# Chapter 15
# Solving Equations

This chapter describes how to solve equations ranging from a single equation in one unknown to systems of up to fifty equations in fifty unknowns. The techniques described here generate numeric solutions. Chapter 17, “Symbolic Calculation,” describes a variety of techniques for solving equations symbolically.

The following sections make up this chapter:

**Solving one equation**

How to use Mathcad's *root* function to numerically solve one equation in one unknown.

**Systems of equations**

How to use “solve blocks” to solve systems of *n* equations in *n* unknowns.

**Using the solver effectively**

Examples of how to solve systems of equations efficiently for various values of a parameter.

## Solving one equation

To solve a single equation in a single unknown, use the *root* function. This function takes an expression and one of the variables from the expression. It then varies that variable until the expression is equal to zero. Once this is done, the function returns the value that makes the expression equal zero.

---

root($f(z)$, $z$) Returns the value of $z$ at which the expression or function $f(z)$ is equal to 0. Both arguments to this function must be scalar. The function returns a scalar.

---

The first argument is either a function defined elsewhere in the worksheet, or an expression. It must return a scalar value.

The second argument is a variable name that appears in the expression. It is this variable that Mathcad will vary to make the expression go to zero. You should assign a number to this variable before using the *root* function. Mathcad uses this as a starting value in its search for a solution.

For example, to define *a* as the solution to the equation <Insert equation here>, follow these steps:

- Define a guess value for *x*. Type **x:3**. Your choice of guess value determines which root Mathcad returns.

$x := 3$

- Set the whole expression equal to zero. In other words, rewrite $e^x = x^3$ as $x^3 - e^x = 0$. It is this expression that you give the *root* function.
- Type **a:root(x^3[Space]-e^x[Space],x)** This defines the variable *a* to be a root of the desired equation.

$a := \text{root}(x^3 - e^x, x)$

- Type **a=** to see the root.

$a = 1.857$

When you use the root function, keep these suggestions in mind:

- Make sure that the variable is defined with a guess value before you use the *root* function.
- For expressions with several roots, for example $x^2 - 1 = 0$, your guess value determines which root Mathcad will return. Figure 15-1 shows an example in which

the *root* function returns several different values, each of which depends on the initial guess value.

- Mathcad will solve for complex roots as well as real roots. To find a complex root, you must start with a complex value for the initial guess.
- Solving an equation of the form $f(x) = g(x)$ is equivalent to using the *root* function as follows:

$$\text{root}(f(x) - g(x), x)$$

The root function can solve only one equation in one unknown. To solve several equations simultaneously, use the technique described in the next section, "Systems of Equations." To solve an equation symbolically, or to find an exact numerical answer in terms of elementary functions, choose **Solve for Variable** from the **Symbolic** menu or use the **solve** keyword. See Chapter 17, "Symbolic Calculation."

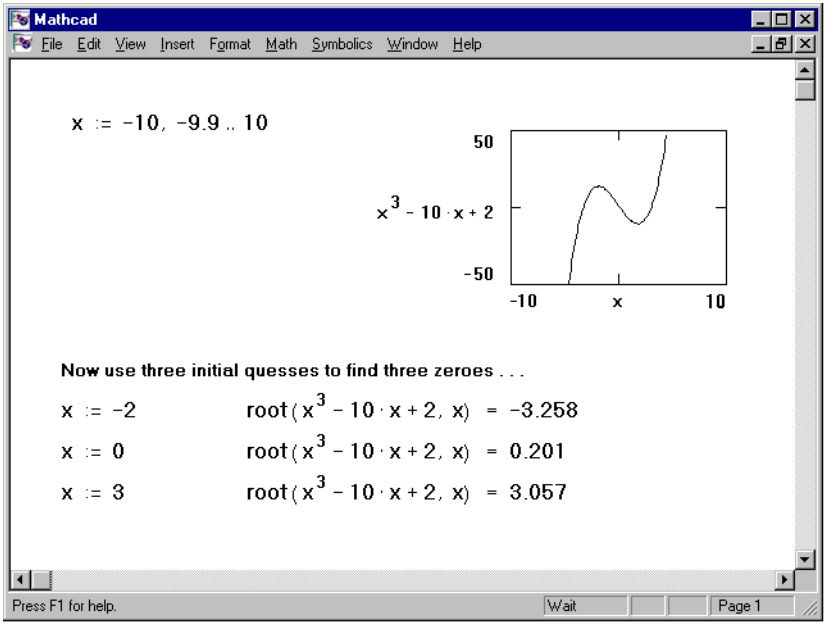


*Figure 15-1: Using a plot and the root function to find roots of an expression.*

## What to do when the root function does not converge

Mathcad evaluates the *root* function using the *secant method*. The guess value you supply for *x* becomes the starting point for successive approximations to the root value. When the magnitude of $f(x)$ evaluated at the proposed root is less than the value of the predefined variable TOL, the *root* function returns a result.

If after many approximations Mathcad still cannot find an acceptable answer, it marks the *root* function with an error message indicating its inability to converge to a result. This error can be caused by any of the following:

- The expression has no roots.
- The roots of the expression are far from the initial guess.
- The expression has local maxima or minima between the initial guess and the roots.
- The expression has discontinuities between the initial guess and the roots.
- The expression has a complex root but the initial guess was real (or vice versa).

To find the cause of the error, try plotting the expression. This will help determine whether or not the expression crosses the *x*-axis and if so, approximately where it does so. In general, the closer your initial guess is to where the expression crosses the *x*-axis, the more quickly the *root* function will converge on an acceptable result.

## Hints on using the root function

Here are some hints on getting the most out of the *root* function:

- To change the accuracy of the *root* function, change the value of the built-in variable TOL. If you increase TOL, the *root* function will converge more quickly, but the answer will be less accurate. If you decrease TOL, the *root* function will converge more slowly, but the answer will be more accurate. To change TOL at a specified point in the worksheet, include a definition like $TOL := 0.01$. To change TOL for the whole worksheet, choose **Options** from the **Math** menu, click on the Built-In Variables tab, and replace the number in the text box beside "TOL." After you click "OK," choose **Calculate Worksheet** from the **Math** menu to update the entire worksheet using the new value of TOL.
- If an expression has multiple roots, try different guess values to find them. Plotting the function is a good way to determine how many roots there are, where they are, and what initial guesses are likely to find them. Figure 15-1 shows an example of this. If two roots are close together, you may have to reduce TOL to distinguish between them.
- If $f(x)$ has a small slope near its root, then $\mathrm{root}(f(x), x)$ may converge to a value $r$ that is relatively far from the actual root. In such cases, even though $|f(r)| < TOL$, $r$ may be far from the point where $f(r) = 0$. To find a more accurate root, decrease the value of TOL. Or, try finding root($g(x)$,$x$), where

$$g(x) = \frac{f(x)}{\frac{d}{dx}f(x)}$$

- For an expression $f(x)$ with a known root $a$, solving for additional roots of $f(x)$ is equivalent to solving for roots of $h(x) = (f(x))/(x-a)$. Dividing out known roots like this is useful for resolving two roots that may be close together. It's often easier to solve for roots of $h(x)$ as defined here than it is to try to find other roots for $f(x)$ with different guesses.

## Solving an equation repeatedly

Suppose you want to solve an equation many times while varying one of the parameters in the equation. For example, suppose you want to solve the equation $e^x = a \cdot x^2$ for several different values of the parameter *a*. The simplest way to do this is to define a function:

$$f(a, x) := \text{root}(e^x - a \cdot x^2, x)$$

To solve the equation for a particular value of *a*, supply both *a* and a guess value, *x*, as arguments to this function. Then evaluate the function by typing **f(a,x)=**.

Figure 15-2 shows an example of how such a function can be used to find several solutions to the *root* function. Note that since the guess value, *x*, is passed into the function itself, there is no need to define it elsewhere in the worksheet.

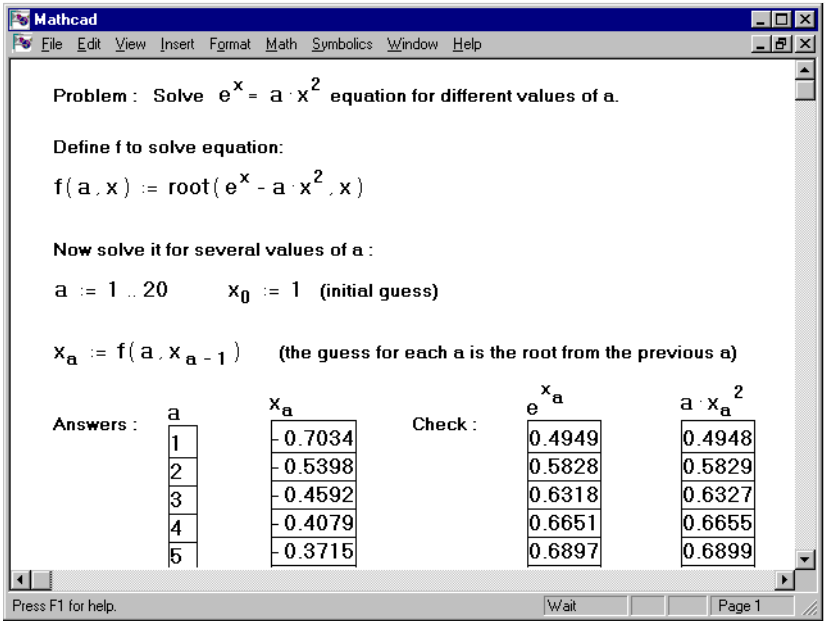


*Figure 15-2: Defining a user function with the root function.*

## Finding the roots of a polynomial

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. Figure 15-3 and Figure 15-4 show examples.

polyroots(**v**) Returns the roots of an *n*th degree polynomial whose coefficients are in **v**, a vector of length $n + 1$ . Returns a vector of length *n*.

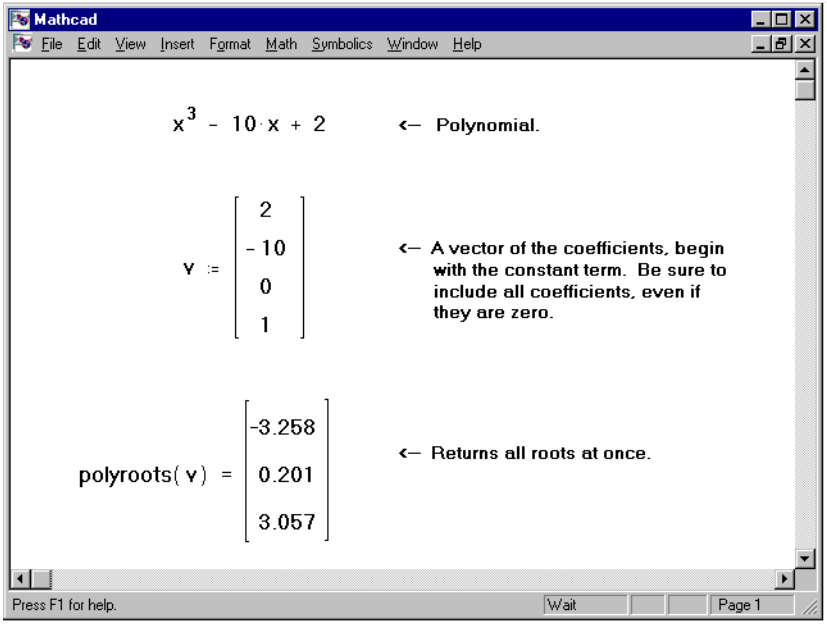


*Figure 15-3: Using polyroots to do the example shown in Figure 15-1.*

The *polyroots* function will always return numerical values for the roots of a polynomial. To find the roots symbolically, use the **solve** symbolic keyword or choose **Solve for Variable** from the **Symbolic** menu. See Chapter 17, "Symbolic Calculation."

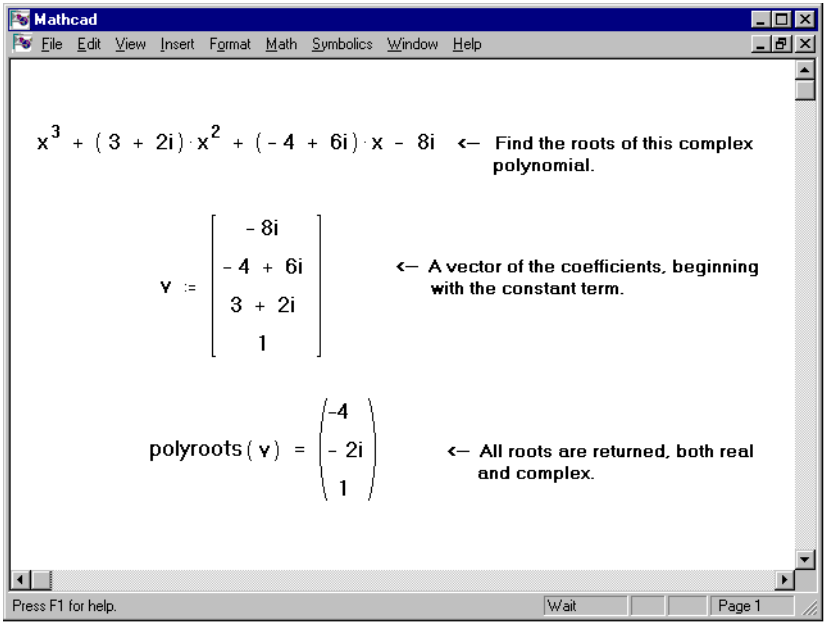


*Figure 15-4: Using polyroots to find the roots of a polynomial.*

## Systems of equations

Mathcad lets you solve a system of up to fifty simultaneous equations in fifty unknowns. The first part of this section sketches the procedure. The remainder contains several examples as well as a discussion of some common errors. The method given here will always return numbers for the unknown variables. To see the unknowns in terms of the other variables and constants, use the symbolic solve blocks discussed in the section "Solving equations symbolically" in Chapter 17.

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

- Provide an initial guess for all the unknowns you intend to solve for. Mathcad solves equations by making a series of guesses which ultimately converge on the right answer. The initial guesses you provide give Mathcad a place to start searching for solutions.
- Type the word *Given*. This tells Mathcad that what follows is a system of equations. You can type *Given* in any combination of upper and lower case letters, and in any font. Just be sure you don't type it while in a text region or paragraph.
- Now type the equations and inequalities in any order below the word *Given*. Make sure you use the symbol "=" to separate the left and right sides of an equation. Press

[**Ctrl**]**=** to type "=." You can separate the left and right sides of an inequality with any of the symbols <, >, ≤, and ≥.

- Type any equation that involves the *Find* function. Like *Given*, you can use any combination of upper and lowercase letters. You can also use any font, size or style.

| | |
|---|---|
| Find(*z1*,*z2*,*z3*, …) | Returns the solution to a system of equations. Number of arguments matches the number of unknowns. |

The *Find* function returns values as follows:

- If *Find* has one argument, it returns the value of that variable that solves the equation between it and the *Given*.
- If *Find* has more than one argument, it returns a vector of answers. For example, Find(*z1*, *z2*) returns a vector containing the values of *z1* and *z2* that solve the system of equations.

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

Figure 15-5 shows a worksheet that contains a solve block for one equation in one unknown. Since there is only one equation, only one equation appears between the word *Given* and the expression involving *Find*. Since there is only one unknown, the *Find* function has only one argument. For one equation in one unknown, you can also use the *root* function shown below.

$$a := \mathrm{root}(x^2 + 10 - e^x, x)$$

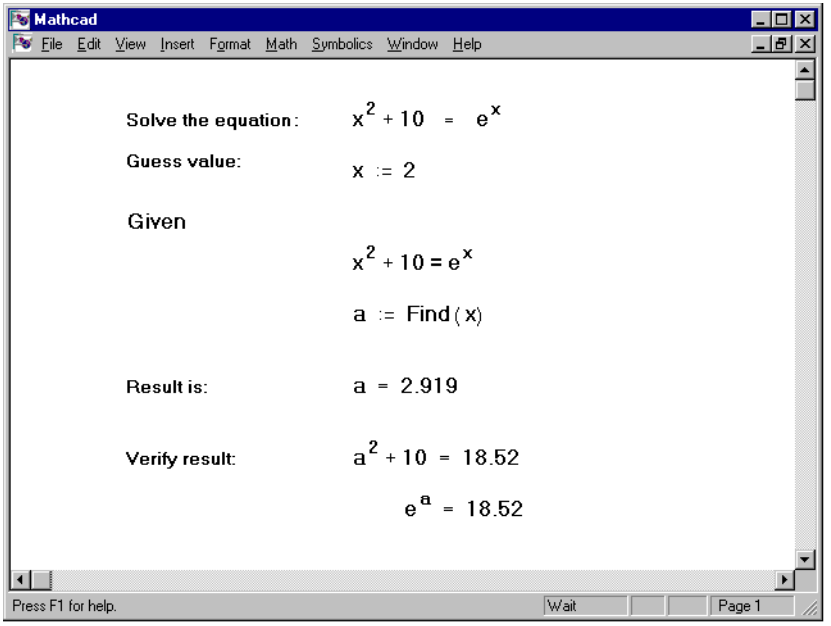


*Figure 15-5: A solve block with one equation in one unknown.*

Mathcad is very specific about the types of expressions that can appear between the *Given* and the *Find*. The table below lists all the expressions that can be placed in a solve block. These expressions are often called "constraints." In the table below, *x* and *y* represent real-valued scalar expressions, *z* and *w* represent arbitrary scalar expressions.

| Condition | Keystroke | Description |
|---|---|---|
| $z=w$ | **[Ctrl]=** | Constrained to be equal. |
| $x>y$ | **>** | Greater than. |
| $x<y$ | **<** | Less than. |
| $x\geq y$ | **[Ctrl]0** | Greater than or equal to. |
| $x\leq y$ | **[Ctrl]9** | Less than or equal to. |

Note that Mathcad does not allow the following inside a solve block:

- Constraints with "≠" in solve blocks.
- Range variables or expressions involving range variables of any kind.
- Inequalities of the form $a < b < c$.

If you want to include the outcome of a solve block in an iterative calculation, see the section "Using the solver effectively" on page 331.

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. This too is discussed in the section "Using the solver effectively" on page 331.

As a rule, you should never use assignment statements (statements like **x := 1**) inside a solve block. Mathcad marks assignment statements inside solve blocks with an appropriate error message.

Figure 15-6 shows a solve block with several kinds of constraints. There are two equations and two unknowns. As a result, the *Find* function contains two arguments, *x* and *y*, and returns a vector with two components.

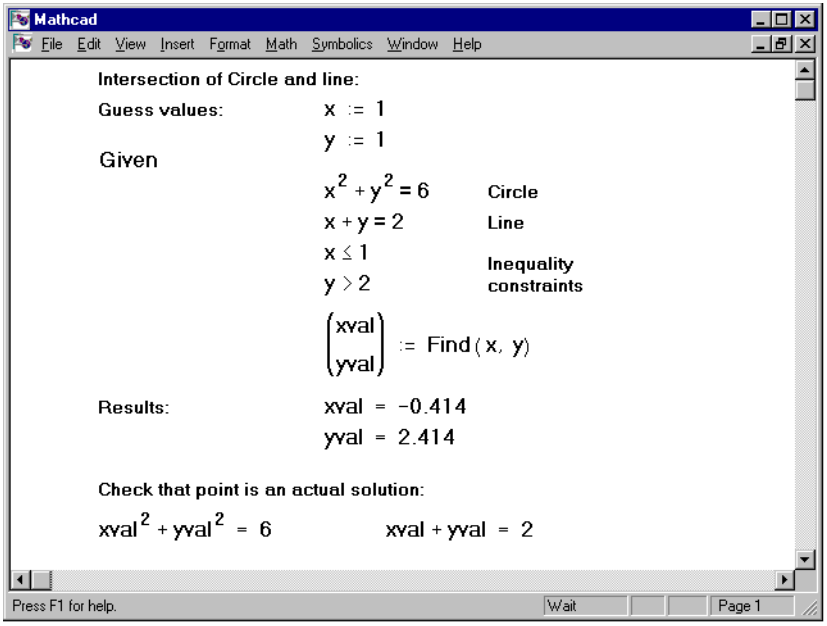


*Figure 15-6: A solve block with both equations and inequalities.*

## What to do with your solution

The *Find* function that terminates a solve block behaves like any other function. There are three things you can do with it:

- You can display it with an equation like $\text{Find}(variable) =$. An example is shown in the top half of Figure 15-7. If you have several variables, you can display a vector of results with an equation like: $\text{Find}(var1, var2, \ldots) =$. An example of how this would look for a system of two equations in two unknowns is shown in Figure 15-8.
- You can define a variable in terms of it by ending the solve block with an equation like $a := \text{Find}(x)$. This is useful when you want to use the solution of a system of equations elsewhere in the worksheet. Once you make this definition, *a* has the solved value of the variable. An example of this is shown in the lower half of Figure

15-7. If the *Find* returns a vector of values, you can enter an equation like $variable := \text{Find}(var1, var2, \ldots)$. If you do this, *variable* will end up being a vector instead of a scalar. You can also define variables as shown in Figure 15-6.

- Finally, you can define another function in terms of it by ending the solve block with an equation like $f(a, b, c) := \text{Find}(var1, var2, \ldots)$. This construction is useful for solving system of equations repeatedly for different values of some parameters *a, b, c, …* that appear within the system of equations itself. This method is described in the section"Using the solver effectively" on page 331.

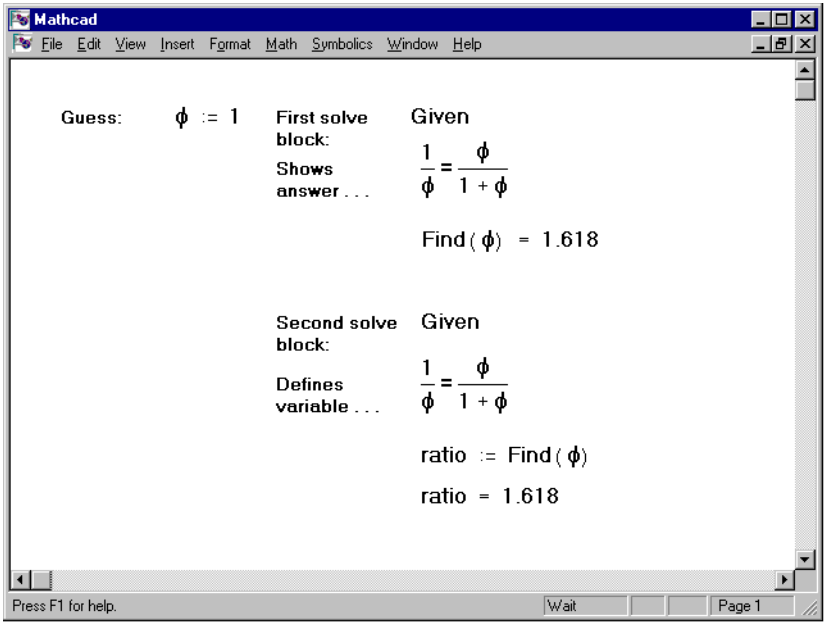


*Figure 15-7: You can display the result of a solve block directly, or you can put the result in a variable name for later use.*

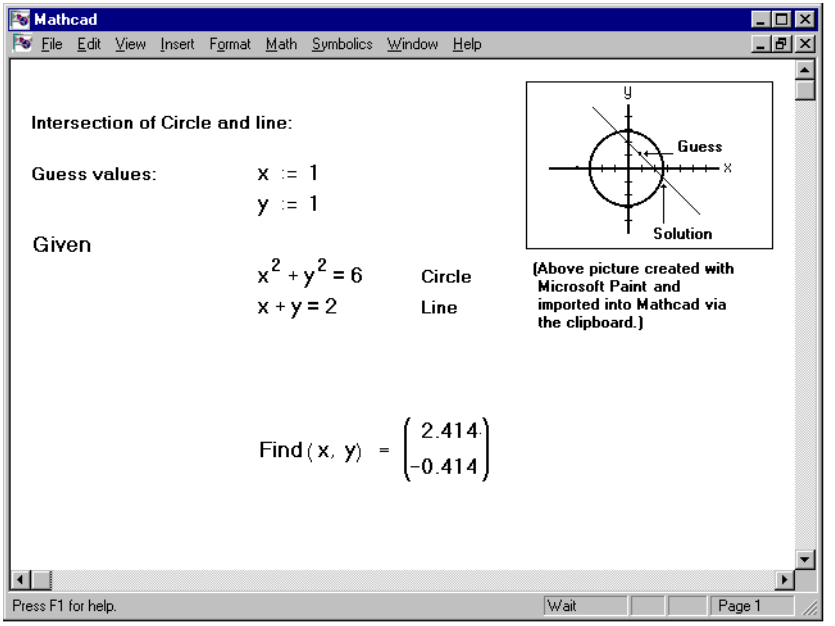


*Figure 15-8: When there are two or more unknowns, the Find function no longer returns a scalar. Instead, it returns a vector with as many elements as there are unknowns.*

Mathcad can return only one solution for a solve block. There may, however, be multiple solutions to a set of equations. To find a different solution, try different starting values or enter an additional inequality constraint that the current solution does not satisfy. Figure 15-9 shows how different starting values can yield a solution different from that shown in Figure 15-8. Figure 15-10 shows how to add an inequality to force Mathcad to find a different solution.

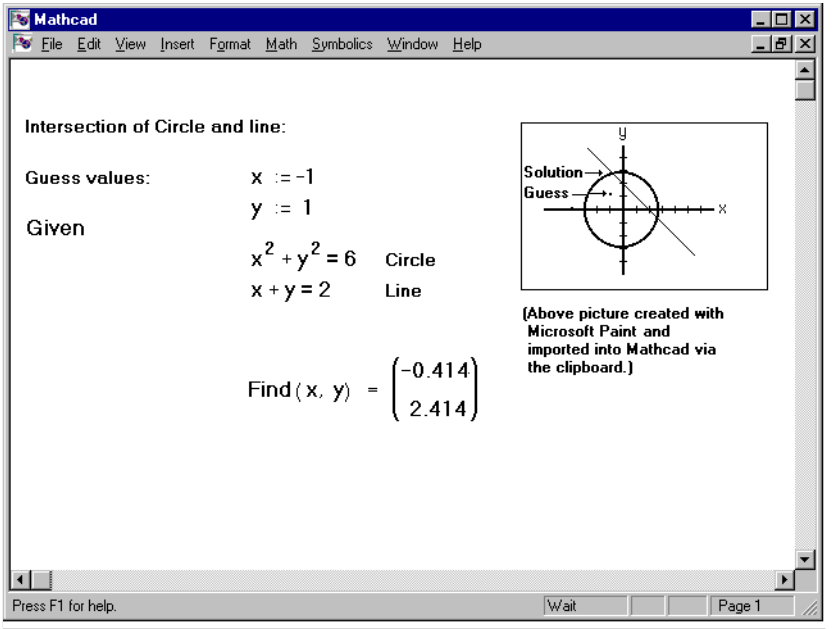


*Figure 15-9: A different guess leads to a solution different from that shown in Figure 15-8.*

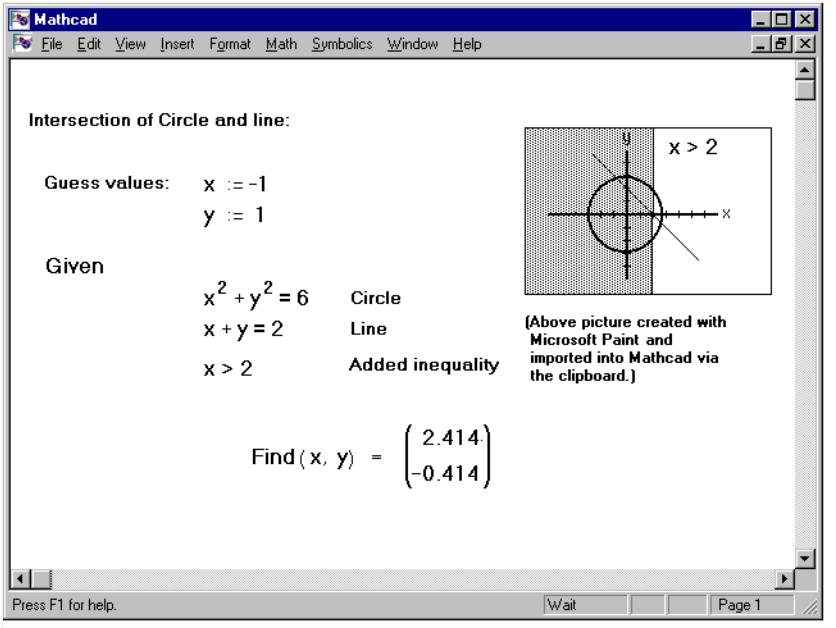


*Figure 15-10: Adding a constraint forces a different solution.*

## What to do when the solver does not reach a solution

If the solver cannot make any further improvements to the solution but the constraints are *not* all satisfied, then the solver stops and marks the *Find* with an error message indicating that it was unable to find a solution.

If you are having difficulty finding a solution, it often helps to plot the curve or curves in question. Plotting can provide graphical insight into where the solution might be. This will help you choose appropriate initial guesses for the variables.

Figure 15-11 shows a problem for which Mathcad could not find a solution.

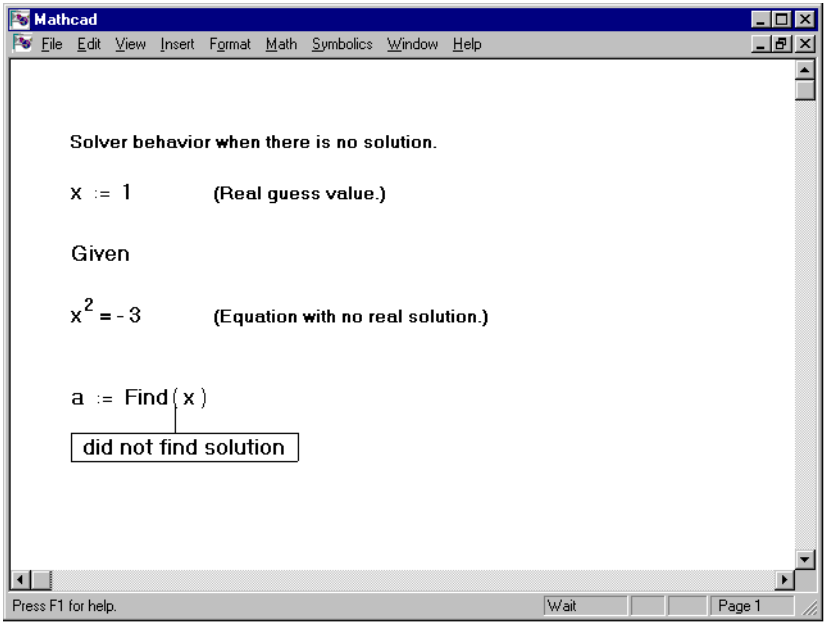


*Figure 15-11: A problem in which the solver fails to find a solution.*

The solver gives up trying to solve a system of equations 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. Figure 15-11 shows an example.
- The solver may have become trapped in a local minimum for the error values. The solving method that Mathcad uses will sometimes reach a point from which it cannot minimize the errors any further. 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. The strategies for avoiding this problem are the same as those for avoiding a local minimum: change the initial guesses or add an inequality to avoid the undesirable stopping point.
- It may not be possible to solve the constraints to within the desired tolerance. If the value of TOL is relatively small, Mathcad may have reached something very close to a solution but still be unable to solve all the constraints to an error less than TOL. 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.

### What to do when there are too few constraints

If there are fewer constraints than variables, Mathcad cannot run the solver at all. Mathcad then marks the *Find* with an appropriate error message.

A problem like that shown in Figure 15-12 is *underdetermined*. The constraints do not give enough information to find a solution. Because there are five arguments in the *Find* function, Mathcad thinks that you want to solve two equations with five unknowns. In general, such a problem has an infinite number of solutions.

To use the solver in Mathcad, you must provide at least as many equations as there are variables to solve for. If you specify the value of some of the variables, you may be able to solve for the remaining variables. Figure 15-13 shows how to fix the problem in Figure 15-12. Because the *Find* function contains only the arguments *z* and *w*, Mathcad knows that you want *x*, *y*, and *v* to be held constant at 10, 50 and 0 respectively. A solve block with two equations becomes legitimate because there are now only two unknowns, *z* and *w*.

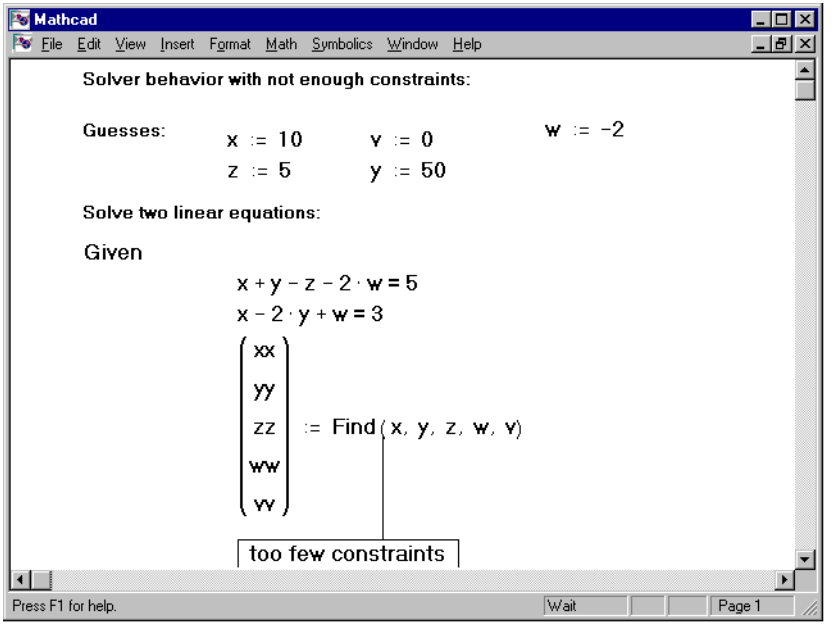


*Figure 15-12: Five arguments in the Find make the solver think you want to solve two equations in five unknowns.*

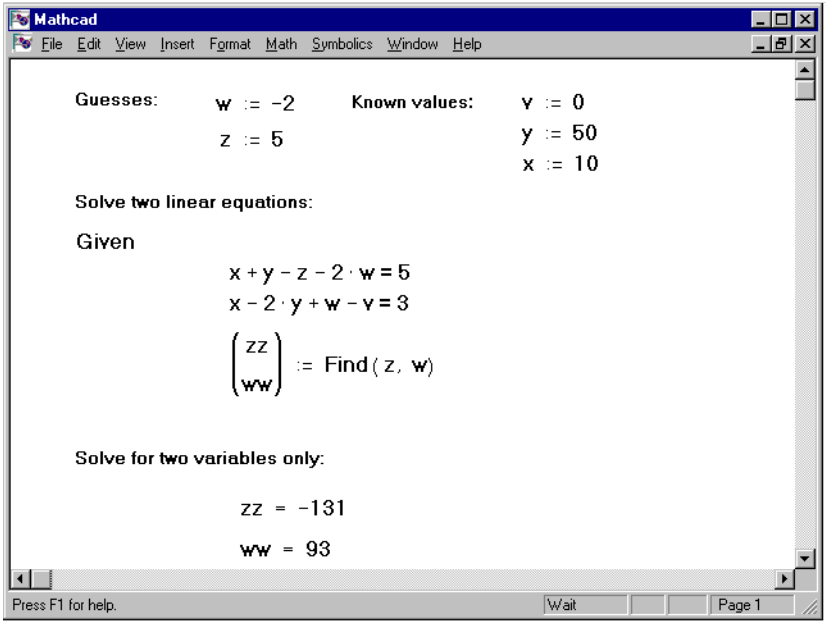


*Figure 15-13: The problem can be solved with fewer variables as arguments to Find.*

# Using the solver effectively

This section provides some ideas on how to effectively exploit Mathcad's ability to solve systems of simultaneous equations.

## Repeatedly solving an equation

The techniques given thus far, while they are effective for solving a particular system of equations, are limited by two things:

- Every time you use a *Find*, you must have the rest of the solve block to go with it.
- If you want to change some of the parameters or constants in your system of equations to see how these affect the solution, you have to go all the way back to the solve block to change them.

Both these drawbacks are overcome by Mathcad's ability to define a function in terms of a solve block.

If you define a function with *Find* somewhere on the right hand side, this function will solve the system of equations each time you use it. This overcomes the first problem.

If this function has as its arguments the same parameters that you want to vary in the solve block, you can simply change the parameters by changing the numbers you place in the function's argument list. This overcomes the second problem.

Figure 15-14 shows a concrete example. The friction factor, *f*, of a pipe depends on the pipe's diameter *D*, its roughness ε, and the Reynolds number *R*. It's quite conceivable that you would want to experiment with different size pipes (*D*) made of different types of concrete (ε).

The equation in Figure 15-14 shows the relationship between these parameters. The equation is too complicated to define a function of *R*, *D* and ε simply by solving for *f* in terms of *R*, *D* and ε.

You can, however, define a function in terms of a solve block. Whenever you ask Mathcad to evaluate the function $FricFac(\varepsilon, D, R)$, Mathcad takes the ε, *D*, and *R* that you supply, replaces the corresponding variables in the solve block, solves for *f*, and returns the value.

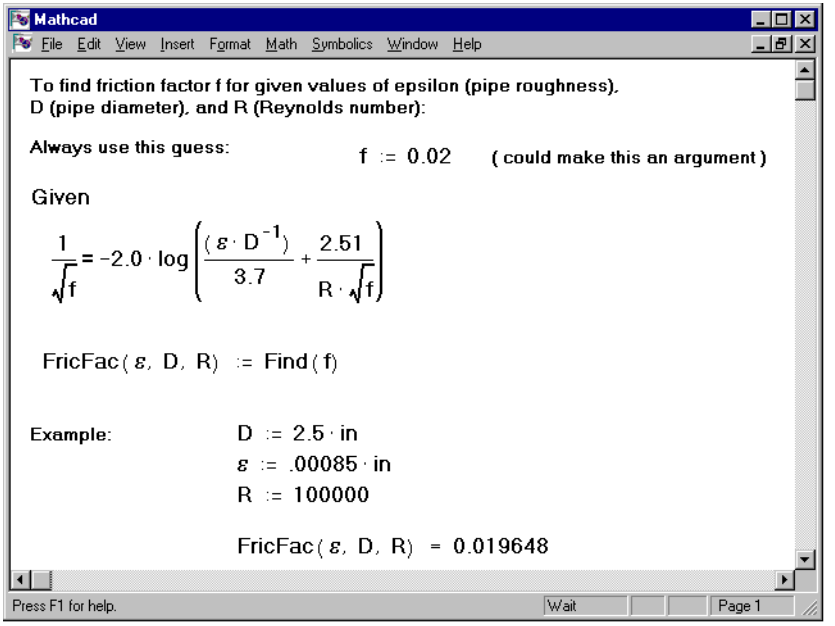


*Figure 15-14: Defining a function in terms of a solve block.*

Suppose that you've settled on a pipe size and material (*D* and ε), and you now want to try several different values of the Reynolds numbers. Although the *FricFac* function in Figure 15-14 was defined in terms of a solve block, it still is a function like any other. As such, you can use it with range variables.

Figure 15-15 shows how to solve for and plot the friction factor for many different values of the Reynolds number. Note that when you use range variables in conjunction with a solve block this way, you are actually solving the system of equations once for each value of the range variable. As a result, this type of calculation has the potential to be quite time consuming.

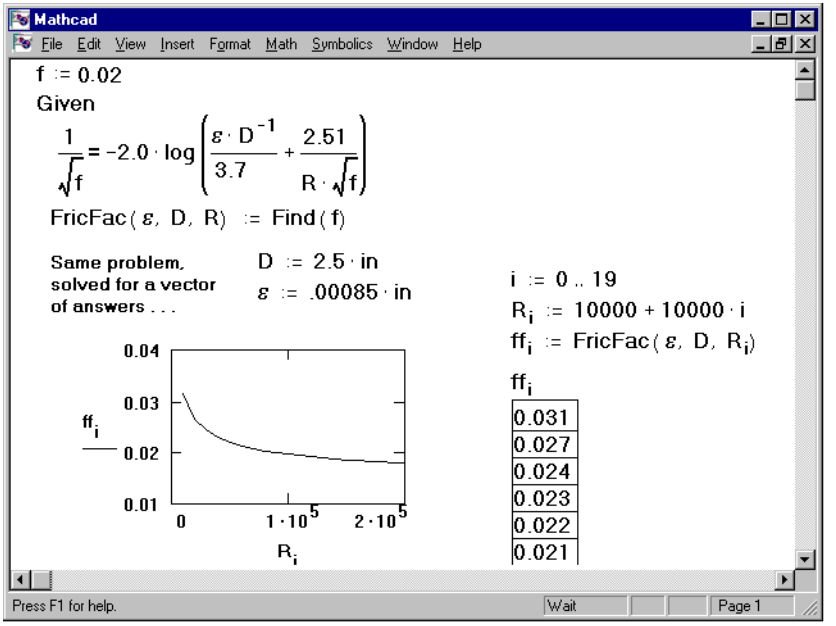


*Figure 15-15: A vector of solutions.*

The previous example involves only one equation in one unknown. It is possible to solve a system of equations iteratively as well; however, you must be careful not to ask Mathcad to display a table in which each entry in the table is something other than a single number. The example shown in Figure 15-16, a variation of Figure 15-10, shows how you can do this.

Suppose you are looking for the intersection of a line and a circle of varying radius, *R*. In keeping with the example of Figure 15-15, you could define a function in terms of a solve block. In this case, the appropriate function is $F(R) := Find(x, y)$. This function returns a vector whose elements are the *x* and *y* coordinates of the intersection.

The key difference is that this function returns a *vector* of two values for each value of *R*. Therefore when you ask for the answers by typing **F(R)=**, you are asking not for a table of numbers, but a table in which each element is a vector of two numbers. Since Mathcad has no way to display such a thing on your screen it returns an error message.

The solution is to display a table of the components $F(R)_0$ and $F(R)_1$ separately. By typing **F(R)[0=**, you get a table of all the *x* values of the intersection points. Similarly, by typing **F(R)[1=**, you get a table of all the *y* values of the intersection points.

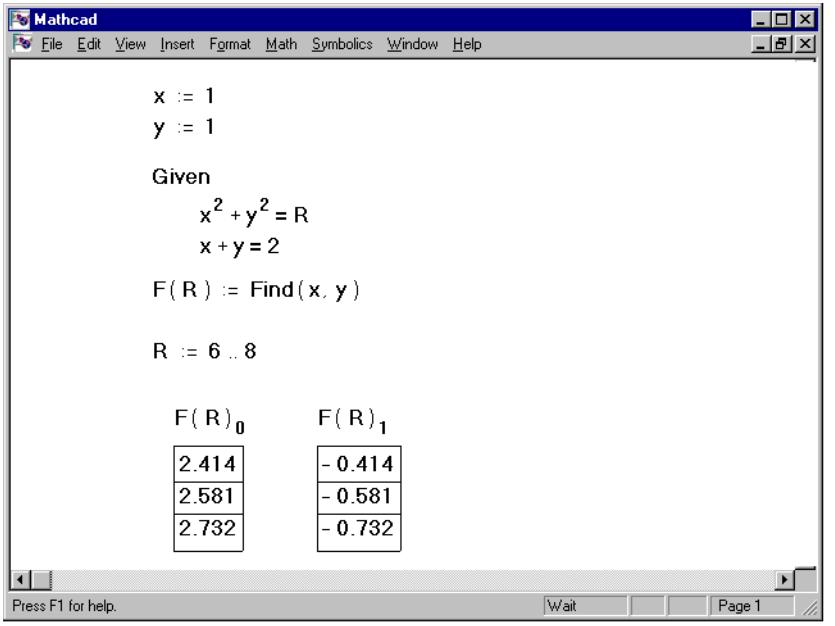


*Figure 15-16: How to display three solutions, each of which is a two element vector.*

## Solving the same problem for different variables

You will occasionally run into a problem in which you want to change the roles of knowns and unknowns in an equation. For example, consider the equation that relates interest rate, loan amount, term of loan, and payments. If you know three of these four quantities, you can solve for the missing one.

The worksheet in Figure 15-17 shows that for a 12% loan on a 30-year mortgage and a payment of $1000 per month, the largest possible loan is $97,218.33.

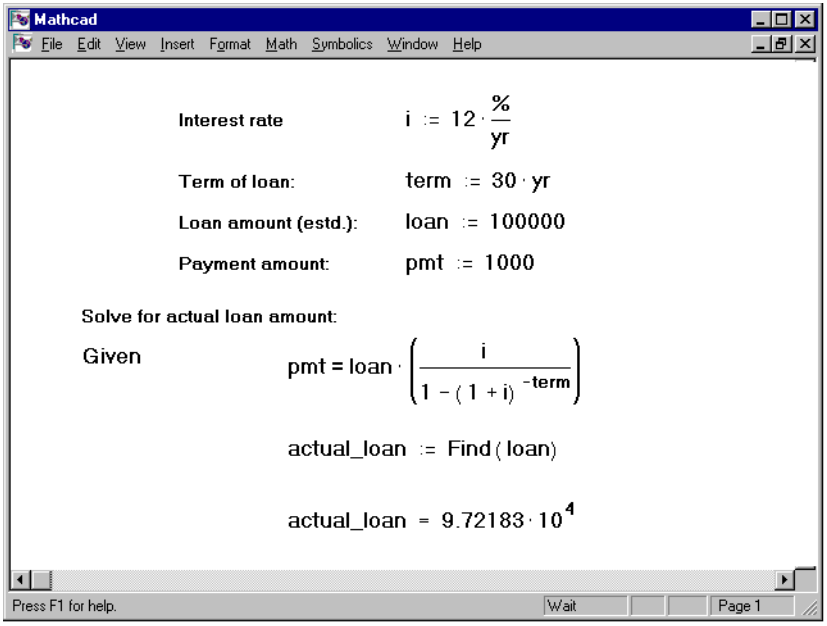


*Figure 15-17: Solving for the loan in a mortgage.*

With a few simple changes, the same worksheet can be used to solve for the interest rate. Suppose now that the amount of the loan is known to be $120,000. How far would interest rates have to drop to before the payments dropped to $1000 per month? Figure 15-18 shows the answer.

If you compare Figure 15-17 and Figure 15-18, you'll see that they are very much the same. The main difference lies in the argument of the *Find* function. To change what is fixed and what is variable in an equation, simply change the arguments of the *Find* function.

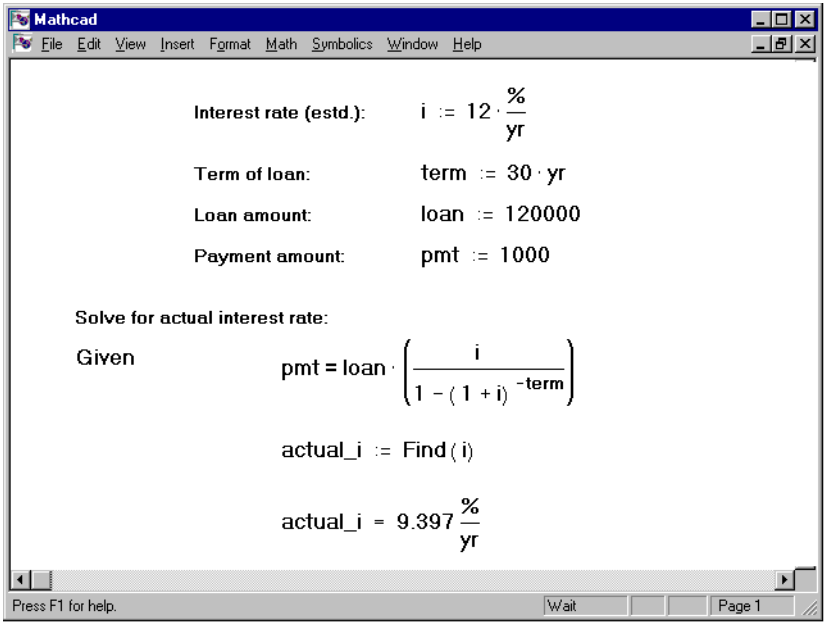


*Figure 15-18: Solving for the interest rate in a mortgage.*

## Approximate solutions

Mathcad supplies a function very similar to *Find* called *Minerr*. This function uses exactly the same algorithm as *Find*. The difference is that if the solver cannot make any further improvements to the solution, *Minerr* returns a value anyway. 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 would use *Find*.

| | |
|---|---|
| Minerr(*z1,z2,z3,* … ) | Returns the solution to a system of equations. Number of arguments matches the number of unknowns. |

*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. Figure 15-19 shows an example in which *Minerr* is used to obtain the unknown parameters in a Weibull distribution. The function *genfit* is also useful for solving

nonlinear least-squares problems. See Chapter 14, “Statistical Functions,” for more information on *genfit*.

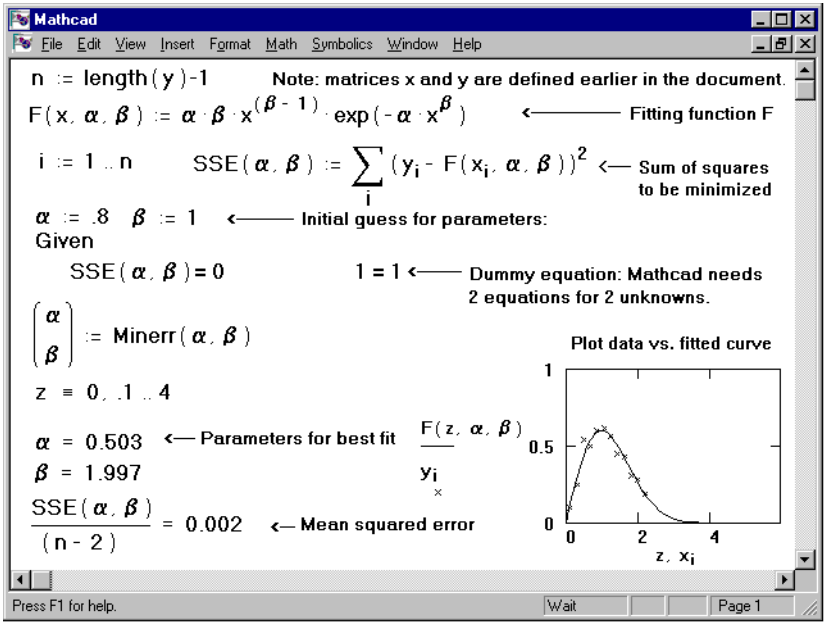


*Figure 15-19: Using the minerr function to do nonlinear least-squares fitting.*

## Using the symbolic solver

You can usually find numerical roots quickly and accurately with Mathcad's *root* function. But there are some circumstances in which you might want to use Mathcad's symbolic solver find exact or approximate roots:

- If the equation you're solving has a parameter, a symbolic solution may allow you to express the answer directly in terms of the parameter. Then instead of solving the equation over again for each new value of the parameter, you can just substitute its value into your symbolic solution.
- If you need all the complex roots of a polynomial of degree 4 or less, the symbolic solver will give them to you in a single vector, either exactly or numerically. The symbolic solver will also find complete solutions for *some* polynomials of higher degree.

See the section “Solving equations symbolically” in Chapter 17 for more information about solving equations symbolically.