

Zero factor or numerator

For efficiency reasons, Mathcad always assumes that for any expression x :

$$0 \cdot x = 0$$

and

$$0 / x = 0$$

Presented with a calculation of this type, Mathcad will not even evaluate x . This has the following consequences:

- Mathcad will instantly compute a result of zero for these expressions, even if x requires a time-consuming calculation like an integral or a summation.
- If computing x would result in an error, Mathcad will return zero without detecting the error. In some cases this is desirable; in others, it isn't.
- Mathcad evaluates $0/0$ as zero, not as an error.

Integrals

Mathcad computes definite integrals numerically using a Romberg algorithm. The Romberg method accelerates the convergence of a sequence of simple trapezoidal or midpoint approximations to the integral by extrapolating the sequence of estimates and the corresponding subinterval widths.

This section describes how Mathcad computes the integral of $f(x)$ from a to b . In this description, $estimate_0$, $estimate_1$, and so on represent Mathcad's internal approximations to the answer. Only the last estimate, the one that passes Mathcad's tolerance test, is available to the user.

To compute the integral of $f(x)$ from a to b , Mathcad follows these steps:

- Calculate an estimate for the integral using the trapezoidal rule. The first trapezoidal estimate is:

$$estimate_0 = \frac{f(a) + f(b)}{2} (b - a)$$

The second trapezoidal estimate doubles the number of subintervals, using two subintervals each of width $b - a/2$. To begin the algorithm, compute the first three estimates, subdividing the interval into one, two, and four subintervals, respectively.

- Fit a polynomial to the sequence of trapezoidal estimates computed so far and the corresponding subinterval widths. Extrapolate this polynomial to width zero to produce a Romberg estimate for the integral.

- The error for a particular estimate of the derivative's value is given by:

$$err_{i,j} = \max \left(\left| e_{i,j} - e_{i-1,j} \right|, \left| e_{i,j} - e_{i-1,j-1} \right| \right)$$

- Let $errmin$ be the smallest of the $err_{i,j}$ and let der be the estimate of the derivative's value corresponding to $errmin$. Then if:

$$errmin < \max(E, E \cdot der),$$

where E is a small number depending on the order. Mathcad returns der as the value of the derivative. Otherwise, Mathcad marks the derivative with the message "not converging."

Note that any numerical differentiation algorithm can lead to computational problems associated with round-off and truncation. This is particularly true of function values lying near a discontinuity or singularity in the function. With Ridder's algorithm, you can expect the first derivative to be accurate to within 7 or 8 significant digits, provided that the value at which you evaluate the derivative is not too close to a singularity of the function. The accuracy of this algorithm tends to decrease by one significant digit for each increase in the order of the derivative.

The root function

Mathcad's *root* function solves for zeros using the secant method.

This section describes how Mathcad computes $\text{root}(f(x), x)$. In this description, $estimate_0$, $estimate_1$, and so on represent Mathcad's internal approximations to the answer. Only the last estimate, the one that passes Mathcad's tolerance test, is available to the user.

To compute a root of $f(x)$ using the initial guess x , Mathcad follows these steps:

- If $|f(x)| < \text{TOL}$, then x is already a root, so return x as the value of the *root* function.
- Set $n = 1$. If $x \neq 0$, set $h = \text{TOL} \cdot x$, otherwise, set $h = \text{TOL}$. Make the following initial definitions:

$$estimate_0 = x$$

$$estimate_1 = x + h$$

- Increase n by one. Then compute the straight line connecting the points $(estimate_{n-2}, f(estimate_{n-2}))$ and $(estimate_{n-1}, f(estimate_{n-1}))$. Set x_n to the point where this straight line crosses the zero axis.
- If $|f(estimate_n)| < \text{TOL}$, return $estimate_n$ as the value of the *root* function. Otherwise, go back to step 3.

There is a limit on the number of calculations that Mathcad will perform in search of an answer for the *root* function. If it exceeds this limit without returning an answer, the *root* function is marked with the error “not converging.”

Solve blocks

Mathcad’s solve areas use the iterative Levenberg-Marquardt method to solve for several constraints simultaneously. Mathcad’s method is taken from the public-domain MINPACK algorithms developed and published by the Argonne National Laboratory in Argonne, Illinois. For more information, see the *User’s Guide to Minpack I*, by Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstom, Argonne National Laboratory publication ANL-80-74, 1980.

The error vector

The MINPACK algorithm attempts to find the zeros of, or at worst minimize the sum of squares of, the value of a vector of functions relative to the values of certain variables. The vector of functions Mathcad uses is an *error vector* whose elements represent the errors in the constraints. The errors are defined as follows:

- For equality constraints (constraints using “=”):

$$error = left_side - right_side$$

- For inequality constraints defined with <, >, ≤, or ≥:

$$error = 0$$

if inequality is true, otherwise

$$error = left_side - right_side$$

Mathcad treats the Levenberg-Marquardt algorithm as an algorithm on real variables. When you solve for a complex variable, Mathcad treats the real and imaginary parts as separate variables in the algorithm. When you solve an equality constraint, Mathcad creates two real constraints for the algorithm, one for the real part and one for the imaginary part.

Steps in the Levenberg-Marquardt method

The Levenberg-Marquardt method is a quasi-Newton method (a variation on the gradient method). At each step, Mathcad estimates the first partial derivatives of the errors with respect to the variables to be solved to create a *Jacobian matrix*. Ordinarily, Mathcad can determine the next estimate to make by computing the Gauss-Newton step *s* for each variable. In matrix notation, Mathcad solves the matrix equation:

$$J \cdot s = -f(x)$$

In this equation, \mathbf{J} is the Jacobian matrix, \mathbf{s} is the step to take, and \mathbf{x} is the vector of current estimates for unknown variables. For the first step, \mathbf{x} is the vector of guesses; at each subsequent step, the new \mathbf{x} is the old \mathbf{x} plus \mathbf{s} , the vector of steps. Notice that computing this step involves inverting the Jacobian matrix \mathbf{J} .

Computing this step is not always possible. Computing the step fails when the Jacobian matrix cannot be inverted or when there are more constraints than variables to be solved. In these cases, Mathcad adds the additional condition that the following quantity be reduced to a minimum:

$$\sum_j D_j^2 \cdot s_j^2$$

Here \mathbf{D} is a vector of weight factors computed from the norms of the columns of the Jacobian matrix. In these cases, \mathbf{s} is computed to satisfy this minimization criteria as well as solving the Newton equation with the Jacobian.

The Levenberg-Marquardt method does not work when there are fewer constraints than variables. In these cases, Mathcad returns the error “*too few constraints.*”

Termination criteria

The Levenberg-Marquardt method ends when it reaches one of the following termination criteria:

- When it is no longer possible to reduce the norm of the error vector significantly. In this context, “significantly” means by more than the larger of TOL and $TOL \cdot |error_vector|$. This criterion stops the solver when the errors cannot be reduced further.
- When \mathbf{s} becomes relatively close to zero. In this method, “relatively close” means a norm smaller than the larger of TOL and $TOL \cdot |\mathbf{v}|$. This criterion stops the solver when there is no preferred direction to move the guesses.

When it reaches one of these termination conditions, Mathcad checks the magnitude of the error vector and returns an answer:

- If the magnitude of the error vector is less than or equal to TOL, Mathcad returns the variable values as a solution.
- If the solve block ends in *Find* and the magnitude of the error vector is greater than TOL, Mathcad marks the *Find* with the error “*did not find solution.*” If the solve block ends in *Minerr*, Mathcad returns the solution anyway, even though the error vector is not close to zero.

There is a limit on the number of calculations that Mathcad will perform in search of an answer for a solve block. If it exceeds this limit without returning an answer, the *Find* or *Minerr* is marked with the error “*not converging.*”

In all cases, when the solver stops, Mathcad sets the value of the variable ERR to the magnitude of the error vector.

Mathcad's modifications to the Levenberg-Marquardt method

To make the Levenberg-Marquardt method more effective on actual problems, Mathcad includes the following modifications to the basic method:

- The first time the solver stops at a point that is not a solution, Mathcad adds a small random amount to all the variables and tries again. This helps avoid getting stuck in local minima and other points from which there is no preferred direction. Mathcad does this only once per solve, the first time the solver stops on a point that is not a solution.
- If you include inequality constraints in a solve block, Mathcad solves the subsystem consisting of only the inequalities first before adding in the equality constraints and attempting a full solution. This is equivalent to moving the guesses into an area where the inequalities are all satisfied before starting the solver.

Matrix operations

This section describes Mathcad's algorithms for computing determinants and inverses for square matrices.

To compute the determinant or inverse of a square matrix \mathbf{M} , Mathcad decomposes it into a lower-triangular matrix \mathbf{L} and an upper-triangular matrix \mathbf{U} , such that:

$$\mathbf{M} = \mathbf{L} \cdot \mathbf{U}$$

This is called the *LU decomposition* of the matrix.

The LU decomposition makes the solution of a system of equations $\mathbf{M} \cdot \mathbf{x} = \mathbf{y}$ into a matter of simple substitutions with the elements of \mathbf{L} , \mathbf{U} , and \mathbf{y} .

Performing the decomposition

Mathcad performs the LU decomposition using Crout's method with partial pivoting. This method is not described here. See *Numerical Recipes in C* by William H. Press, Brian P. Flannery, Saul A. Teukolosky and William T. Vetterling (Cambridge University Press, 1986), pp. 31 to 39, for a complete description.