8, 1) isn't permitted. Like Find, you can type Minerr or minerr in any style.

Minerr usually returns an answer that minimizes the errors in the constraints. However, Minerr cannot verify that its answers represent an absolute minimum for the errors in the constraints.

If you use Minerr in a solve block, you should always include additional checks on the reasonableness of the results. The built-in variable ERR gives the size of the error vector for the approximate solution. There is no built-in variable for determining the size of the error for individual solutions to the unknowns.

Minerr is particularly useful for solving certain nonlinear least-squares problems. In the example, Minerr is used to obtain the unknown parameters in a Weibull distribution. The function genfit is also useful for solving nonlinear least-squares problems.

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

- AutoSelect – chooses an appropriate algorithm for you
- Linear option – not available for Minerr (since the objective function is quadratic, hence the problem can never be linear)
- 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 more control for you to try 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, e.g., 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

Find for more details about solve blocks; Maximize, Minimize

Minimize

Syntax Minimize(*f*, *var1*, *var2*,...)

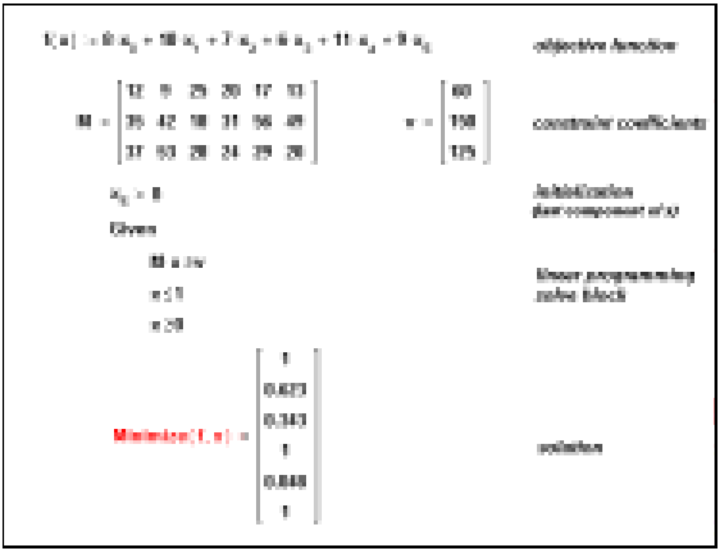
Description Returns values of *var1*, *var2*,... which solve a prescribed system of equations, subject to prescribed inequalities, and which make the function *f* take on its smallest value. The number of arguments matches the number of unknowns, plus one. Output is a scalar if only one unknown; otherwise it is a vector of answers.

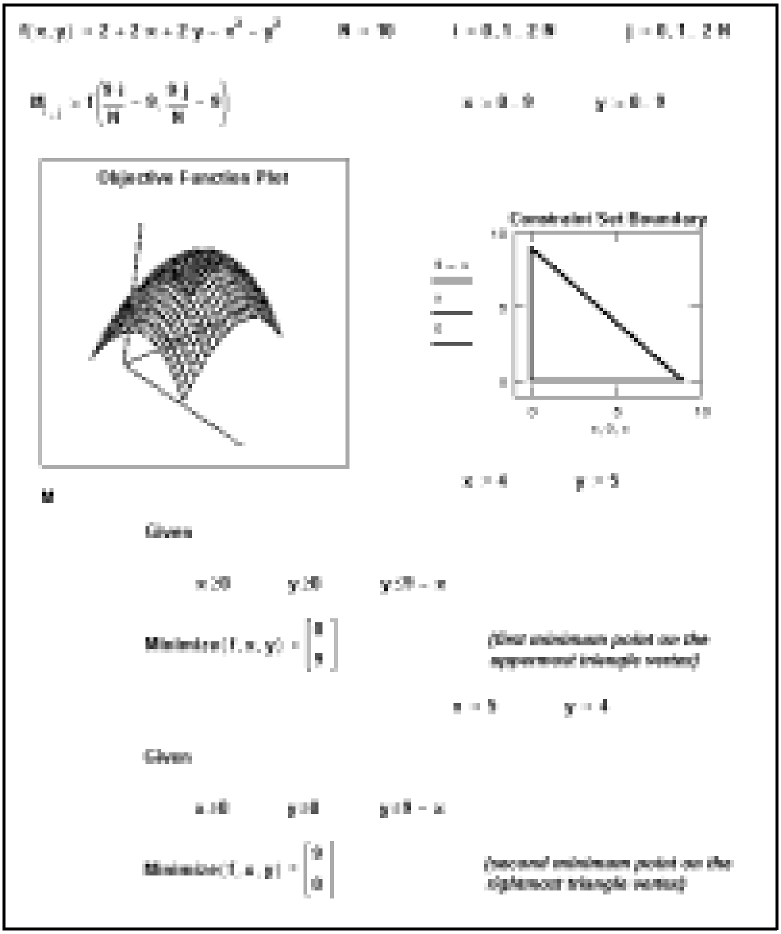
Arguments

f real-valued function

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

Examples





Comments

There are five steps to solving a minimization problem:

1. Define the objective function *f*.
2. Provide an initial guess for all the unknowns you intend to solve for. This gives Mathcad a place to start searching for solutions.
3. 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 be sure you don't type it while in a text region.
4. Now type the equations and inequalities in any order below the word given. Use [**Ctrl**]**=** to type "=."
5. Finally, type the Minimize function with *f* and your list of unknowns. You can't put numerical values in the list of unknowns; for example, Minimize(*f*, 2) isn't permitted. Like given, you can type minimize or Minimize in any style.

The Minimize function returns values as follows:

- If there is one unknown, Minimize returns a scalar value that optimizes *f*.
- If there is more than one unknown, Minimize returns a vector of answers; for example, Minimize(*f*, *var1*, *var2*) returns a vector containing values of *var1* and *var2* that satisfy the constraints and optimize *f*.

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

By default, Mathcad examines your objective function and the constraints, and solves using an appropriate method. In Mathcad Professional, if you want to try different algorithms for testing and comparison, you can choose options from the context menu associated with Minimize (available via right mouse click), which include:

- AutoSelect – chooses an appropriate algorithm for you
- 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 more control for you to try 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, e.g., 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, which is discussed further in connection with the Find function. Since Minimize can be used without constraints, the value of CTOL will sometimes be irrelevant. Its default value is 0.

For an unconstrained minimization problem, the word Given and constraints are unnecessary.

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

Find for more details about solve blocks; Maximize, Minerr

mod — Number Theory/Combinatorics

Syntax

mod(*n*, *k*)

Description

Returns the remainder of *n* when divided by *k*. The result has the same sign as *n*.

Arguments

n, *k* — integers, $k \neq 0$

mode

Statistics

Syntax mode(**A**)

Description Returns the value in **A** that occurs most often.

Arguments

A real or complex $m \times n$ matrix or vector

See also gmean, hmean, mean, median

multigrid

(Professional) Differential Equation Solving

Syntax multigrid(**M**, *ncycle*)

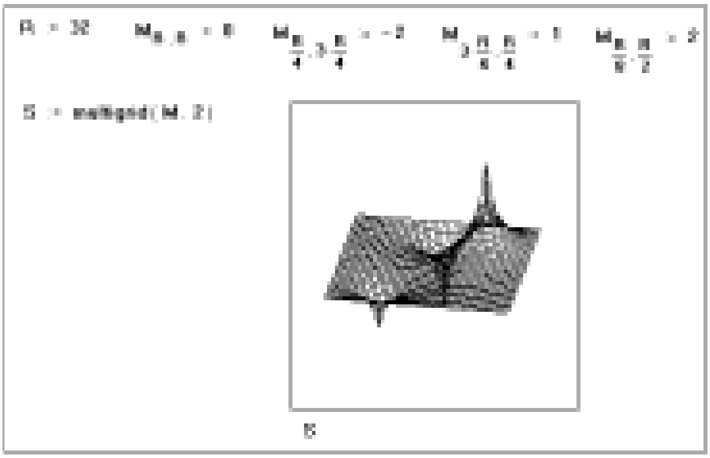
Description Solves the Poisson partial differential equation over a planar square region. The $n \times n$ matrix **M** gives source function values, where $n - 1$ is a power of 2 and zero boundary conditions on all four edges are assumed. multigrid uses a different algorithm and is faster than relax, which is more general.

Arguments

M $(1 + 2^k) \times (1 + 2^k)$ real square matrix containing the source term at each point in the region in which the solution is sought (for example, the right-hand side of equation below)

ncycle positive integer specifying number of cycles at each level of the multigrid iteration; a value of 2 generally gives a good approximation of the solution

Example



Comments Two partial differential equations that arise often in the analysis of physical systems are Poisson's equation:

$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$ and its homogeneous form, Laplace's equation.

Mathcad has two functions for solving these equations over a square region, assuming the values taken by the unknown function $u(x, y)$ on all four sides of the boundary are known. The most general solver is the relax function. In the special case where $u(x, y)$ is known to be zero on all four sides of the boundary, you can use the multigrid function instead. This function often solves the problem faster than relax. If the boundary condition is the same on all four sides, you can simply transform the equation to an equivalent one in which the value is zero on all four sides.

The multigrid function returns a square matrix in which:

- an element's location in the matrix corresponds to its location within the square region, and
- its value approximates the value of the solution at that point.

Algorithm — Full multigrid algorithm (Press *et al.*, 1992)

See also — relax

norm1

(Professional) — Vector and Matrix

Syntax — norm1(**M**)

Description — Returns the L_1 norm of the matrix **M**.

Arguments

M — real or complex square matrix

norm2

(Professional) — Vector and Matrix

Syntax — norm2(**M**)

Description — Returns the L_2 norm of the matrix **M**.

Arguments

M — real or complex square matrix

Algorithm — Singular value computation (Wilkinson and Reinsch, 1971)

norme

(Professional) — Vector and Matrix

Syntax — norme(**M**)

Description — Returns the Euclidean norm of the matrix **M**.

Arguments

M — real or complex square matrix

normi

(Professional) Vector and Matrix

Syntax
normi(**M**)

Description
Returns the infinity norm of the matrix **M**.

Arguments
M real or complex square matrix

num2str

(Professional) String

Syntax
num2str(*z*)

Description
Returns the string whose characters correspond to the decimal value of *z*.

Arguments
z real or complex number

See also
str2num

pbeta

Probability Distribution

Syntax
pbeta(*x*, *s1*, *s2*)

Description
Returns the cumulative beta distribution with shape parameters *s1* and *s2*.

Arguments
x real number, $0 < x < 1$
s1, *s2* real shape parameters, $s1 > 0$, $s2 > 0$

Algorithm
Continued fraction expansion (Abramowitz and Stegun, 1972)

pbinom

Probability Distribution

Syntax
pbinom(*k*, *n*, *p*)

Description
Returns $\Pr(X \le k)$ when the random variable *X* has the binomial distribution with parameters *n* and *p*.

Arguments
k, *n* integers, $0 \le k \le n$
p real numbers, $0 \le p \le 1$

Algorithm
Continued fraction expansion (Abramowitz and Stegun, 1972)

pcauchy

Probability Distribution

Syntax: pcauchy(*x, l, s*)

Description: Returns the cumulative Cauchy distribution.

Arguments

x real number

l real location parameter

s real scale parameter, $s > 0$

pchisq

Probability Distribution

Syntax: pchisq(*x, d*)

Description: Returns the cumulative chi-squared distribution.

Arguments

x real number, $x \ge 0$

d integer degrees of freedom, $d > 0$

Algorithm: Continued fraction and asymptotic expansions (Abramowitz and Stegun, 1972)

permut

Number Theory/Combinatorics

Syntax: permut(*n, k*)

Description: Returns the number of ways of ordering *n* distinct objects taken *k* at a time.

Arguments

n, k integers, $0 \le k \le n$

Comments: Each such ordered arrangement is known as a permutation. The number of permutations is

$$P_k^n = \frac{n!}{(n-k)!}$$

See also: combin

pexp

Probability Distribution

Syntax: pexp(*x, r*)

Description: Returns the cumulative exponential distribution.

Arguments

x real number, $x \ge 0$

r real rate, $r > 0$

pF

Probability Distribution

Syntax pF(*x, d1, d2*)

Description Returns the cumulative F distribution.

Arguments

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

Algorithm Continued fraction expansion (Abramowitz and Stegun, 1972)

pgamma

Probability Distribution

Syntax pgamma(*x, s*)

Description Returns the cumulative gamma distribution.

Arguments

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

Algorithm Continued fraction and asymptotic expansion (Abramowitz and Stegun, 1972)

pgeom

Probability Distribution

Syntax pgeom(*k, p*)

Description Returns $\Pr(X \leq k)$ when the random variable X has the geometric distribution with parameter p.

Arguments

- *k* — integer, $k \geq 0$
- *p* — real number, $0 < p \leq 1$

phypergeom

Probability Distribution

Syntax phypergeom(*m, a, b, n*)

Description Returns $\Pr(X \leq m)$ when the random variable X has the hypergeometric distribution with parameters a, b and n.

Arguments

- *m, a, b, n* — integers, $0 \leq m \leq a$, $0 \leq n - m \leq b$, $0 \leq n \leq a + b$

plnorm

Probability Distribution

Syntax: plnorm(x, μ, σ)

Description: Returns the cumulative lognormal distribution.

Arguments

- x — real number, $x \geq 0$
- μ — real logmean
- σ — real logdeviation, $\sigma > 0$

plogis

Probability Distribution

Syntax: plogis(x, l, s)

Description: Returns the cumulative logistic distribution.

Arguments

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

pnbinom

Probability Distribution

Syntax: pnbinom(k, n, p)

Description: Returns the cumulative negative binomial distribution with parameters n and p.

Arguments

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

Algorithm: Continued fraction expansion (Abramowitz and Stegun, 1972)

pnorm

Probability Distribution

Syntax: pnorm(x, μ, σ)

Description: Returns the cumulative normal distribution.

Arguments

- x — real number
- μ — real mean
- σ — real standard deviation, $\sigma > 0$

polyroots

Solving

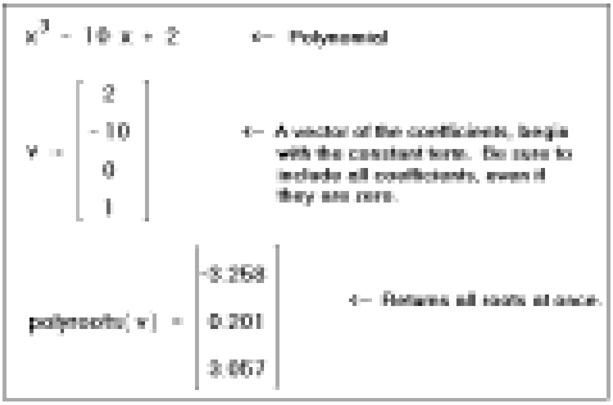
Syntax polyroots(**v**)

Description Returns the roots of an *n*th degree polynomial whose coefficients are in **v**. Output is a vector of length *n*.

Arguments

v real or complex vector of length $n + 1$

Example



Comments To find the roots of an expression having the form: $v_n x^n + \ldots + v_2 x^2 + v_1 x + v_0$

you can use the polyroots function rather than the root function. Unlike root, polyroots does not require a guess value. Moreover, polyroots returns all roots at once, whether real or complex.

The polyroots function can solve only one polynomial equation in one unknown. See root for a more general equation solver. To solve several equations simultaneously, use solve blocks (Find or Minerr). To solve an equation symbolically – that is, to find an exact numerical answer in terms of elementary functions – choose **Solve for Variable** from the **Symbolics** menu or use the solve keyword.

Algorithm Laguerre with deflation and polishing (Lorczak)

See also See coeff keyword for a way to create the coefficient vector **v** immediately, given a polynomial.

ppois

Probability Distribution

Syntax ppois(k, λ)

Description Returns the cumulative Poisson distribution.

Arguments

k integer, $k \geq 0$

λ real mean, $\lambda > 0$

Algorithm Continued fraction and asymptotic expansions (Abramowitz and Stegun, 1972)

predict *(Professional)* Interpolation and Prediction

Syntax predict(**v**, *m*, *n*)

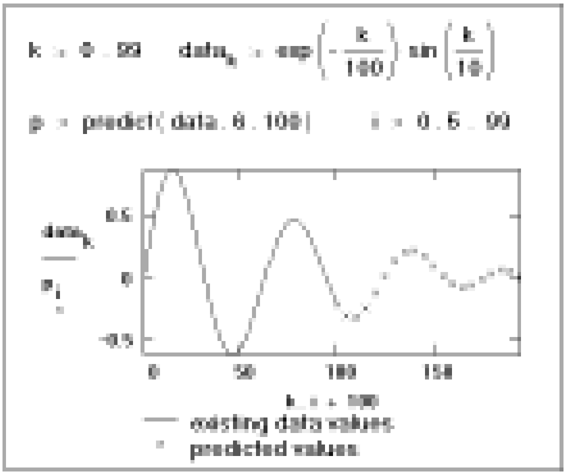
Description Returns *n* predicted values based on *m* consecutive values from the data vector **v**. Elements in **v** should represent samples taken at equal intervals.

Arguments

v real vector

m, n integers, $m > 0$, $n > 0$

Example



Comments Interpolation functions such as cspline, lspline, or pspline , coupled with interp, allow you to find data points lying between existing data points. However, you may need to find data points that lie beyond your existing ones. Mathcad provides the function predict which uses some of your existing data to predict data points lying beyond existing ones. This function uses a linear prediction algorithm which is useful when your data is smooth and oscillatory, although not necessarily periodic. This algorithm can be seen as a kind of extrapolation method but should not be confused with linear or polynomial extrapolation.

The predict function uses the last *m* of the original data values to compute prediction coefficients. After it has these coefficients, it uses the last *m* points to predict the coordinates of the $(m+1)^{st}$ point, in effect creating a moving window that is *m* points wide.

Algorithm Burg’s method (Press *et al.*, 1992)

pspline

One-dimensional Case

Syntax pspline(**vx**, **vy**)

Description Returns the vector of coefficients of a cubic spline with parabolic 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

Two-dimensional Case

Syntax pspline(**Mxy**, **Mz**)

Description Returns the vector of coefficients of a two-dimensional cubic spline, constrained to be parabolic at region boundaries spanned by **Mxy**. This vector becomes the first argument of the interp function.

Arguments

Mxy $n \times 2$ matrix whose elements, $Mxy_{i,0}$ and $Mxy_{i,1}$, specify the *x*- and *y*-coordinates along the *diagonal* of a rectangular grid. This matrix plays exactly the same role as **vx** in the one-dimensional case described earlier. Since these points describe a diagonal, the elements in each column of **Mxy** must be in ascending order ($Mxy_{i,k} < Mxy_{j,k}$ whenever $i < j$).

Mz $n \times n$ matrix whose *ij*th element is the *z*-coordinate corresponding to the point $x = Mxy_{i,0}$ and $y = Mxy_{j,1}$. **Mz** plays exactly the same role as **vy** in the one-dimensional case above.

Algorithm Tridiagonal system solving (Press *et al.*, 1992, Lorczak)

See also lspline for more details

pt

Probability Distribution

Syntax pt(*x, d*)

Description Returns the cumulative Student's *t* distribution.

Arguments

x real number, $x \geq 0$

d integer degrees of freedom, $d > 0$

Algorithm Continued fraction expansion (Abramowitz and Stegun, 1972)

punif

Probability Distribution

Syntax punif(x, a, b)

Description Returns the cumulative uniform distribution.

Arguments

x real number

a, b real numbers, $a < b$

pweibull

Probability Distribution

Syntax pweibull(x, s)

Description Returns the cumulative Weibull distribution.

Arguments

x real number, $x \geq 0$

s real shape parameter, $s > 0$

qbeta

Probability Distribution

Syntax qbeta($p, s1, s2$)

Description Returns the inverse beta distribution with shape parameters *s1* and *s2*.

Arguments

p real number, $0 \leq p \leq 1$

s1, s2 real shape parameters, $s_1 > 0, s_2 > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)

qbinom

Probability Distribution

Syntax qbinom(p, n, q)

Description Returns the inverse binomial distribution function, that is, the smallest integer *k* so that pbinom(k, n, q) $\geq p$.

Arguments

n integer, $n > 0$

p, q real numbers, $0 \leq p \leq 1$, $0 \leq q \leq 1$

Comments *k* is approximately the integer for which $\Pr(X \leq k) = p$, when the random variable *X* has the binomial distribution with parameters *n* and *q*. This is the meaning of “inverse” binomial distribution function.

Algorithm Discrete bisection method (Press *et al.*, 1992)

qcauchy

Probability Distribution

Syntax qcauchy(p, l, s)

Description Returns the inverse Cauchy distribution function.

Arguments

p real number, $0 < p < 1$

l real location parameter

s real scale parameter, $s > 0$

qchisq

Probability Distribution

Syntax qchisq(p, d)

Description Returns the inverse chi-squared distribution.

Arguments

p real number, $0 \le p < 1$

d integer degrees of freedom, $d > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)
Rational function approximations (Abramowitz and Stegun, 1972)

qexp

Probability Distribution

Syntax qexp(p, r)

Description Returns the inverse exponential distribution.

Arguments

p real number, $0 \le p < 1$

r real rate, $r > 0$

qF

Probability Distribution

Syntax qF($p, d1, d2$)

Description Returns the inverse F distribution.

Arguments

p real number, $0 \le p < 1$

$d1, d2$ integer degrees of freedom, $d1 > 0$, $d2 > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)

qgamma

Probability Distribution

Syntax qgamma(p, s)

Description Returns the inverse gamma distribution.

Arguments

p real number, $0 \le p < 1$

s real shape parameter, $s > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)
Rational function approximations (Abramowitz and Stegun, 1972)

qgeom

Probability Distribution

Syntax qgeom(p, q)

Description Returns the inverse geometric distribution, that is , the smallest integer k so that pgeom(k, q) $\ge p$.

Arguments

p, q real numbers, $0 < p < 1$, $0 < q < 1$

Comments k is approximately the integer for which $\Pr(X \le k) = p$, when the random variable X has the geometric distribution with parameter q. This is the meaning of “inverse” geometric distribution function.

qhypergeom

Probability Distribution

Syntax qhypergeom(p, a, b, n)

Description Returns the inverse hypergeometric distribution, that is, the smallest integer k so that phypergeom(k, a, b, n) $\ge p$.

Arguments

p real number, $0 \le p < 1$

a, b, n integers, $0 \le a$, $0 \le b$, $0 \le n \le a + b$

Comments k is approximately the integer for which $\Pr(X \le k) = p$, when the random variable X has the hypergeometric distribution with parameters a, b and n. This is the meaning of “inverse” hypergeometric distribution function.

Algorithm Discrete bisection method (Press *et al.*, 1992)

qlnorm

Probability Distribution

Syntax qlnorm(p, μ, σ)

Description Returns the inverse log normal distribution.

Arguments

p	real number; $0 \le p < 1$
μ	logmean
σ	logdeviation; $\sigma > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)

qlogis

Probability Distribution

Syntax qlogis(p, l, s)

Description Returns the inverse logistic distribution.

Arguments

p	real number, $0 < p < 1$
l	real location parameter
s	real scale parameter, $s > 0$

qnbinom

Probability Distribution

Syntax qnbinom(p, n, q)

Description Returns the inverse negative binomial distribution function, that is, the smallest integer k so that pnbinom(k, n, q) $\ge p$.

Arguments

n	integer, $n > 0$
p, q	real numbers, $0 < p < 1$, $0 < q < 1$

Comments k is approximately the integer for which $\Pr(X \le k) = p$, when the random variable X has the negative binomial distribution with parameters n and q. This is the meaning of “inverse” negative binomial distribution function.

Algorithm Discrete bisection method (Press *et al.*, 1992)

qnorm

Probability Distribution

Syntax qnorm(p, μ, σ)

Description Returns the inverse normal distribution.

Arguments

p real number, $0 < p < 1$

μ real mean

σ standard deviation, $\sigma > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)

qpois

Probability Distribution

Syntax qpois(p, λ)

Description Returns the inverse Poisson distribution, that is, the smallest integer k so that ppois(k, λ) $\geq p$.

Arguments

p real number, $0 \leq p \leq 1$

λ real mean, $\lambda > 0$

Comments k is approximately the integer for which $\Pr(X \leq k\ ) = p$, when the random variable X has the Poisson distribution with parameter λ. This is the meaning of "inverse" Poisson distribution function.

Algorithm Discrete bisection method (Press *et al.*, 1992)

qr

(Professional)

Vector and Matrix

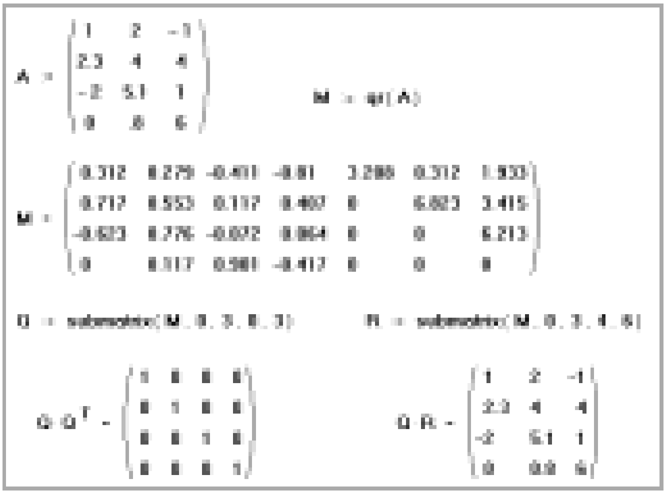
Syntax qr(**A**)

Description Returns an $m \times (m + n)$ matrix whose first m columns contain the $m \times m$ orthonormal matrix **Q**, and whose remaining n columns contain the $m \times n$ upper triangular matrix **R**. These satisfy the matrix equation $\mathbf{A} = \mathbf{Q} \cdot \mathbf{R}$.

Arguments

A real $m \times n$ matrix

Example



---

qt

Probability Distribution

Syntax qt(p, d)

Description Returns the inverse Student's t distribution.

Arguments

- p real number, $0 < p < 1$
- d integer degrees of freedom, $d > 0$

Algorithm Root finding (bisection and secant methods) (Press *et al.*, 1992)

---

qunif

Probability Distribution

Syntax qunif(p, a, b)

Description Returns the inverse uniform distribution.

Arguments

- p real number, $0 \le p \le 1$
- a, b real numbers, $a < b$

---

qweibull

Probability Distribution

Syntax qweibull(p, s)

Description Returns the inverse Weibull distribution.

Arguments

- p real number, $0 < p < 1$
- s real shape parameter, $s > 0$