Chapter 11
Range Variables

A range variable is a variable that takes on a range of values each time you use it. This chapter describes range variables and shows how to use them to perform iterative calculations, display tables of numbers, and facilitate the entry of many data values into a table.

The following sections make up this chapter.

Range variables

How to step through a range of numbers by defining a range variable.

Output tables

How to display a table of numbers.

Entering a table of numbers

How to use range variables to enter a table of numbers.

Iterative calculations

How to perform iteration with one or two range variables.

Seeded iteration

How to perform iteration when values in one step depend on the values in the previous step. Recursive techniques such as this provide the foundation for solving difference equations with Mathcad.

Vector or subscript notation

When to use the “vectorize” operator rather than subscripts.

Range variables

Iterative processes in Mathcad worksheets depend on *range variables*. Except for the way it's defined, a range variable looks just like a conventional variable. The difference is that a conventional variable takes on only one value. A range variable, on the other hand, takes on a range of values separated by uniform steps. For example, you could define a range variable to go from –4 through 4 in steps of 2. If you now use this range variable in an expression, Mathcad evaluates that expression five times, once for each value taken by the range variable.

Range variables are crucial to exploiting Mathcad's capabilities to their fullest. This section shows how to define and use range variables to perform iteration. For a description of more advanced iterative operations made possible by the programming operators in Mathcad Professional, turn to Chapter 18, "Programming."

Defining and using range variables

To define a range variable, type the variable name followed by a colon and a range of values. For example, here's how to define the variable *j* ranging from 0 to 15:

- Type **j** and then press the colon key (**:**). The empty placeholder indicates that Mathcad expects a definition for *j*. At this point, Mathcad does not know whether *j* is to be a conventional variable or a range variable.

- Type **0**. Then press the semicolon key (**;**). This tells Mathcad that you are defining a range variable. Mathcad shows the semicolon as two periods ".." to indicate a range. Complete the range variable definition by typing **15** in the remaining placeholder.

This definition indicates that *j* now takes on the values $0, 1, 2 \ldots 15$. To define a range variable that changes in steps other than 1, see the section "Types of ranges" on page 222.

Once you define a range variable, it takes on its complete range of values *every time you use it*. If you use a range variable in an equation, for example, Mathcad must evaluate that equation once for each value of the range variable.

You must define a range variable exactly as shown above. There must be:

- a variable name on the left,
- either a ":=" or a "≡" in the middle, and
- a valid range on the right.

In particular, you *cannot* define a variable in terms of a range variable. For example, if after having defined *j* as shown you now define $i := j + 1$, Mathcad assumes you are

trying to set a scalar variable equal to a range variable and marks the equation with an appropriate error message.

One application of range variables is to fill up the elements of a vector or matrix. You can define vector elements by using a range variable as a subscript. For example, to define x_j for each value of *j*:

- Type **x[j:j^2**[**Space**]**+1**.

$$x_j := j^2 + 1$$

Figure 11-1 shows the vector of values computed by this equation. Since *j* is a range variable, the entire equation is evaluated once for each value of *j*. This defines x_j for each value of *j* from 0 to 15. The effect is exactly the same as if you had typed

$$x_0 := 0^2 + 1$$
$$x_1 := 1^2 + 1$$
$$\vdots$$
$$x_{15} := 15^2 + 1$$

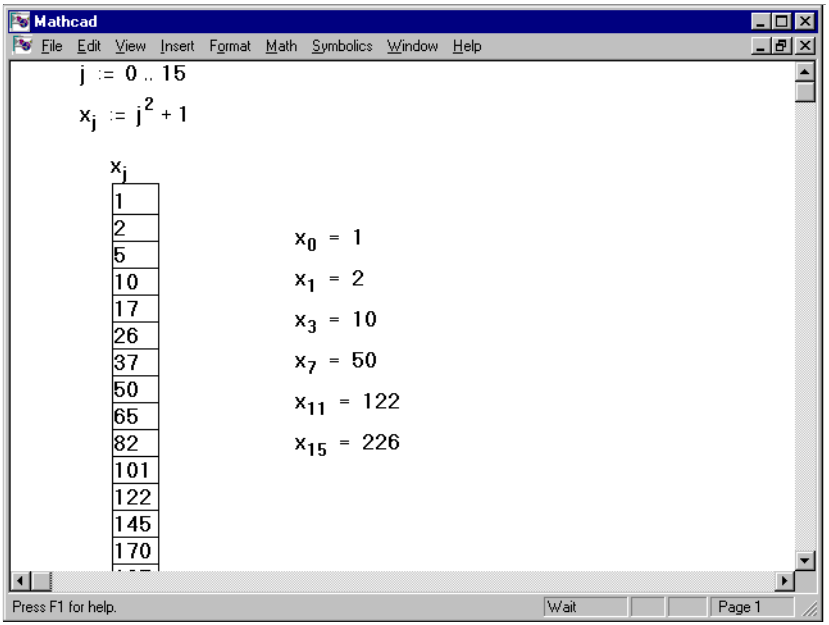


Figure 11-1: Using a range variable to define the values of the vector x.

To understand how Mathcad computes with range variables, keep in mind this fundamental principle:

If you use a range variable in an expression, Mathcad evaluates the expression once for each value of the range variable.

This principle sums up the difference between expressions with and without range variables. Expressions that involve no range variables have only one value. Expressions that involve range variables take on many values, one for each value of each range variable.

If you use two or more range variables in an equation, Mathcad evaluates the equation once for each value of each range variable. The section "Iterative calculations" on page 228 discusses this in more detail.

Mathcad takes longer to compute equations with ranged expressions since there are many computations for each equation. While Mathcad is computing, the mouse pointer changes its appearance. To learn how to interrupt a calculation in progress, see the section "Interrupting calculations" on page 152.

Types of ranges

The definition of *j* in the previous section, ranging from 0 to 15, is the simplest type of range definition. Mathcad permits range variables with values ranging from any value to any other value, using any constant increment or decrement.

To define an arbitrary range variable, type an equation of this form:

k:1,1.1;2

This appears in your document window as:

$$k := 1, 1.1 .. 2$$

In this range definition:

- The variable *k* is the name of the range variable itself. It must be a simple name. No subscripts or function definitions are allowed.
- The number 1 is the first value taken by the range variable *k*.
- The number 1.1 is the second value in the range. Note that this is *not* the step size. The step size in this example is 0.1, the difference between 1.1 and 1. If you omit the comma and the 1.1, Mathcad assumes a step size of one in whatever direction (up or down) is appropriate.
- The number 2 is the last value in the range. In this example, the range values are constantly increasing. If instead you had defined $k := 10 .. 1$, then *k* would count down from 10 to 1. Even if the third number in the range definition is not an even number of increments from the starting value, the range will not go beyond it. For example, if you define $k := 10, 20 .. 65$ will take values 10, 20, 30 . . . 60.

You can use arbitrary scalar expressions in place of 1, 1.1, and 2. However, these values must always be real numbers. Complex numbers do not make sense in range variable

definitions because given two complex numbers, there is an infinite number of paths connecting them. Figure 11-2 shows the results of various range variable definitions.

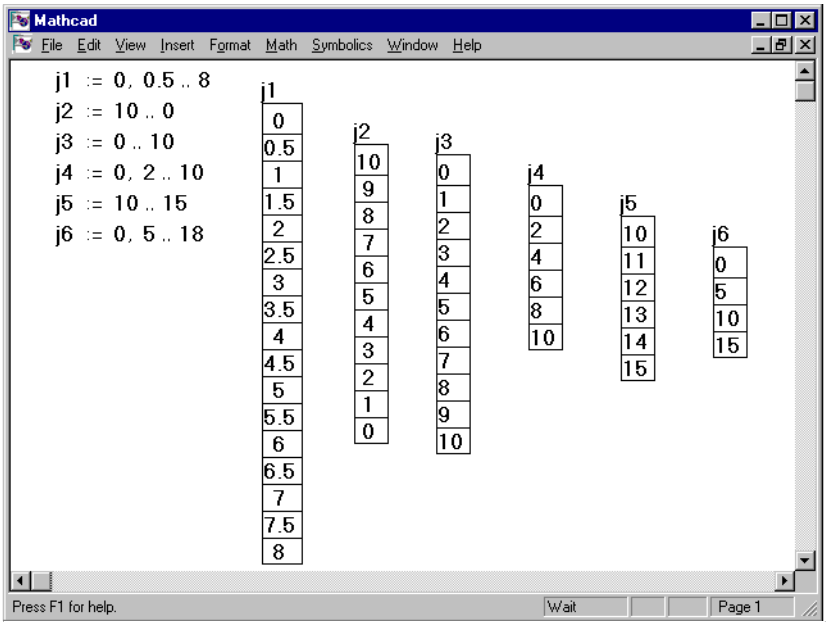


Figure 11-2: Some valid definitions for range variables.

Note that if you use a fractional increment for a range variable, you will not be able to use that range variable as a subscript. This is because subscripts must be integers.

Output tables

Whenever you type "=" after an expression involving range variables, Mathcad shows the computed values in an *output table*. Figure 11-2 shows the values of several range variables displayed as output tables.

Figure 11-3 shows some output tables for slightly more complicated expressions involving range variables.

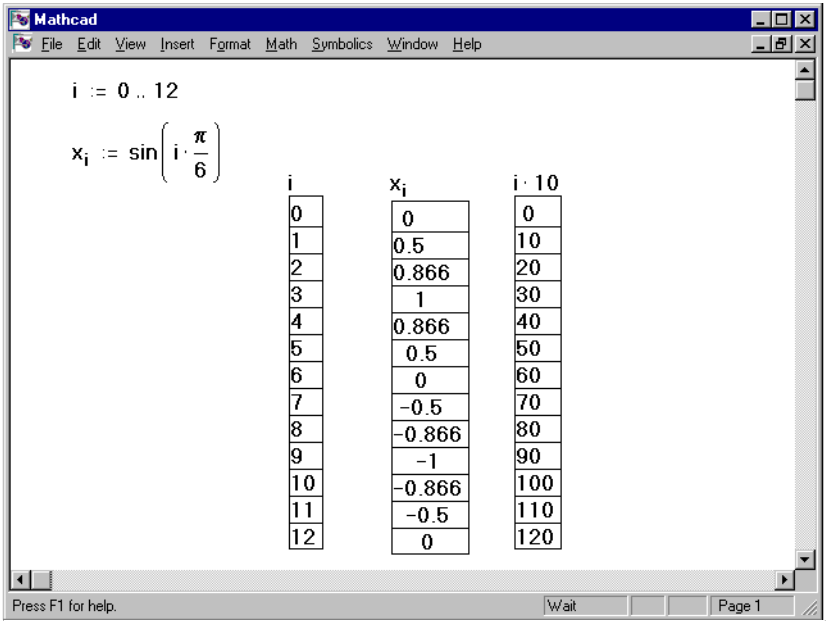


Figure 11-3: Typing "=" after an expression with range variables gives an output table.

To create the three tables in Figure 11-3, first define the range variable. Then type these equations:

i=
x[i=
i*10=

Whenever you type an expression followed by "=" Mathcad displays:

- a number, if the result is a scalar (a single number).
- a vector or a matrix, if the result is a vector or a matrix *and* the expression to the left of the "=" contains no range variables.
- a table like that shown in Figure 11-3 if the expression to the left of the "=" contains range variables.
- A scrolling output table if the result is a vector or a matrix, the expression to the left of the "=" contains no range variables, *and* the result has more than nine rows or columns. Scrolling output tables are discussed in the section "Displaying vectors and matrices" in Chapter 10.

Since both **x=** and **x[i=** display the same numbers, you can think of tables as another way of viewing the contents of a vector. Tables are particularly convenient for viewing selected parts of a vector. For example, if you've defined a vector **v**, you can view every other element of that vector by typing:

$$i := 0, 2 .. \text{last}(\mathbf{v})$$

$$v_i =$$

Here are some facts about output tables in Mathcad:

- Mathcad shows only the first 50 values of a ranged expression in a table. For example, even if *i* ranges from 1 to 100, typing **`i^2=`** will show only the values from 1^2 up to 50^2 in a table. To see more than 50 values, use several range variables and several tables. You could, for example, define *j1* from 1 to 50 and *j2* from 51 to 100, and then show tables for **`j1^2=`** and **`j2^2=`**, side by side.
- To format the numbers in a table, click in the table and choose **Number** from the **Format** menu. Then specify your formatting preferences in the dialog box as you would for an equation with a single numeric result. For more information on number formatting, see Chapter 6, "Equation and Result Formatting."
- There are three ways to show the values in a vector. If you use a vector name *with* a subscript like $x_j =$, Mathcad shows an output table. If instead you type a vector name *without* a subscript like $x =$, Mathcad shows the vector as a vector rather than as an output table. If you type a vector name without a subscript and the vector has more than nine elements, you see a scrolling output table as described in the section "Displaying vectors and matrices" in Chapter 10. Keep in mind that these are just three different ways of looking at the same thing: an ordered collection of numbers.
- You can't use units with a table as you would with a single scalar answer. If the results in a table have dimensions, Mathcad shows dimensions *on each value in the table*. To avoid this, divide the ranged expression by the units. See Figure 11-4.

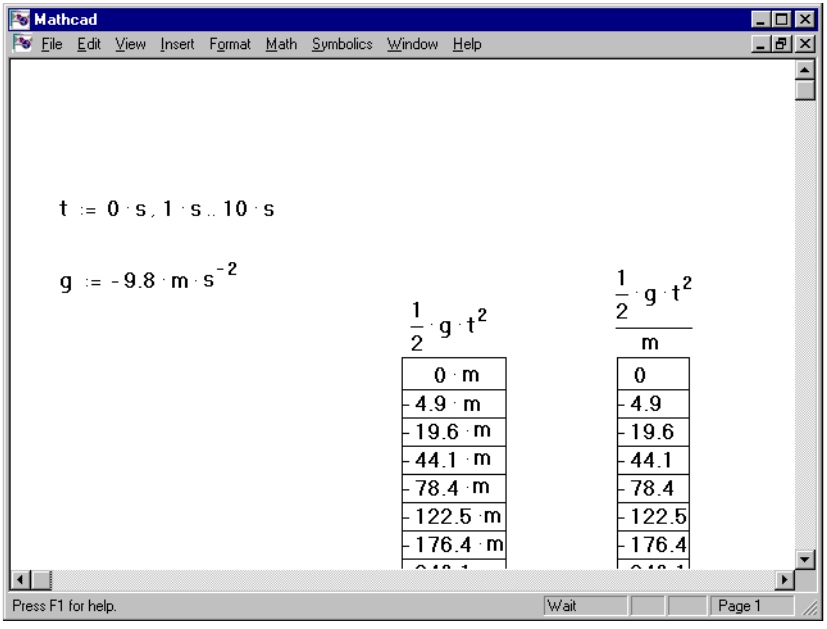
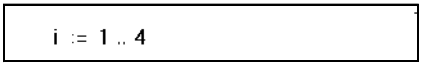


Figure 11-4: Units in a table.

Entering a table of numbers

When you enter a table of numbers, you are actually assigning elements to a vector. This section discusses how to do this using *input tables* and range variables. To enter an input table, enter a definition with a subscripted variable on one side and a sequence of values separated by commas on the other. For example:

- To define *i* to run through four values, type **i:1;4**. Note that *i* must take integer values only. Otherwise it can't be used as a subscript in the next step.



- Click in a new spot and type **x[i:** The placeholder indicates that Mathcad is expecting a value for x_1.

- Type **3** and press the comma key. Mathcad shows another placeholder to indicate that Mathcad now expects a value for x_2.

- Type **5,15,20** to supply values for x_2, x_3, and x_4.

Once you have created an input table, you can do any of the following:

- **Insert a value in the middle of a table**. Click on the value immediately above wherever you want to insert the new value. Then type a comma. Under the selected value in the table, Mathcad creates a placeholder surrounded by a box. To enter another number, just type it into this placeholder.
- **Extend the table to hold additional values**. Click on the last value in the table and follow the steps above for inserting a value in the table.
- **Replace or delete a value from the table**. Place the value you want to replace or delete between the two editing lines and choose **Cut** from the **Edit** menu. Mathcad replaces the value with an empty placeholder. Type a new value in this placeholder to replace the old one. To delete the value completely and decrease the array length by one in the process, backspace over the placeholder.

Some notes about input tables:

- Each value in an input table must be either a number or an expression that returns a number, the name of an array or an expression that returns an array. Expressions

involving range variables and expressions created by using the **Matrix** command on the **Insert** menu are not permitted.

- All expressions in an input table must have the same dimensions if any. If you want each expression to be in meters, for example, you may have to include the abbreviation for meters in each table entry. A shortcut is to enter the numbers without units and redefine the vector with units by typing something like $x := x \cdot \mathrm{m} / \mathrm{sec}^2$.
- An input table ordinarily has one entry for each value of the range variable used in defining it. If the table has too few entries, Mathcad will define only as many values as are present. If the table has too many entries, the extra entries will be ignored.
- Input tables assign values *only to those elements specified by the range variable*. If in the previous example, the range variable definition had been $i := 10, 20 .. 40$, Mathcad would have assigned values to x_{10}, x_{20}, x_{30} and x_{40}. Mathcad would then pad the unassigned entries, namely x_0 through x_9, x_{11} through x_{19}, and so on, with zeros. You will see these zeros if you display the vector by typing "**x=**." It is possible to inadvertently create large tables this way.
- Input tables are limited to 50 elements. If you want to enter more than 50 elements, enter them using several tables. You could, for example, define *j1* from 1 to 50 and *j2* from 51 to 100, type **x[j1:** followed by the first fifty numbers, then type **x[j2:** followed by the second fifty numbers.
- When confronted with a very large number of data values, consider reading them in from a data file stored on your disk as an alternative to typing them in with input tables. Chapter 19, "Data Management," discusses this in more detail.

Figure 11-5 shows some input tables. Note how typing **x=** and **y=** displays the elements of *x* and *y* in vector form. Mathcad ignores the last number in the input table for *y* since this entry would have index 5 and the range variable *i* stops at 4. Contrast this with typing **x[i=** as shown in Figure 11-3.

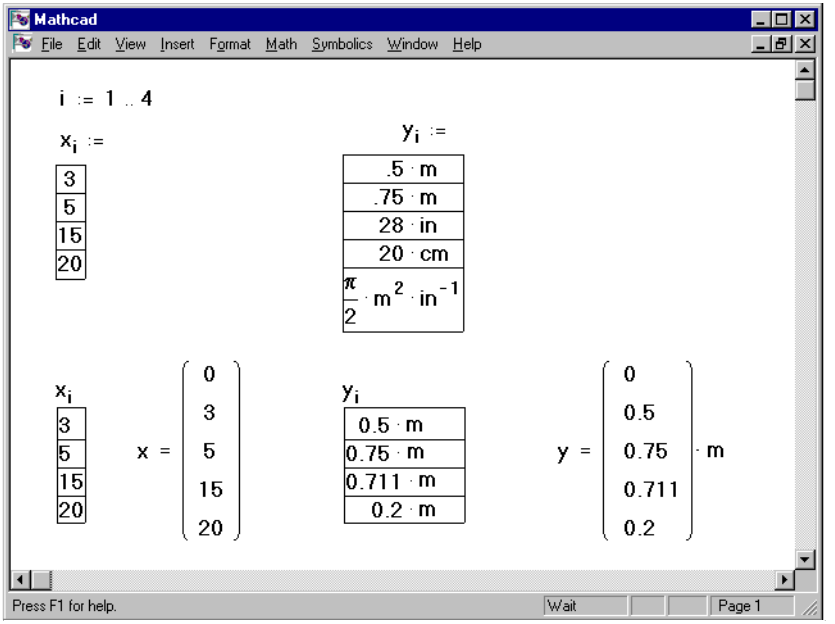


Figure 11-5: Input tables.

Note that the first element of both vectors is zero. This is because Mathcad's array origin is set to zero by default. Since the range variable *i* starts at 1, the zeroth element is never explicitly defined. In the absence of an explicit definition, Mathcad assumes a value of zero.

Iterative calculations

This section shows how to use range variables to perform iteration.

Iteration over a range

The simplest kind of iteration in Mathcad is just a generalization of scalar calculations. Any calculation you can perform once, you can perform over a range of values.

For example, suppose you want to create a list of *x* and *y* values for points on the polar curve $r = \cos(\theta) + 1$. The basic idea is as follows:

- θ should take on values between 0 and 2π.
- For each θ, the corresponding *r* is given by $r = \cos(\theta) + 1$.
- For each *r* and θ, there is a corresponding *x* and *y* given by $x = r \cdot \cos(\theta)$ and $y = r \cdot \sin(\theta)$.

The strategy for solving this problem is simple: create a range variable *i* and then compute θ, *r*, *x*, and *y* for each value of *i*. The formula for θ_i defines θ to run from 0 to 2π in steps of $2\pi/N$. To create the other formulas, just put the subscript *i* on each variable in the formula. Figure 11-6 shows the result.

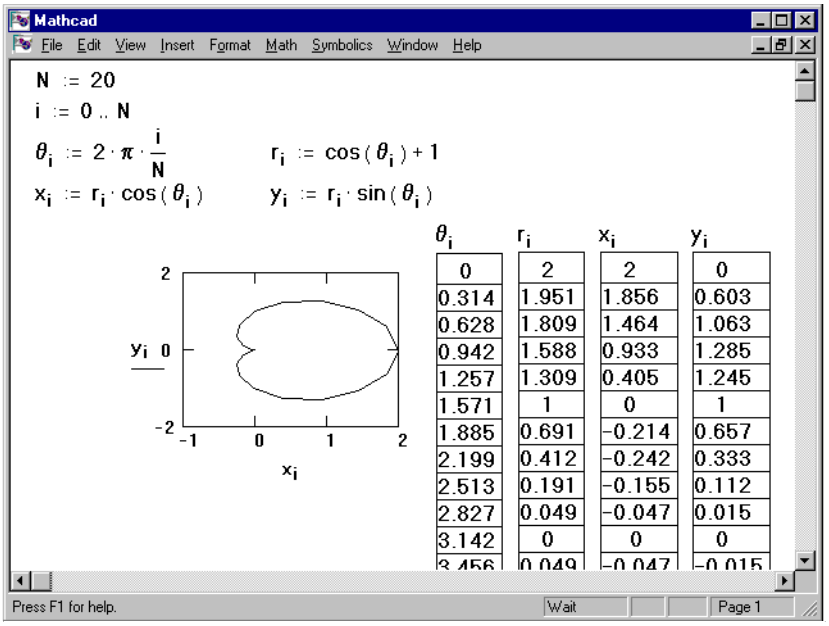


Figure 11-6: Using iteration to create a draw a polar curve using an X-Y plot.

Notice that in this example *i*, not θ, is defined as the range variable. Since *i* takes on only whole-number values, it is a valid subscript. On the other hand, θ takes on fractional values. It therefore cannot be used as a subscript. In many cases, you can avoid this extra step by using functions instead of vectors. Figure 11-7 shows how to generate the cardioid shown in Figure 11-6 with functions instead of vectors.

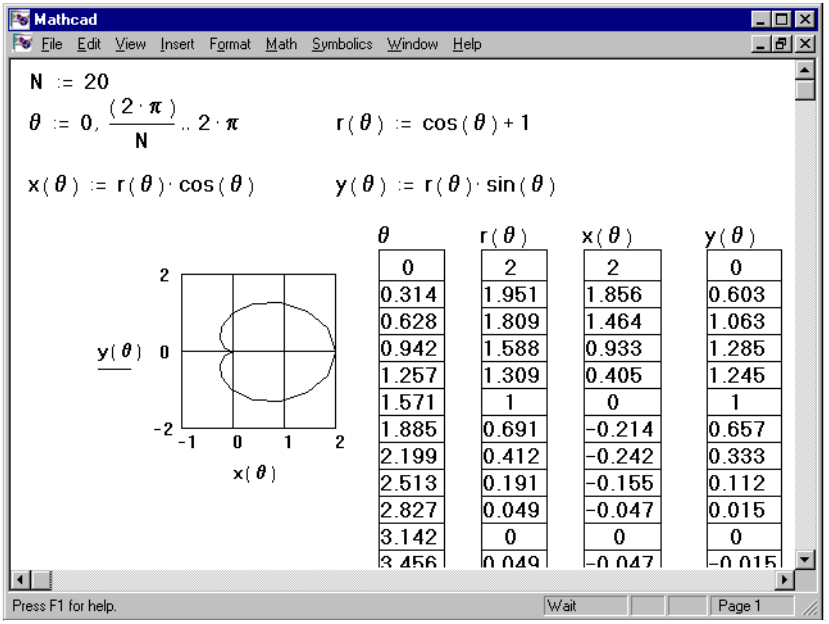


Figure 11-7: Using a function to do the same thing as shown in Figure 11-6.

By using vector notation and the vectorize operator, you can eliminate the use of a subscript in the last three equations in Figure 11-6. Figure 11-8 shows an example of how to do this.

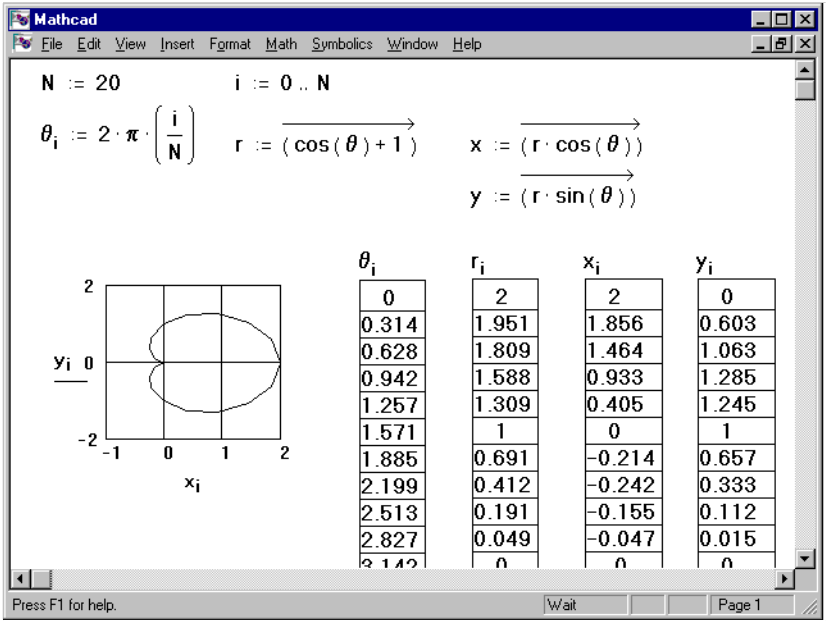


Figure 11-8: Using the vectorize operator to create a polar plot.

Equations that use vector notation instead of subscripts typically compute much more quickly. For more information, see Chapter 10, “Vectors and Matrices.”

Multiple range variables and double subscripts

If you use two range variables in an equation, Mathcad runs through each value of each range variable. This is useful for defining matrices. For example, to define a 5×5 matrix whose *i,j*th element is $i + j$, type these equations:

```
i:0;4
j:0;4
x[i,j:i+j
```

Note that you don't need to type [**Space**] to leave the subscript in this case. Typing **:** leaves the subscript and creates the definition symbol.

Figure 11-9 shows the result of typing the above equations. It is usually best to display the matrix in the form shown in Figure 11-9. If instead of typing **x=** you were to type **x[i,j=**, Mathcad would show one long output table with 25 numbers. Such a table is often difficult to interpret. A similar problem arises when you use a pair of range variables in a graph.

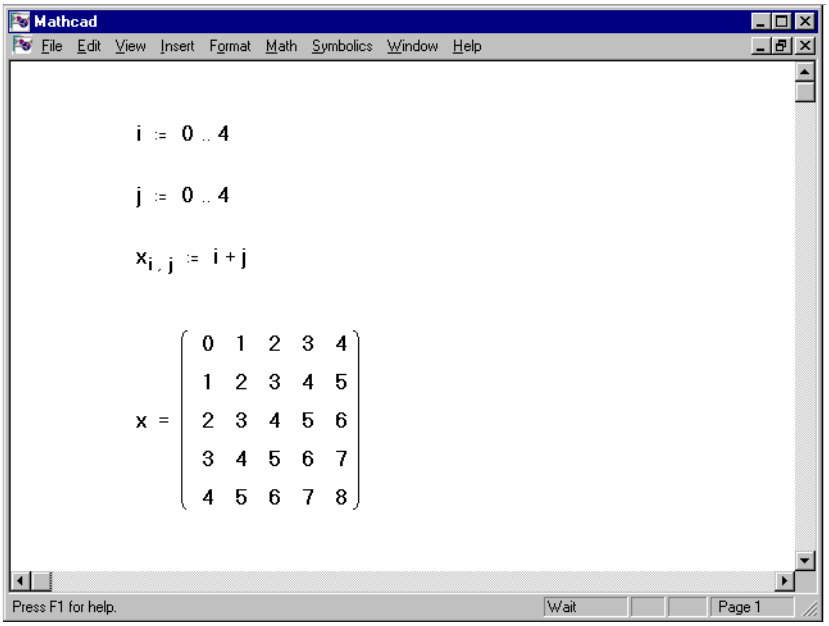


Figure 11-9: Defining a matrix.

The $x_{i,j}$ equation is evaluated for each value of each range variable, for a total of 25 evaluations. The result is the matrix shown at the bottom, with 5 rows and 5 columns. The element in the *i*th row and *j*th column of this matrix is $i + j$.

Note that if the two range variables have *m* and *n* values, respectively, then an equation using both range variables will calculate $m \cdot n$ results. If you try to use two range

variables in an output table, Mathcad will show these $m \cdot n$ results in a long table with one entry for each result. If you use two range variables in a graph, Mathcad will plot one point for each of the $m \cdot n$ results.

Seeded iteration

Seeded iteration is a recursive technique for solving difference equations such as those that arise in compound interest problems, Markov processes, and many state variable equations. It can also be used for obtaining approximate solutions for certain differential equations. In a seeded iteration, you specify the first element of an array and then compute successive elements based on the first element. This section describes three types of seeded iteration: iterating a single variable, iterating multiple variables, and iterating a vector.

Seeded iteration on one variable

The classical method for estimating square roots arithmetically is as follows:

- To find $\sqrt{a}$, begin with a guess value.
- Compute a new guess based on the old guess, with this formula:

$$NewGuess = \frac{OldGuess + a/OldGuess}{2}$$

- Continue until the guesses converge to an answer.

Figure 11-10 shows how to implement this method in Mathcad.

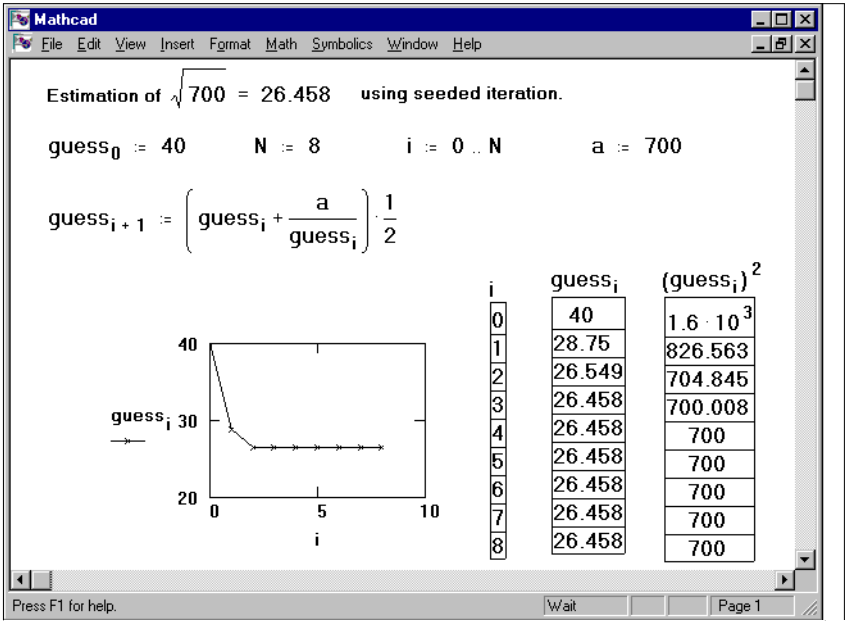


Figure 11-10: Using seeded iteration to estimate a square root.

The important characteristics of this example are:

- The seed value is defined as the zeroth element of the array, $guess_0$.
- Each $guess_{i+1}$ is defined in terms of a previously computed element.

It is this dependence of array elements on previously computed array elements that distinguishes seeded iteration from the more straightforward iteration discussed in the previous section.

Seeded iteration on several variables

You can use Mathcad's vector notation to iterate several variables simultaneously. This variation on simple seeded iteration is a powerful method for solving a system of simultaneous difference equations.

When you iterate several variables, each step computes the value of the variables from all of their previous values. You can't accomplish this with several equations because when Mathcad sees an equation with range variables, it attempts to evaluate it for *each* value of the range variable before going on to the next equation. You must, therefore, create one equation that performs all the iterations simultaneously.

For example, consider an infection model with four variables: *i* for the number of individuals infected, *s* for the number susceptible, *d* for the number deceased, and *r* for the number recovered and hence immune. The four equations that relate these four variables over time are:

$$
\begin{aligned}
i_{t+1} &= 0.0001 \cdot s_t \cdot i_t \\
s_{t+1} &= s_t - 0.0001 \cdot s_t \cdot i_t \\
d_{t+1} &= d_t + 0.55 \cdot i_t \\
r_{t+1} &= r_t + 0.45 \cdot i_t
\end{aligned}
$$

Figure 11-11 shows how to perform a simultaneous iteration using these equations.

The single most important thing about this example is that all the $t + 1$ subscripts are on the left-hand side of the matrix equation. The right-hand side contains only the subscript *t*. Mathcad evaluates all the expressions on the right-hand side before performing any assignments to the left-hand side. This means that nothing on the right-hand side can depend on something on the left-hand side.

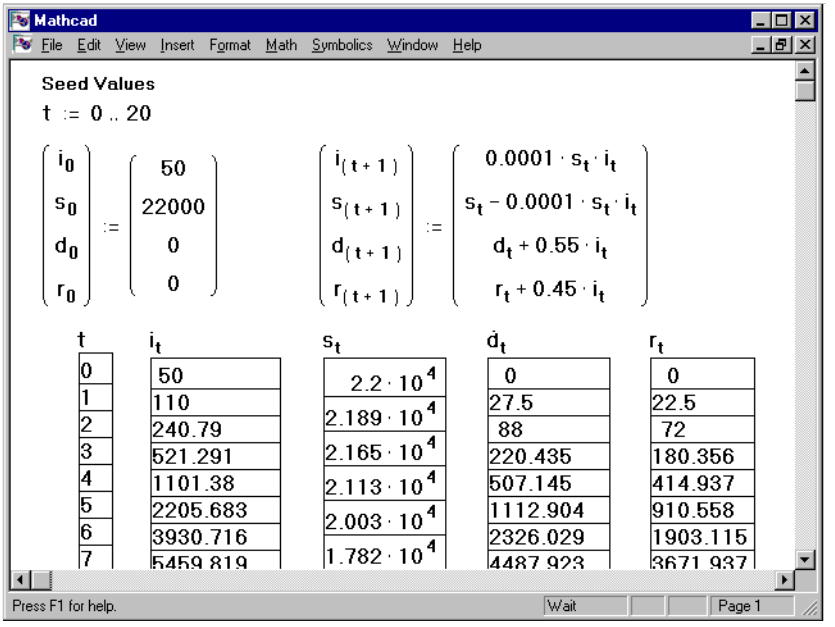


Figure 11-11: Simultaneous iteration to model an infection.

Seeded iteration on a vector

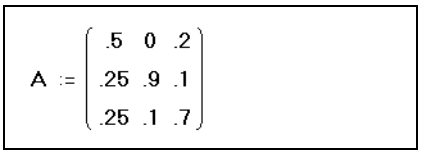
You can also perform seeded iteration starting with a vector and computing a new vector each time. This type of iteration uses a seed vector and Mathcad's superscript operator.

A *Markov process* is an example of a problem that involves iteration on a vector. A Markov process begins with a vector **v** that represents the starting values of some quantities, for example, the number of voters planning to vote for different candidates, the number of trucks at regional offices of a truck rental company, or the market share of different companies. Each step in the Markov process computes a new vector by multiplying the previous vector by a "state transition" matrix **A**.

Figure 11-12 shows how to set up a Markov process. This technique uses *superscripts* to index an entire column of a matrix at once. To create a superscript, press [**Ctrl**]**6**. This generates a placeholder between angle brackets: **<>**.

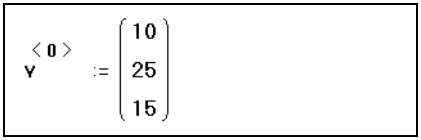
Here's how to enter the equations in Figure 11-12:

- First, define the state transition matrix **A**. Type **A**, press the colon key (**:**), and create a 3×3 matrix. To create a matrix, choose **Matrix** from the **Insert** menu.

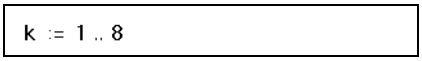


- Click to the right of the matrix and type **v**. Then press [**Ctrl**]**6**. Type **0** in the placeholder for the superscript.

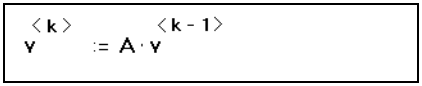
- Now complete the definition of the initial vector. First press the colon key (**:**). Then choose **Matrix** from the **Insert** menu. Specify that you want to create a matrix with three rows and one column. Then fill in the matrix entries.



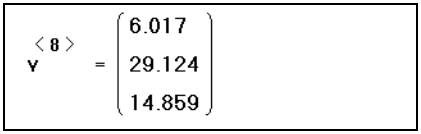
- Type **k:1;8**. This defines a range variable *k* to count eight iterations.



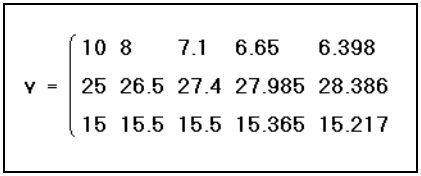
- To define the *k*th vector in terms of the $(k-1)$ st, type **v**[**Ctrl**]**6** **k**. Then type a colon (**:**) for the definition symbol. Complete the equation by typing this expression after the definition symbol: **A*v**[**Ctrl**]**6** **k-1** .



- To see the eighth (last) column of the matrix, type **v** [**Ctrl**]**6** **8** **=** .



- To see all the vectors as columns of a matrix, type **v=** . Note that not all the columns are displayed in the picture to the right.



The superscript operator actually retrieves or defines one column in a matrix. When you define $\mathbf{v}^{\langle k \rangle}$ in terms of $\mathbf{v}^{\langle k-1 \rangle}$, you are actually defining each column of a matrix in terms of the preceding column. The last equation in Figure 11-12 shows the matrix composed from these columns.

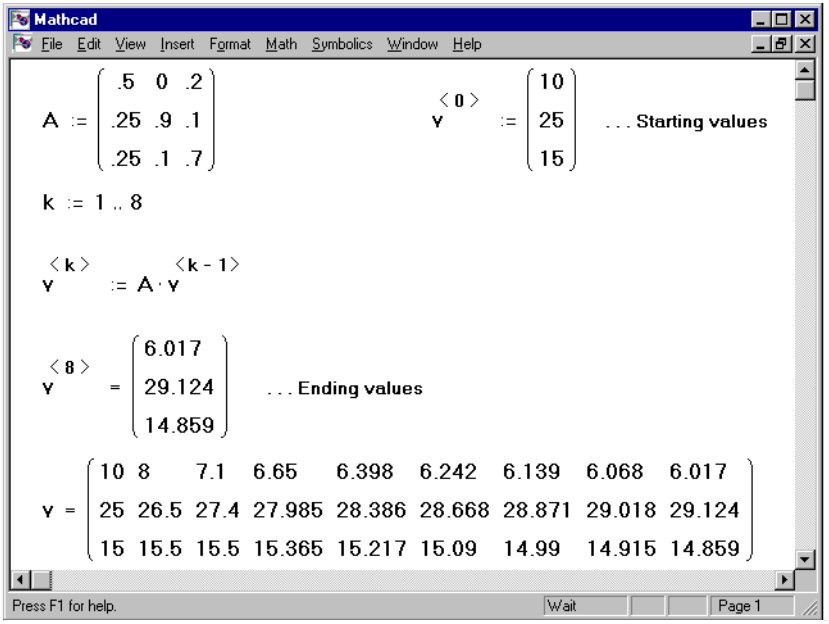


Figure 11-12: Iterating a vector to model a Markov process.

Vector or subscript notation

This chapter has shown many examples using subscript notation and range variables. Chapter 10, "Vectors and Matrices," showed many examples using vector notation without subscripts. This distinction is important. If you use subscripts when they are not required, or vice versa, you probably won't get the answer you're looking for.

Subscripts refer to individual array elements. When you use range variables as subscripts like $M_{i,j}$, Mathcad runs through the individual elements of the array one at a time.

Without subscripts, the variable name refers to the whole array.

Here are some rules of thumb for when to use subscripts:

- To refer to an individual array element, use numbers as subscripts. For example, to see matrix element (2,3), type **M[2,3=**.
- To refer to an array as a whole, use the array name without subscripts. (The term array means either a vector or a matrix.) The array name with no subscripts is appropriate for multiplying one matrix by another, applying the *mean* function to an array, or viewing a whole vector or matrix—all cases where you want to treat the array as a whole. For example, to view the whole matrix **M** in Figure 11-13, type **M=.**

- To refer to each of the array elements in succession, use the array name with a range variable subscript. This is useful when defining the matrix elements using some sort of formula. For example, to define the matrix **M** in Figure 11-13, type three equations:

```
i:0;3
j:0;3
M[i,j:i*j
```

Figure 11-13 shows some examples of using array names with and without subscripts.

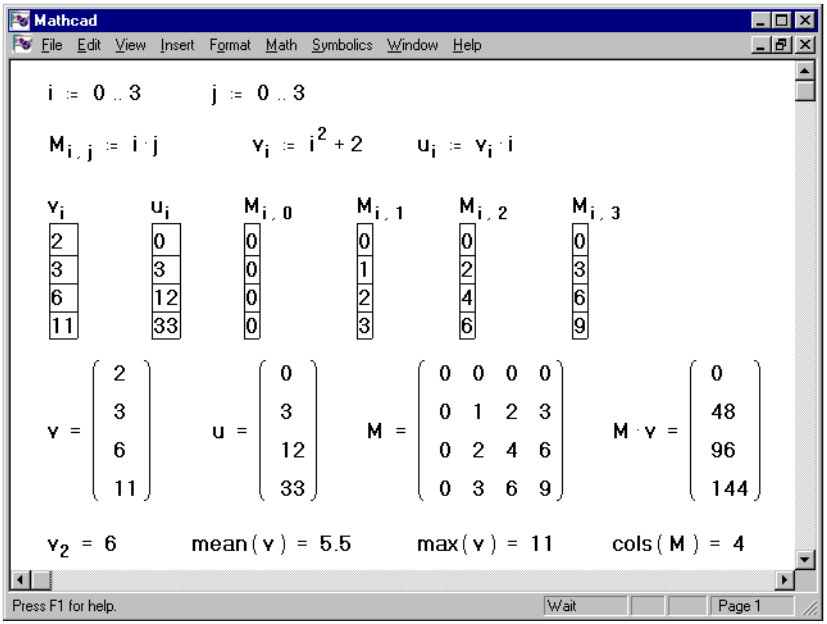


Figure 11-13: Array names with and without subscripts.

Iterative calculations with and without subscripts

It's often faster to use vector operations to perform iterative calculations than it is to do them element by element with a range variable. This is discussed in more detail in the section “Doing calculations in parallel” in Chapter 10. Unfortunately, not all iterative calculations can be done as vector operations. For example, the seeded iteration technique discussed earlier in this chapter cannot be done with vector operations.

To tell if an equation with subscripts could be rewritten using vector notation and the vectorize operator, check the following:

- If all the subscripts in the calculation are the same, then the calculation can probably be done quickly using vector operations. For example, consider this equation:

$$x_i := r_i \cdot \cos(\theta_i)$$

Since this equation contains no subscript other than *i*, it can be rewritten with the vectorize operator ([**Ctrl**]-) instead of with subscripts:

$$x := \overrightarrow{(r \cdot \cos(\theta))}$$

- If the subscripts in a calculation vary or if there is arithmetic involved in computing the subscript, then the calculation probably cannot be done using vector operations. For example, seeded iteration involves the subscripts *i* and *i*–1 in the same equation. Since the subscripts are not the same, and since the second subscript involves arithmetic, this calculation cannot be done using vector operations.
- If the range variable appears anywhere in an equation other than in a subscript, the equation cannot be written using vector operations. For example, the variable θ below cannot be defined using the vectorize operator. Although the range variable appears on the left side as a subscript, its presence in an expression on the right disqualifies the entire equation.

$$\theta_i = 0.1 \cdot i$$

Figure 11-14 shows the polar coordinate conversions from Figure 11-6 computed two ways: using subscripts and using vector operations. The second method is much faster in Mathcad.

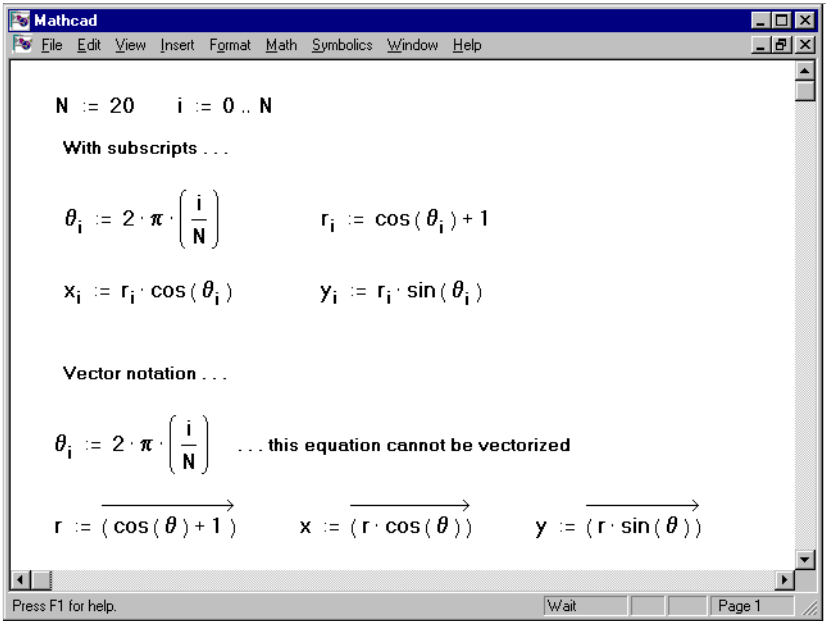


Figure 11-14: Changing an iterative process to vector operations.