Chapter 18
Programming

With Mathcad Professional, you can write your own programs using specialized programming operators. A Mathcad program has many attributes associated with programming languages including conditional branching, looping constructs, local scoping of variables, error handling, the ability to use other programs as subroutines, and the ability to call itself recursively.

Mathcad programs make it easy to do tasks that may be impossible or very inconvenient to do in any other way.

This chapter contains the following sections:

Defining a program

How to create simple programs. Local assignment statements.

Conditional statements

Using a condition to suppress execution of a statement.

Looping

Using **while** loops and **for** loops to control iteration.

Controlling program execution

Using the **break**, **continue**, and **return** statements to modify the execution of a loop or an entire program.

Error handling

Using the **on error** statement to trap errors and the *error* string function to issue error tips.

Programs within programs

Using subroutines and recursion in a Mathcad program.

Evaluating programs symbolically

Using a Mathcad program to generate a symbolic expression.

Programming examples

A sampling of programs displaying some of the power of Mathcad's approach to programming.

Defining a program

A Mathcad program is a special kind of Mathcad expression you can create in Mathcad Professional. Like any expression, a program returns a value—a scalar, vector, array, nested array, or string—when followed by the equal sign. In fact, as described in "Evaluating programs symbolically" on page 422, you can evaluate some Mathcad programs symbolically. And just as you can define a variable or function in terms of an expression, you can also define it in terms of a program.

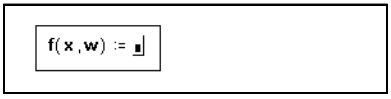
The main difference between a program and an expression is the way you tell Mathcad how to compute an answer. When you use an expression, you have to describe how to compute the answer in one statement. But when you use a program, you can use as many statements as you want to describe how to compute the answer. You can think of a program as a "compound expression."

The following example shows how to make a simple program to define the function:

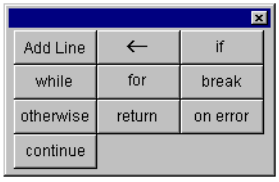
$$f(x, w) = \log\left(\frac{x}{w}\right)$$

Although the example chosen is so simple as to render programming unnecessary, it does illustrate how to separate the statements making up the program and how to use the local assignment operator, "←".

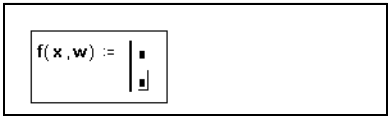
- Type the left side of the function definition, followed by a "**:=**". Make sure the placeholder is selected.



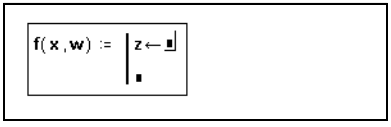
- Open the Programming Palette by clicking on the appropriate button in the Math Palette. Then click on the "Add Line" button. Alternatively, press **]**.



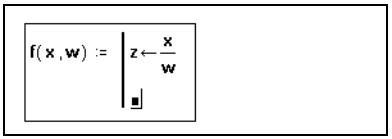
- You'll see a vertical bar with two placeholders. These placeholders will hold the statements making up your program. You can continue adding placeholders for statements as you need them by repeatedly clicking the "Add Line" button.



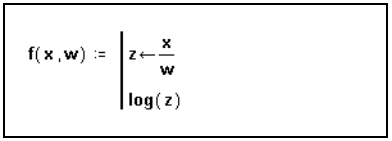
- Press [**Tab**] to move to the top placeholder. In the top placeholder, type **z**, and click on the "←" button on the Programming Palette. Alternatively, press **{** to insert a "←."



- In the placeholder to the right of the "←" type **x/w**.



- Press [**Tab**] to move to the bottom placeholder.
- The remaining placeholder is the actual value to be returned by the program. Type **log(z)**.



You can now use this function just as you would any other function. Figure 18-1 shows this function along with an equivalent function defined on one line instead of two. Note that *z* is undefined everywhere outside the program. The definition of *z* inside the program is local to the program. It has no effect anywhere else.

A program can have any number of statements. To add a statement, click the "Add Line" button on the toolbar again. Mathcad inserts a placeholder below whatever statement you've selected. To delete the placeholder, click on it and backspace over it.

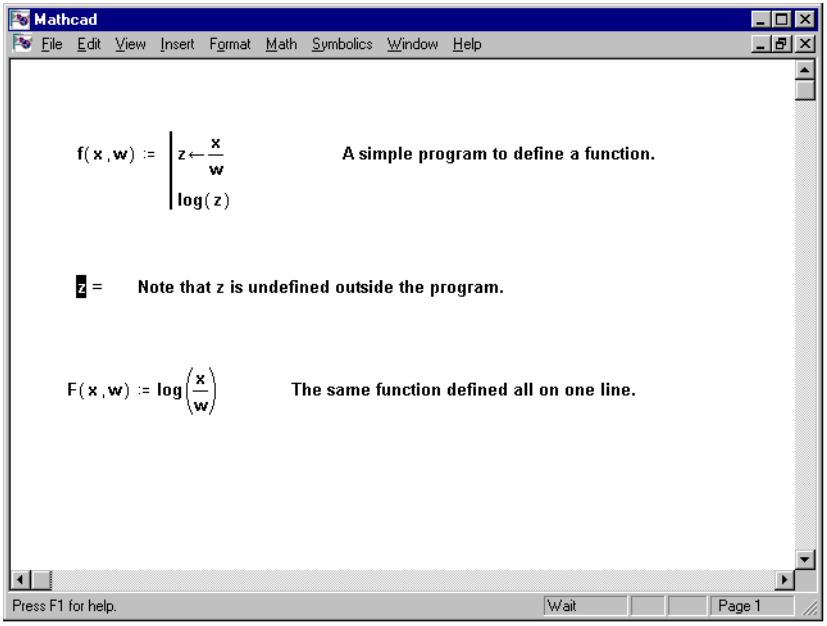


Figure 18-1: A function defined both in terms of a program and in terms of an expression.

Figure 18-2 shows a more complex example involving the quadratic formula. Although you can define the quadratic formula with a single statement as shown in the top half of the figure, you may find it simpler to define it with a series of simple statements as

shown in the bottom half. This lets you avoid having to edit very complicated expressions.

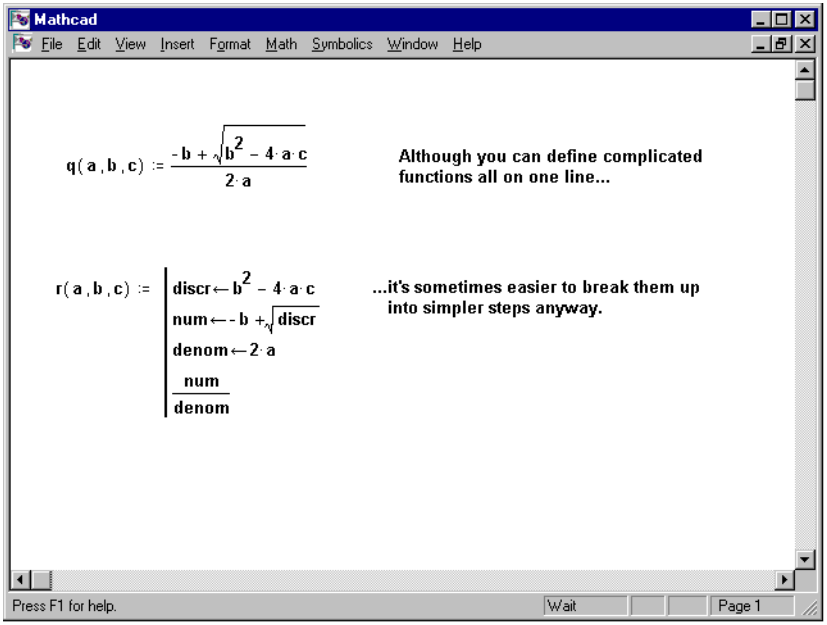


Figure 18-2: A more complex function defined in terms of both an expression and a program.

A Mathcad program, therefore, is an expression made up of a sequence of statements, each one of which is an expression in itself. Like any expression, a Mathcad program must have a value. This value is simply the value of the expression forming the last statement executed by the program. It could be a string expression or a single number, or it could be an array of numbers (see Figure 18-7, for example). It could even be a mixture of types, as described in the section “Nested arrays” in Chapter 10.

The remaining sections describe how to use conditional statements, how to use various looping structures to control program flow, how to handle errors, how to evaluate programs symbolically, and how to build more complicated programs.

Conditional statements

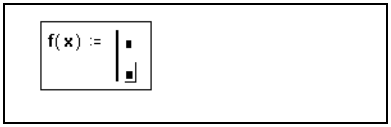
In general, Mathcad evaluates each statement in your program from the top down. There may be times, however, when you want Mathcad to evaluate a statement only when a particular condition is met. You can do this by including an **if** statement in your program. Note that the **if** statement in a Mathcad program is not the same as the *if* function, which you use elsewhere in your worksheet. The *if* function is described in

Chapter 13, “Built-in Functions.”

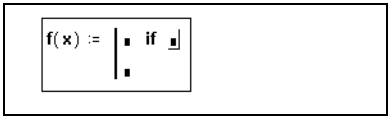
For example, suppose you want to define a function that forms a semicircle around the origin but is otherwise constant. To do this:

- Type the left side of the function definition, followed by a “**:=**”. Make sure the placeholder is selected.

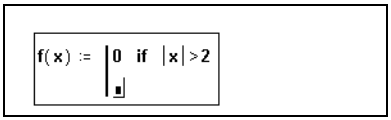
- Click the “Add Line” button on the Programming Palette. Alternatively, press **]**. You'll see a vertical bar with two placeholders. These placeholders will hold the statements making up your program.



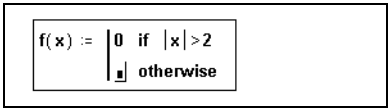
- In the top placeholder, click the “if” button on the Programming Palette. Alternatively, press **}**.



- In the right placeholder, type a boolean expression: an expression that's either true or false. In the left placeholder, type the value you want the expression to take whenever the expression in the right placeholder is true.



- Select the remaining placeholder and click the “otherwise” button on the Programming Palette.



- In the remaining placeholder, type the value you want the program to return if the condition in the first statement is not met.

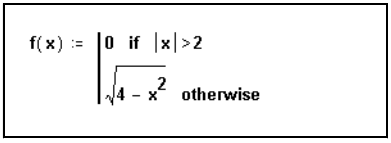


Figure 18-3 shows a plot of this function. Note that since this function only has two branches, it's not hard to define it using the *if* function as shown in Figure 18-3. However, as the number of branches exceeds two, the *if* function rapidly becomes unwieldy. An example is shown in Figure 18-4.

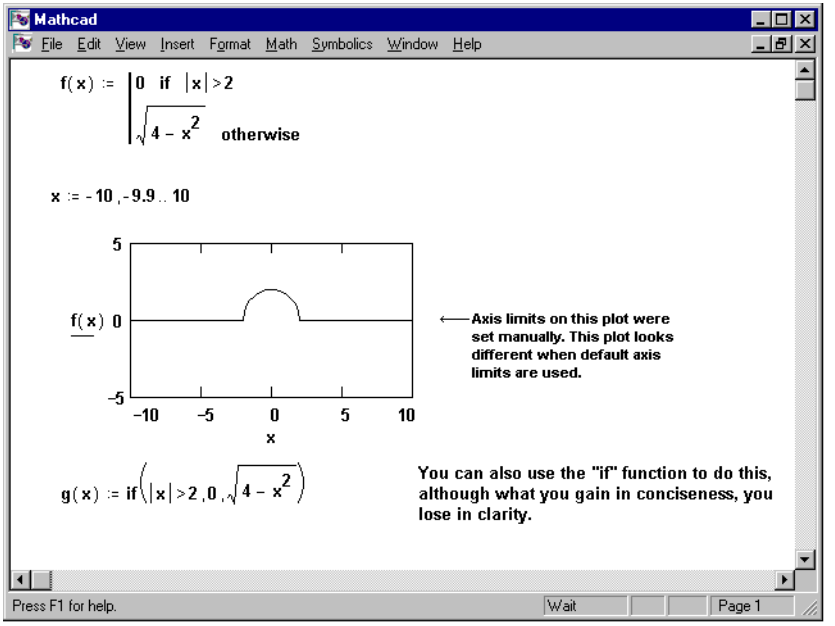


*Figure 18-3: Using the **if** statement to define a piecewise continuous function.*

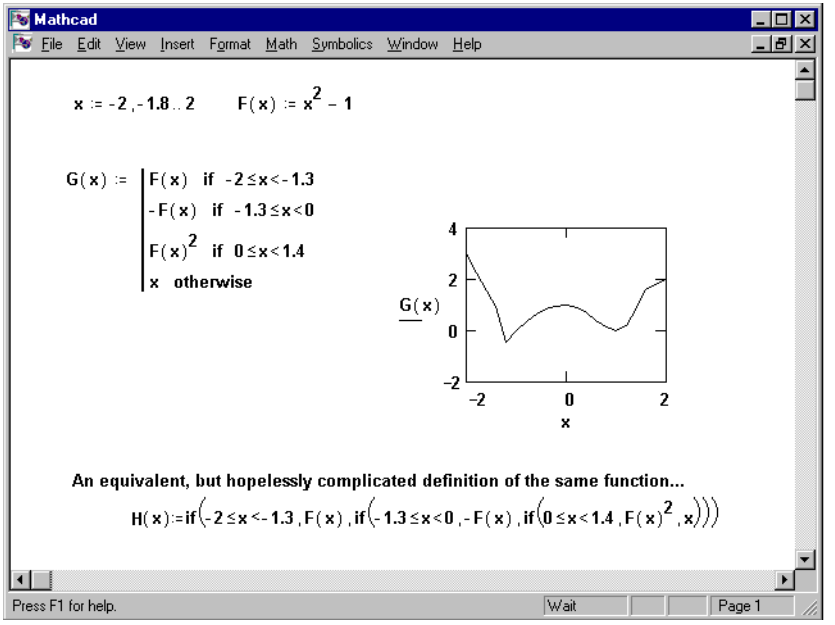


*Figure 18-4: Comparing the **if** statement in a program with the built-in "if" function.*

Looping

One of the greatest strengths of programmability is the ability to execute a sequence of statements over and over again in a loop. Mathcad provides two such loops. The choice of which loop to use depends on how you plan to tell the loop to stop executing.

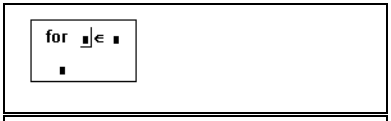
- If you know exactly how many times a loop is to execute, you can use a **for** loop.
- If you want the loop to stop upon the occurrence of a condition, but you don't know *when* that condition will occur, use a **while** loop.

"for" loops

A **for** loop is a loop that terminates after a predetermined number of iterations. Iteration is controlled by an iteration variable defined at the top of the loop. While in most cases you want a **for** loop to calculate for the complete number of iterations, you may also interrupt the loop on the occurrence of a particular condition. In such cases stop execution within the body of the loop using one of the methods described in "Controlling program execution" on page 413.

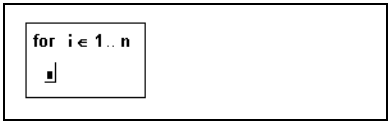
To create a **for** loop:

- Click the button labeled "for" on the Programming Palette. Do not type the word "for".



- In the placeholder to the left of the "∈," type the name of the iteration variable.

- In the placeholder to the right of the "∈," enter the range of values the iteration variable should take. You usually specify this range the same way you would for a range variable. See Chapter 11, "Range Variables," for more details.



- In the remaining placeholder, type the expression you want to evaluate repeatedly. This expression generally involves the iteration variable. If necessary, add placeholders by clicking the "Add Line" button on the Programming Palette.

The upper half of Figure 18-5 shows this **for** loop being used to add a sequence of integers. The undefined variable in Figure 18-5 shows that the definition of an iteration variable is local to the program. It has no effect anywhere outside the program.

The lower half shows an example in which the iteration variable is defined not in terms of a range but in terms of the elements of a vector. Although the expression to the right

of the "∈" is usually a range, it can also be a vector, a list of scalars, ranges and vectors separated by commas.

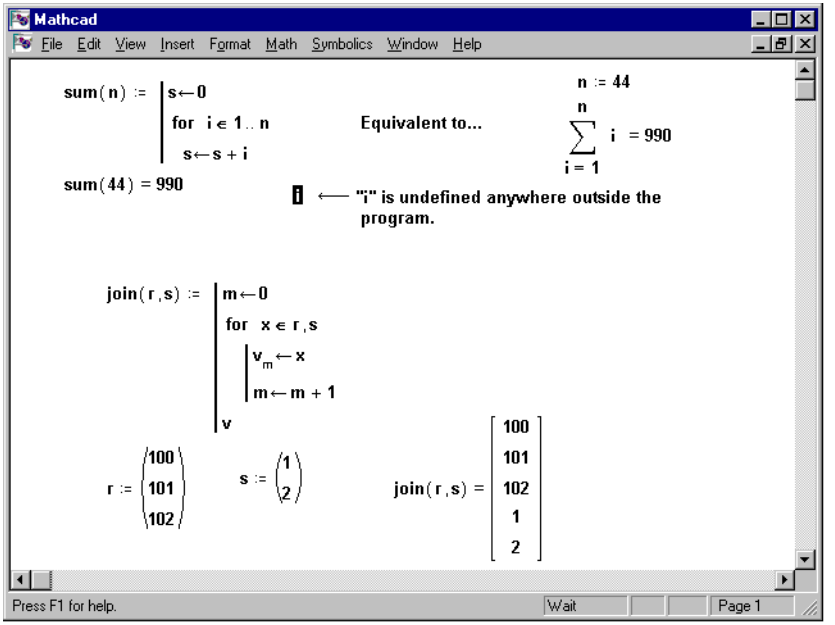


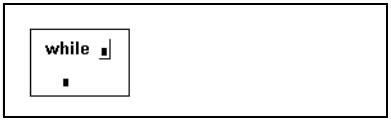
*Figure 18-5: Using a **for** loop with two different kinds of iteration variables.*

"while" loops

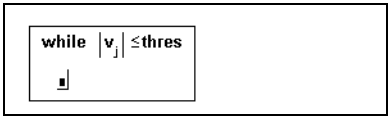
A **while** loop is driven by the truth of some condition. Because of this, you don't need to know in advance how many times the loop will execute. It is important, however, to have a statement somewhere, either within the loop or elsewhere in the program, that eventually makes the condition false. Otherwise, the loop will execute indefinitely. If you *do* find yourself in an endless loop or want to interrupt the loop on the occurrence of a particular condition, you can stop execution within the body of the loop using one of the methods described in "Controlling program execution" on page 413.

To create a **while** loop:

- Click the button labeled "while" on the Programming Palette.



- In the top placeholder, type a condition. This is typically a boolean expression like the one shown.



- In the remaining placeholder, type the expression you want evaluated repeatedly. If necessary, add placeholders by clicking the "Add Line" button on the Programming Palette.

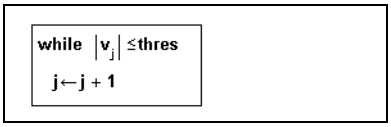


Figure 18-6 shows a larger program incorporating the above loop. Upon encountering a **while** loop, Mathcad checks the condition. If the condition is true, Mathcad executes the body of the loop and checks the condition again. If the condition is false, Mathcad exits the loop.

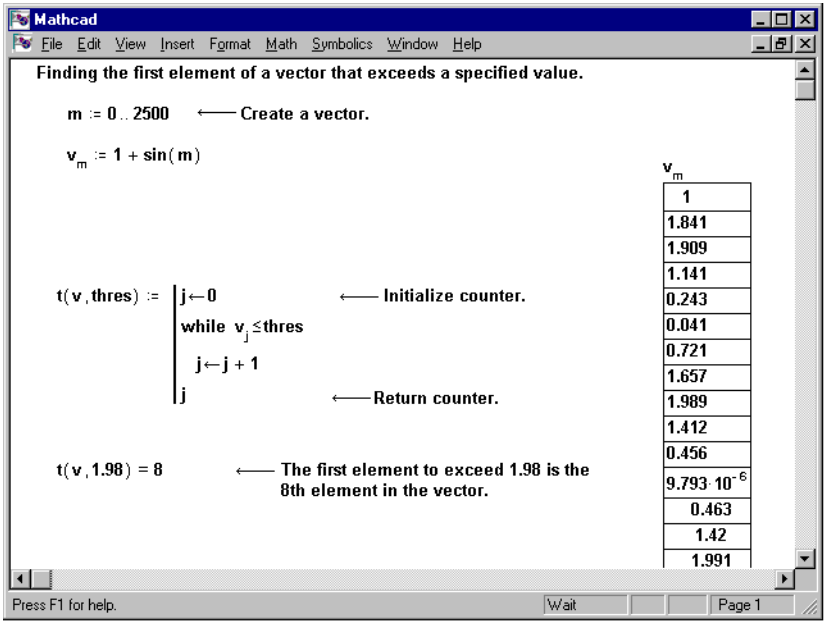


*Figure 18-6: Using a **while** loop to find the first occurrence of a particular number in a matrix.*

Controlling program execution

The Programming Palette in Mathcad Professional includes three statements for controlling program execution:

- Use the **break** statement within the body of the program to halt execution of the program upon the occurrence of a particular condition. Use **break** also within a **for** or **while** loop to stop the loop and move execution to the next statement outside the loop when a particular condition occurs.

- Use the **continue** statement within a loop to interrupt the current iteration and force program execution to continue with the next iteration of the loop.
- Use the **return** statement to stop a program and return a particular value from within the program rather than from the last statement evaluated.

The "break" statement

It is often useful to break out of a loop or stop program execution completely upon the occurrence of some condition. For example, there is a possibility of a runaway iteration in the program in Figure 18-6. If every element in **v** is less than *thresh*, the condition will never become false and iteration will continue past the end of the vector. This will result in an error message indicating that the index is pointing to a nonexistent array element. To prevent this from occurring, you can use a **break** statement as shown in Figure 18-7.

The program in Figure 18-7 will return 0 if no elements larger than *thresh* are found. Otherwise it returns the index and value of the first element exceeding *thresh*.

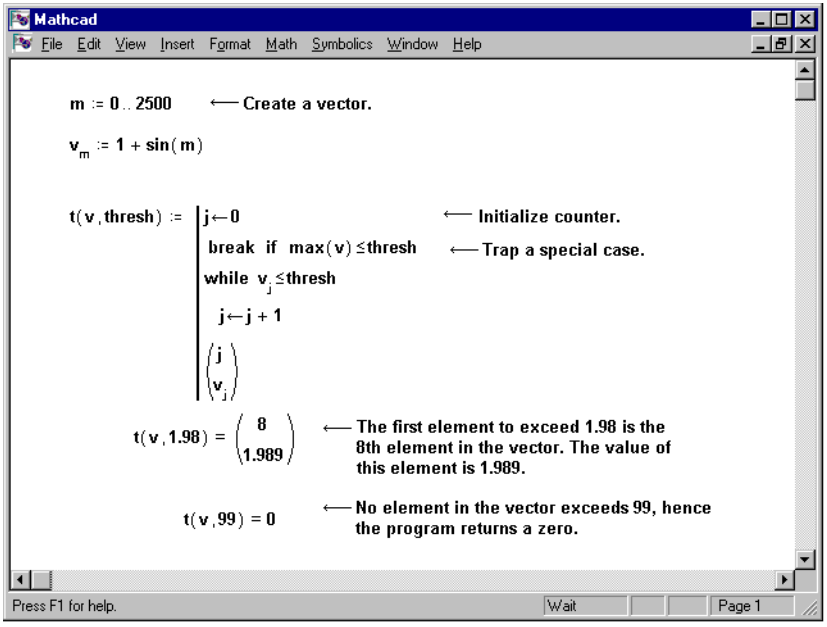


Figure 18-7: Example in Figure 18-6 modified to return both the index and the actual array value. Note the use of ***break*** *to prevent an error arising when thresh is too large.*

To insert the **break** statement, click on the "break" button in the Programming Palette. Note that to create the program in Figure 18-7, you would click the "break" button first, then click "if".

The “continue” statement

If you use **break** within a **for** or **while** loop, Mathcad stops execution within the loop and resumes execution starting with the next statement after the loop. To interrupt a loop and force calculation to resume at the beginning of the next iteration of the *current* loop, use **continue** instead.

To insert the **continue** statement, click on the “continue” button in the Programming Palette. As with **break**, you typically insert **continue** into the left-hand placeholder of an **if** statement created by clicking on the “if” button in the Programming Palette. The **continue** statement is evaluated only when the right-hand side of the **if** is true.

Figure 18-8 compares the effects of **break** and **continue** on the execution of a loop.

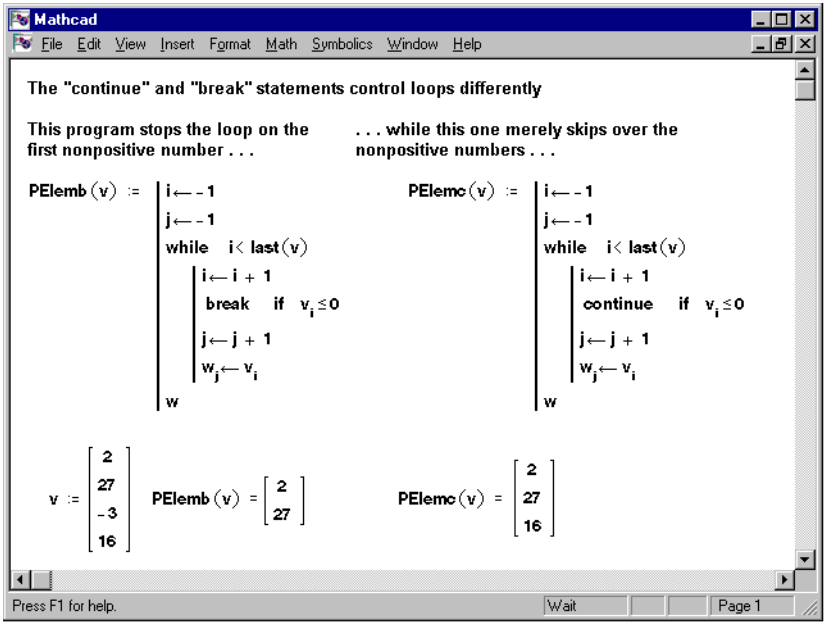


*Figure 18-8: The **break** statement halts the loop, but execution resumes on the next iteration when **continue** is used.*

The “return” statement

A Mathcad program returns the value of the last expression evaluated in the program. In simple programs, the last expression evaluated in the program is in the last line of the program (or, if **break** is used to interrupt a program, the last expression is the one that’s evaluated before **break** is called). As you create more complicated programs, you may need more flexibility. The **return** statement allows you to interrupt the program and return particular values other than the default value last computed by the program.

A **return** statement can be used anywhere in a program, even within a deeply nested loop, to force program termination and the return of a scalar, vector, array, or string. As with **break** and **continue**, you typically use **return** on the left-hand side of an **if**

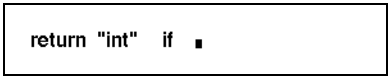
statement, and the **return** statement is evaluated only when the right-hand side of the **if** is true.

The following program fragment shows how a **return** statement is used to return a string value upon the occurrence of a particular condition:

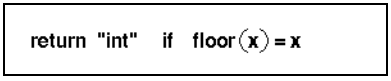
- Click the button labeled “return” on the Programming Palette.

- Now click the button labeled “if” on the Programming Palette.

- In the placeholder to the right of **return**, create a string expression by typing the double-quote key (**"**). Then type the string to be returned by the program. Mathcad displays the string between a pair of quotes. See Chapter 8, “Variables and Constants,” for more information on creating string expressions.



- In the placeholder to the right of **if**, type a condition. This would typically be a boolean expression like the one shown. To create the boolean equal sign, type **[Ctrl]=**.



In this example, the program returns the string “int” when the expression $floor(x) = x$ is true.

You can add more lines to the expression to the right of **return** by clicking on the “Add Line” button in the Programming Palette. The **return** statement forces the program to return the value of the last expression evaluated.

Figure 18-9 shows how the **return** statement can be used in programs.

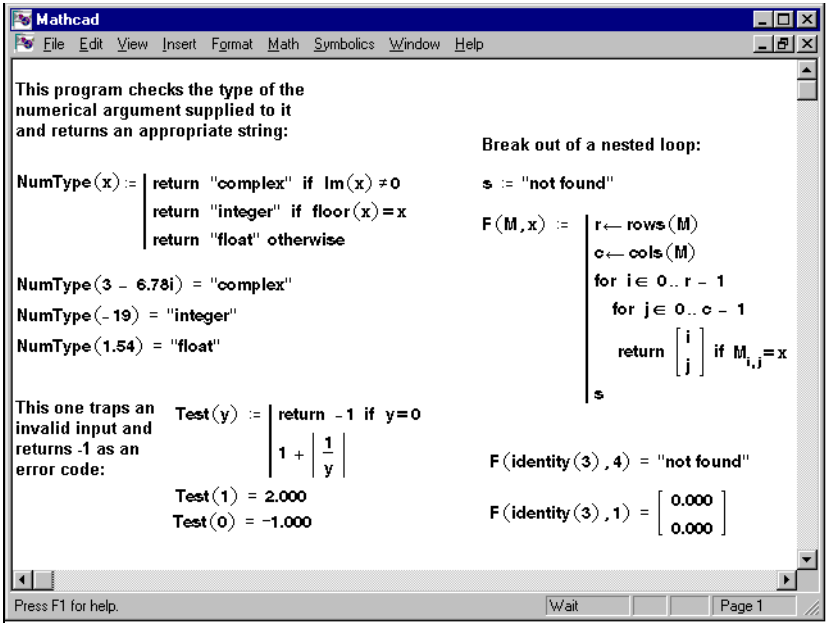


*Figure 18-9: Using **return** to control program execution and return specialized values.*

Error handling

Errors may occur during program execution that will cause Mathcad to stop calculating the program. For example, because of a particular choice of input, a program may attempt to divide by 0 in some expression and will encounter a singularity error. In these cases Mathcad treats the program as it does other math expressions: it marks the offending expression with an error message and highlights the offending name or operator in a different color, as described in Chapter 7, "Equations and Computation."

Mathcad Professional gives you two features to improve error handling in programs:

- The **on error** statement on the Programming Palette allows you to "trap" a numerical error that would otherwise force Mathcad to stop calculating the program.
- The *error* string function gives you access to Mathcad's error tip mechanism and lets you customize error messages issued by your program.

"on error" statement

In some cases you may be able to anticipate program inputs that will lead to a numerical error (such as a singularity, an overflow, or a failure to converge) that would force Mathcad to stop calculating the program. As shown in Figure 18-9, the **return**

statement combined with an appropriate boolean test can be used in simple cases to trap values that will lead to program errors. In more complicated cases, especially when your programs rely heavily on Mathcad's numerical operators or built-in function set, you may not be able to anticipate or enumerate all of the possible numerical errors that can occur in a program. The **on error** statement is designed as a general-purpose error trap to compute an alternative expression when a numerical error occurs that would otherwise force Mathcad to stop calculating the program.

To use the **on error** statement, click on the button labeled "on error" on the Programming Palette. In the placeholder to the right of "on error" create the program statement(s) you ordinarily expect to evaluate, but in which you wish to trap any numerical errors. In the placeholder to the left create the program statement(s) you want to evaluate should the default expression on the right-hand side fail.

Figure 18-10 shows **on error** operating in a program to find a root of an expression.

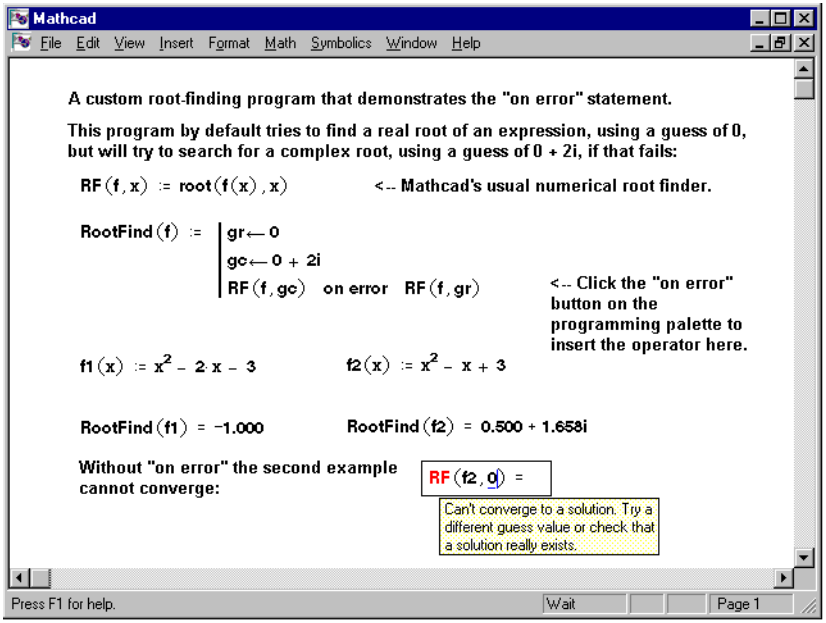


*Figure 18-10: Using the **on error** statement to trap numerical errors in programs.*

Custom error messages

Just as Mathcad automatically produces an appropriate "error tip" when you click on an expression that cannot calculate because of an error (see the bottom of Figure 18-10 for an example), you may want specialized error messages to appear when your programs are used improperly or cannot return answers. Mathcad Professional's *error* string function gives you this capability. For example, you can design custom error tips to appear when incorrect arguments are supplied to the program or when particular conditions are encountered in the program.

The *error* string function, described in the section “String functions” in Chapter 13, evaluates to produce an error tip whose text is simply the string expression it takes as an argument. Typically you will use the *error* string function in the placeholder on the left-hand side of an **if** or **on error** programming statement so that the *error* string function generates an appropriate error tip when a particular condition is encountered.

Figure 18-11 shows how custom error messages can be used even in a small program.

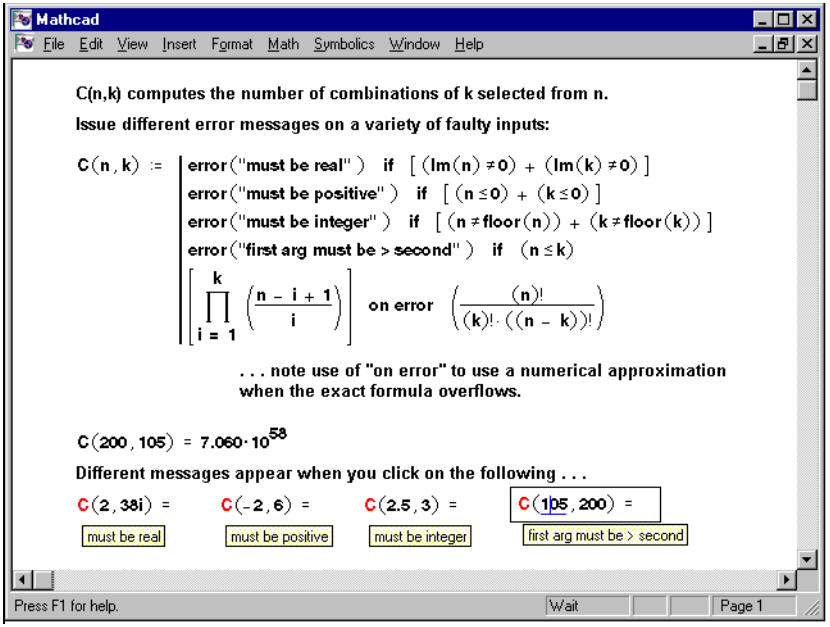


Figure 18-11: Issuing custom error tips via the “error” string function.

Programs within programs

The examples in previous sections have been chosen more for their simplicity than their power. This section shows some examples of more complicated programs capable of performing tasks that would be difficult if not impossible without the availability of these programming features.

Much of the flexibility inherent in programming arises from the ability to embed programming structures inside one another. In Mathcad, you can do this in three ways:

- You can make one of the statements in a program be another program.
- You can define a program elsewhere and call it from within another program as if it were a subroutine.
- You can define a function recursively.

The remainder of this section illustrates these techniques by example.

Subroutines

Recall that a program is just an expression made up of statements, each one of which contains an expression. Since a program statement must be an expression, and since a program is itself an expression, it follows that a program statement can be another program.

Figure 18-12 shows two examples of programs containing a statement which is itself a program. The example on the right-hand side of Figure 18-12 shows how to nest programs even more deeply. In principle, there is no limit to how deeply nested a program can be. As a practical matter, however, programs containing deeply nested programs can become too complicated to understand at a glance.

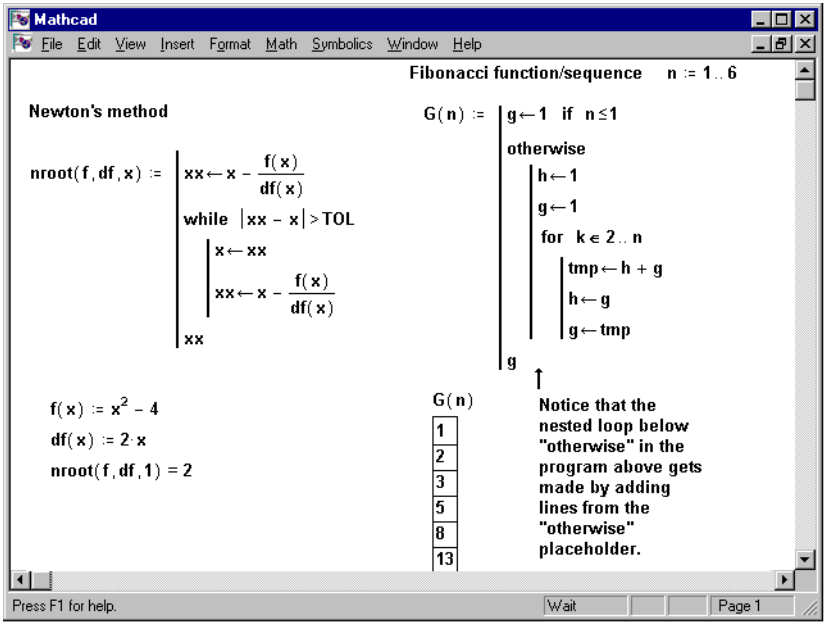


Figure 18-12: Programs in which statements are themselves programs.

One way many programmers avoid overly complicated programs is to bury the complexity in a *subroutine*. Figure 18-13 shows how you can do something similar in Mathcad. By defining *intsimp* elsewhere and using it within *adapt*, the program used to define *adapt* becomes considerably simpler.

The function *adapt* carries out an adaptive quadrature or integration routine by using *intsimp* to approximate the area in each subinterval. If you look at the last line, you'll notice that *adapt* actually calls itself. In other words, it's defined recursively. The following section discusses recursive function definitions in more detail.

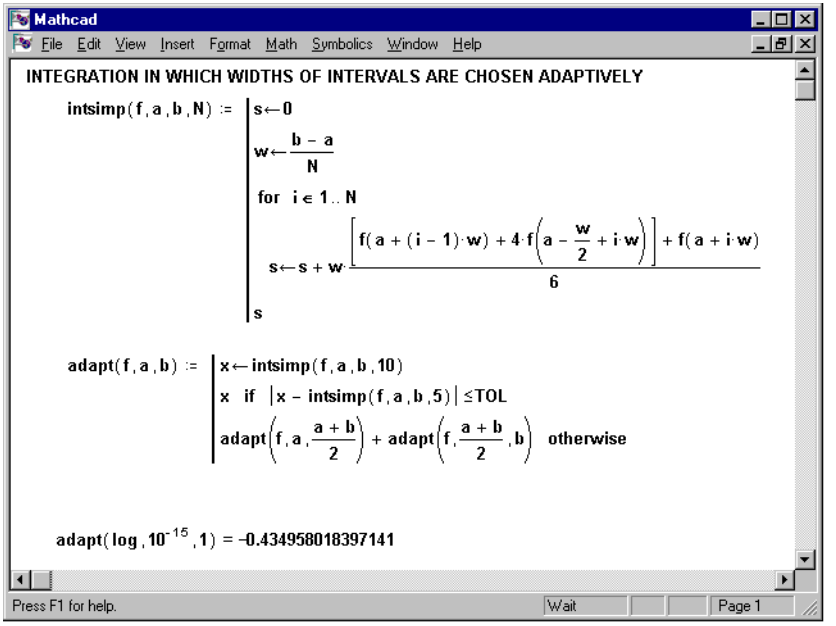


Figure 18-13: Using a subroutine to manage complexity.

Recursion

Recursion is a powerful programming technique that involves defining a function in terms of itself as shown in Figure 18-14. Recursive function definitions should always have at least two parts:

- An initial condition to prevent the recursion from going forever, and
- A definition of the function in terms of a previous value of the function.

The idea is similar to that underlying mathematical induction: if you can get $f(n + 1)$ from $f(n)$, and you know $f(0)$, then you know all there is to know about *f*.

Keep in mind, however, that recursive function definitions, despite their elegance and conciseness, are not always the most computationally efficient definitions. You may find that an equivalent definition using one of the iterative loops described earlier will evaluate more quickly.

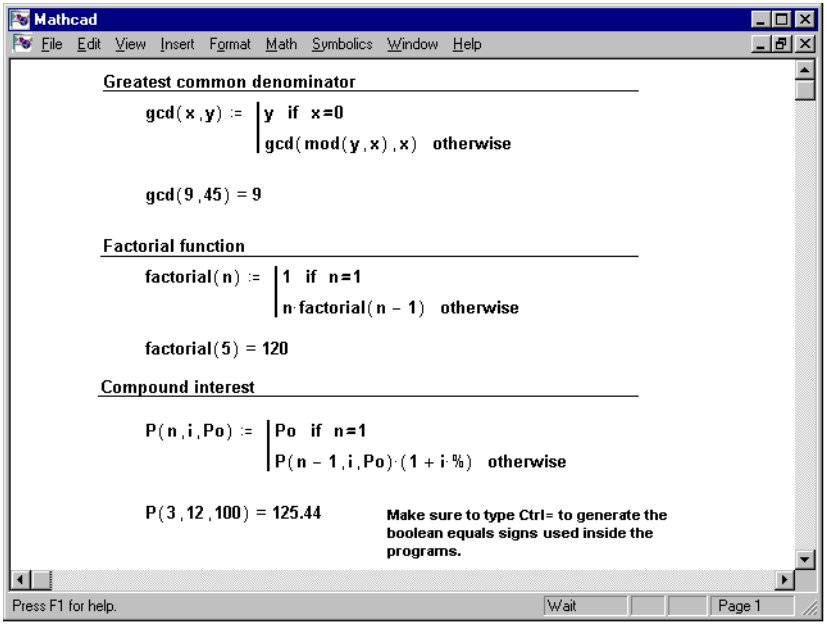


Figure 18-14: Defining a function recursively.

Evaluating programs symbolically

Like any Mathcad expression, a Mathcad program returns a numerical value when followed by the equal sign. You can also write a Mathcad program to return a *symbolic* expression when you evaluate it symbolically using the methods described in Chapter 17, "Symbolic Calculation." For example, when you evaluate a program using the live symbolic operator, "→," Mathcad passes the expression to its symbolic processor and, where possible, returns a simplified symbolic expression. You can use Mathcad's ability to evaluate programs symbolically to generate complicated symbolic expressions, polynomials, and matrices.

The following simple program substitutes one expression for another and returns a simplified symbolic result:

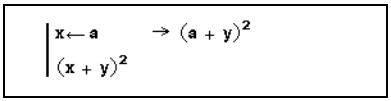
- Click the button labeled "Add Line" on the Programming Palette.

- Create a local assignment of variable *a* to *x* in the top placeholder. Type **x**, click on the "←" button on the Programming Palette, and type **a**.

- In the bottom placeholder, enter an expression that depends on *x*, such as $(x + y)^2$.

- Press **[Ctrl].** (the control key followed by a period). Mathcad displays a right arrow, "→," to the right of the program.

- Click outside the expression to see the simplified result.



Only one of the variables in this example, *x*, is defined locally in the program, but Mathcad understands the other variables, *a* and *y*, as symbols and substitutes *a* for *x* in the expression it returns.

Figure 18-15 below shows a function that is defined in terms of a program. Although the function cannot be evaluated numerically in Mathcad, when the function is evaluated symbolically, it generates symbolic polynomials. See "The Treasury Guide to Programming" in the Resource Center for more examples of evaluating a Mathcad program symbolically.

Programs that include the **return** and **on error** statements cannot be evaluated symbolically since the symbolic processor does not recognize these operators.

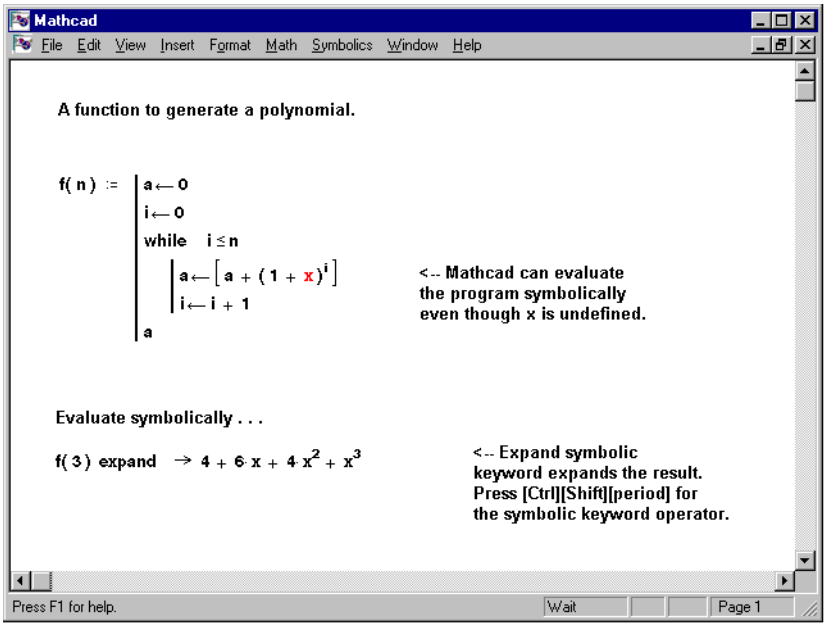


Figure 18-15: Using a Mathcad program to generate a symbolic expression.

Programming examples

With only ten buttons on the Programming Palette, Mathcad's programming environment is easy to use. Nevertheless, this simplicity conceals a surprising amount of programming power. When combined with Mathcad's rich numerical and symbolic functionality and used in conjunction with the abstract data structures provided by Mathcad's nested arrays, these ten operators enable you to write sophisticated programs in Mathcad.

The following figures illustrate just a few of the possibilities. As you experiment with programming in Mathcad, you'll discover many new applications. For further programming examples, see the “Programming” topic in the Resource Center QuickSheets. And in Mathcad Professional the Resource Center includes a special section, “The Treasury Guide to Programming,” which includes detailed examples and applications to show you how to get more out of Mathcad programs.

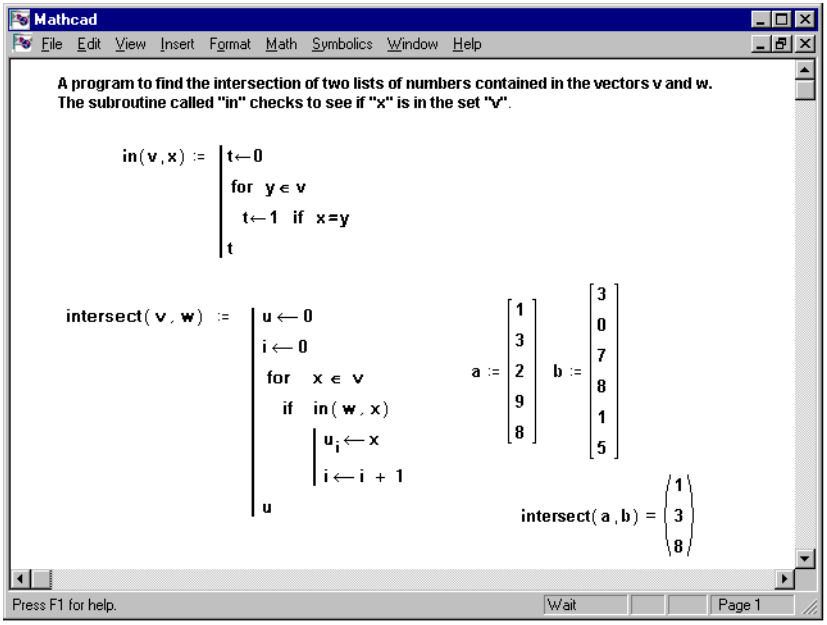


Figure 18-16: Program to find the numbers common to two vectors.

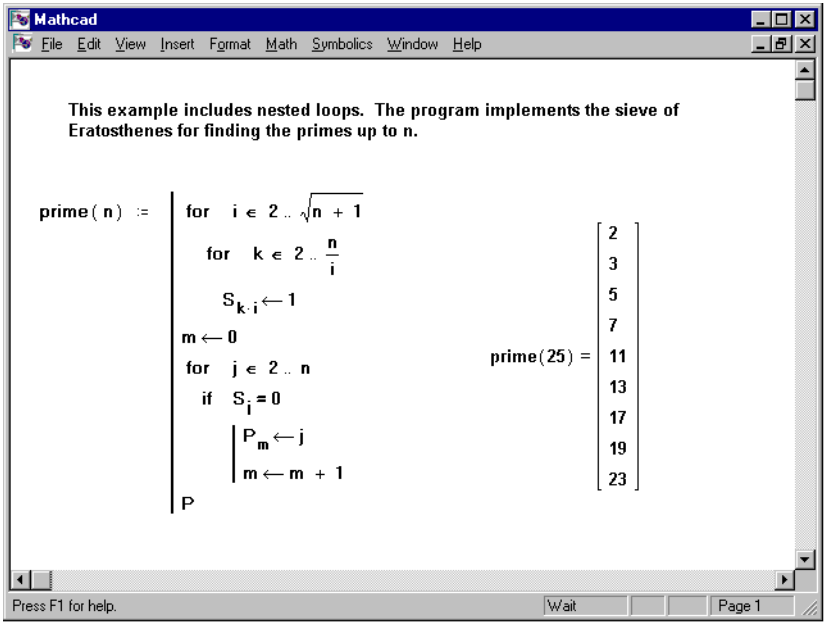


Figure 18-17: Using the sieve of Eratosthenes to find prime numbers.

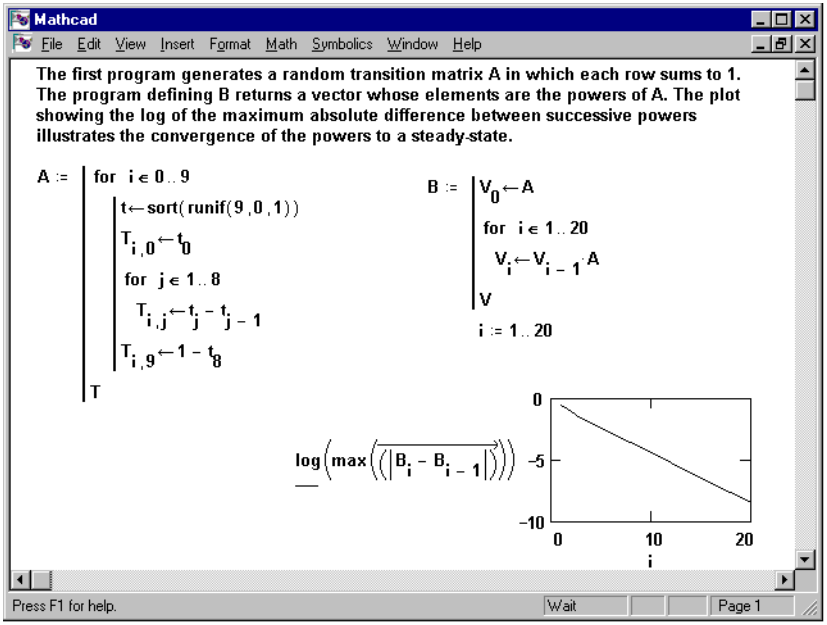


Figure 18-18: Powers of a random transition matrix.

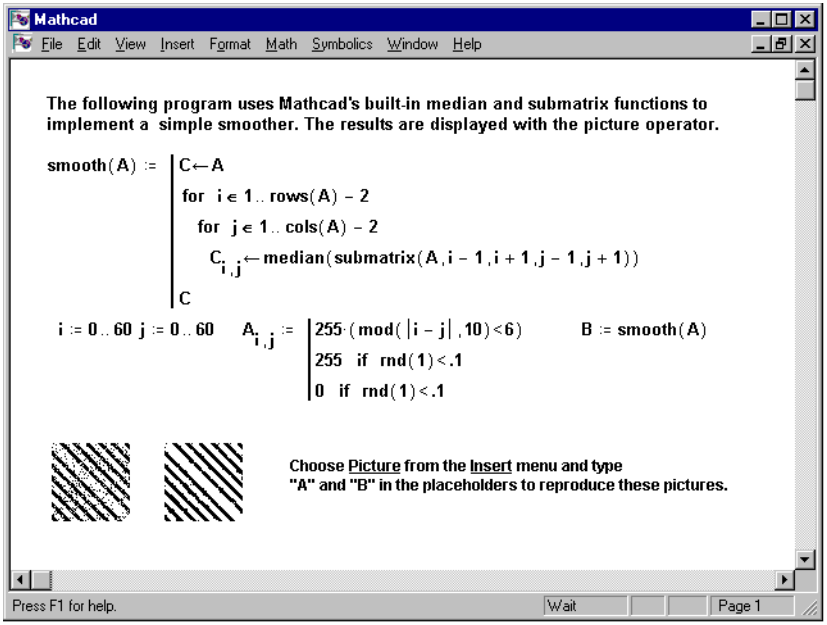


Figure 18-19: Smoothing a matrix.