Understanding Ranges, Sequences, and Vectors

Introduction

New Mathcad users are sometimes confused by the difference between range variables and vectors. This is particularly true considering that other applications use expressions like 0..3 or 1:4 to define vectors.

It can be confusing when you type something like the expressions below and get an error message.

$k := 0, 0.2 .. 1$ a range variable definition

$v := \sin(k)$ a vector of sine values

Since Mathcad handles a vector as the argument to sin, you may expect that Mathcad interprets k as the vector $(0 \;\; 0.2 \;\; 0.4 \;\; 0.6 \;\; 0.6 \;\; 0.8 \;\; 1)^T$. If you evaluate sin(k) and k, you do get vector looking results.

$\sin(k) =$

0
0.199
0.389
0.565
0.717
0.841

$k =$

0
0.2
0.4
0.6
0.8
1

The aim of this article is to explain the difference between range variables and vectors, and why the expression v := sin(k) fails.

Just to show that there is light at the end of the tunnel, here is the correct way of using range variables in this context.

$k := 0, 1 .. 5$ convert range variable to use integers only

$t_k := 0.2 \cdot k$ for each value of k, set the kth element of vector variable, t, to 0.2 times the corresponding value of k

$v_k := \sin(t_k)$ for each value of k, apply sine to the kth element of vector t and assign the resulting value to the kth element of the vector stored in the variable v

Many Mathcad functions work on complete vectors, including sine. You can take advantage of this by simply writing:

$v := \sin(t)$ apply sine to t and assign the resulting vector to v

Note that:

- A vector is a one-column array and can store numbers.
- A vector can only have integer indices.
- A range definition, or 'range' for short, describes how to generate sequential numbers, but is not itself a list of numbers.
- A range variable is a variable that stores a range definition..
- Mathcad treats range variables as an implicit instruction to loop over each value in the implied sequence.

This probably still seems puzzling, so let's examine variables, vectors, and ranges in more detail.

What are Mathcad vectors and range variables?

Vectors

In Mathcad, a **vector** is a single column array of data. The elements of a vector, in common with its parent structure, the array, can hold scalars, strings, function names, or other arrays (forming a "nested" vector). For example, you can create a vector using the function stack and assign it to a variable:

$v := \text{stack}(1, 2)$

$w := \text{stack}(1, \text{"a"}, 3)$

set x equal to w

$x := w$

add a vector to the end

$x_3 := v$

$$v = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \qquad w = \begin{pmatrix} 1 \\ \text{"a"} \\ 3 \end{pmatrix} \qquad x = \begin{bmatrix} 1 \\ \text{"a"} \\ 3 \\ \begin{pmatrix} 1 \\ 2 \end{pmatrix} \end{bmatrix}$$

There are some restrictions on the type of data that a vector can hold - for example, all elements must have the same units - but it is nevertheless a powerful tool for storing, accessing, and manipulating data. Indeed many Mathcad functions are explicitly designed to handle vectors.

You can extract the values of each element using the Mathcad array indexing notation v_n where n is an *integer* ranging from ORIGIN to last(v). Note that n must be an integer. Also note that a vector is a data type and not a variable; a variable in its turn can hold a vector, but isn't a vector itself.

Range Variables

A **range variable** is a variable that stores a range definition (or 'range' for short). When Mathcad encounters a range variable in an expression, it treats the range variable as a request to generate a set of numbers and to evaluate the expression for each one of those numbers in turn. In programming terms, a range variable is an implicit for loop. In the examples below, the range variable is on the left-hand side of the definition, and the corresponding range is on the right:

$k := 0..4$ $t := 0, \frac{1}{4}\pi .. \pi$

k =

0
1
2
3
4

t =

0
0.785
1.571
2.356
3.142

Both a range and a vector are data types. The key to the differences between a vector and a range is that a vector is meant for storing data, while a range is shorthand for a list of values. You can readily access each element of a vector, but you *cannot* get at the individual elements of a range variable (indeed, it doesn't make sense to try). They don't exist, and Mathcad generates an error message if you do try.

$k_2 =$

To understand range variables a little bit better, let's look in detail at ranges and the related concept of a sequence, starting with the sequence.

Sequences

A sequence is a comma-separated list of values, where a value in this instance is a scalar, string, array, sequence, range, function name or an expression that evaluates to one of these types of value (for example an evaluated function or a range variable).

The following are valid sequences:

$0, 0.1, 0.2$ $1, "a", "b", "c", 2$ (the second sequence is valid in Mathcad 13 and 14, but not in Mathcad 11)

Note that a sequence is not a vector, and it isn't possible to pick out a single element from within it. A sequence also differs from a range variable in that it can't be assigned to a variable, which makes statements like the one below flag up an error if it is used.

$s := 0, 0.1, 0.2, 3$

$2s =$

Sequences are commonly used as arguments to functions or in for loops, which will be discussed in a future article.

Now let's turn back to the range.

Ranges

A range is an abbreviated specification for a uniformly spaced real numeric sequence. The syntax for a range is start.. end where end is a scalar and start is either a scalar or a 2-element sequence. A range can define either a set of increasing values or a set of decreasing values.

For example, 0..9 is an ascending range that starts with a scalar, while 0, -0.1..-1 is a descending range that starts with a sequence.

When Mathcad evaluates a range beginning with a sequence, it interprets the second number in the sequence as the *next* number in the sequence.

Note that the second value in the expression for the range is *not* the step size. The step size is the difference between the first two values. If the range starts with a scalar, then Mathcad assumes a step size of 1 in the appropriate direction. If the step is known, then the range can be readily defined as start, start + step.. end. For example,.

$$1, 1 + 0.25 .. 2$$

A range can also form part of a sequence, so 1, 2, (0, 0.25 .. 1) is a valid sequence. Note that the parentheses are mandatory to delimit the range from the rest of the sequence. However, because Mathcad restricts a range to real scalars, "a" .."c" is not a valid range (these are strings not scalars).

Using Range Variables

Range variables have a number of uses within Mathcad, from quickly evaluating an expression over a range, to creating recursive sequences.

Show and Tell

The simplest use of range variables is in calculating an expression for a given list of numbers and displaying the results. The syntax is straightforward "expression =," where the expression contains one or more range variables. When Mathcad encounters this form, it evaluates the expression for each combination of values covered by the range variables. Mathcad then displays the results in table form.

In the example below, there are two range variables, i and j, that hold the ranges 4 .. 5 and 1 .. 2 respectively. Evaluating the expressions $i^2 =$ and $j^2 =$ causes Mathcad to calculate the squares of 4 then 5 for i, and 1 then 2 for j.

$i := 4 .. 5$ $j := 1 .. 2$

$i^2 =$	$j^2 =$
16	1
25	4

In the combined expression $i^2 + j^2$, Mathcad fixes the value of j (the second range variable) and iterates over i, then Mathcad takes the next value of j and iterates over i again. Reversing the order of declaration reverses the order of evaluation.

$i^2 + j^2 =$

17
26
20
29

$4^2 + 1^2$
$5^2 + 1^2$
$4^2 + 2^2$
$5^2 + 2^2$

$j^2 + i^2 =$

17
20
26
29

You can use any type of range variable in this way, whether it has integer steps or fractional ones.

The sin(k) example at the beginning of this article has fractional steps. Remember that k isn't a vector - it's a range variable. When you evaluate sin(k), Mathcad iterates through k and displays a result for each value of k. The results appear in a grid to show they are related, but that grid does not represent an array - merely a visual representation of a list of individual results.

Pointing the Way

Evaluating an expression as described above is often what's needed. However, there are also many occasions where you may want to make use of these results later on in a worksheet.

Fortunately, Mathcad provides a convenient way to do this by allowing a range variable to serve as an array index. Consider the following pair of expressions:

$k := 0 .. 3$ range variable definition

$v_k := k^3$

- The first expression is the familiar range variable definition.
- The right-hand side of the second expression, k^3 contains the range variable and calculates the cube of each value in the range. If you write $k^3 =$ you would see a table of cubes.
- The left-hand side of the second expression, v_k stores the values of k^3. It stores the result of evaluating k^3 in the kth element of the vector v.

$v^T = (0 \quad 1 \quad 8 \quad 27)$ vector creation for storing values of k^3

Evaluating v^T reveals that you have created a vector, because the transpose operator T only works on arrays and generates an error if you apply it to a range variable.

$k^T =$

This mechanism extends to matrices as well; by using the previous definitions for i and j, you get

$A_{i-4,j} := i^2 + j^2$ $\quad A = \begin{pmatrix} 0 & 17 & 20 \\ 0 & 26 & 29 \end{pmatrix}$

Here's another point about using range variables that helps distinguish them from vectors..

The only thing you can do to a range variable is define one to be equal to another one; Mathcad does not allow even basic arithmetic operations on a range variable.

$n := i$

$n := i + 1$

You can't add 1 to an existing range variable because the range is not a numeric type, but a specification for how to produce a set of numbers. It makes as little sense to add 1 to a range as it does to add 1 to a string.

The definition of A above has the expression i-4 in the row index so you may wonder, why Mathcads allow subtraction on the range variable i?. The answer is that Mathcad treats a range variable in an expression as an instruction to evaluate that expression for each value that the range specifies. In the definition above, i does not refer to the whole range variable, but to the values that it generates. Hence, the index expression i-4 is an instruction to Mathcad to take the "current" value of i and subtract 4 from it.

Now that you can calculate an array index, you can generate results that depend upon previously calculated values. A good example of this is the Fibonacci sequence (where "sequence" is used in its general mathematical sense).

The Fibonacci sequence is "1,1,2,3,5,8,13,21 ...", where each number in the sequence is the sum of the preceding two elements. To start the sequence, define the first two elements to both be 1. For the nth number, the indices of the two preceding number are simply n-1 and n-2. This is all you need to generate a vector holding the Fibonacci sequence.

$fib_0 := 1$ first number in the sequence

$fib_1 := 1$ second number in the sequence

$n := 2..7$ range variable to create the next 6 numbers

$fib_n := fib_{n-1} + fib_{n-2}$ make the nth number equal to the sum of the preceding two numbers

$fib^T = (1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21)$ show the resulting vector (transposed to make it easier to read)

If you hadn't defined the first two numbers in the sequence, Mathcad would either have automatically set them to zero (Mathcad 11) or flagged the error, invalid index (Mathcad 13 and Mathcad 14).

Here are are two more examples of common errors involving array indices. The first occurs when you write

$r := 0, 0.5 .. 1$

$w_r := r$

This case fails because an array index must be an integer, whereas r takes on fractional values. This second case looks as though it should work, but fails for a different reason.

$i := \text{stack}(0, 1, 2, 3, 4)$ create a vector of integers

$w_i := i$ try to use them to as indices into another vector

This case fails because Mathcad only knows how to handle array indices that are integers, which a vector isn't, and does not iterate over vectors, since they're just a *normal* data type.

Finally, there is one important piece of syntax to remember. When defining a variable in terms of a range variable, the range variable ***must*** appear on the left-hand side of the definition. Mathcad raises an error if it sees a range variable on the right, but not on the left.

Looking back at the original example,

$k := 0, 0.2 .. 1$

$v := \sin(k)$

you can see that k is a range variable and not a vector. You can also see that although k appears on the right of the second expression, it doesn't appear on the left, so Mathcad returns an error.

Naturally, for every rule there is an exception, and there are two cases where you can use a range variable on just the right of a definition - taking a range variable sum or product.

$sumN := \sum_n n$ $sumN = 27$ $prodN := \prod_n n$ $prodN = 5040$

Conclusion

In summary, a vector and a range are quite distinct entities. A vector stores values while a range specifies values but doesn't store them.

A range variable is a variable that holds a range. A range variable is an instruction to expand the range into a complete set of values and then evaluate each of those values in turn.

Understanding the differences between a range variable and other variables, particularly those holding vectors, is essential to knowing how to use them and when to use them.

The next issue of PTC Express will have a follow-up article on this subject by Stuart Bruff.

About the Author

Stuart Bruff has 25 years experience in engineering, both as a Royal Air Force communications engineer officer and subsequently as a systems engineer in the United Kingdom aerospace industry. He has used Mathcad since version 7 and is a regular contributor to the **Mathcad User's Forums**. He also works as a Mathcad consultant.

Ranges, Sequences, and Vectors

Iteration ... again!

Stuart Bruff

Last month's article, **Range, Sequences, and Variables**, discussed the differences between vectors and range variables, and touched upon sequences. This article expands on the uses of range variables and highlights some of the limitations of a range variable only approach.. simple functions can help make using range variable easier. The article introduces programming and the use of sequences by way of the **for-loop**.

Vectors, Ranges, and Variables

Both ranges and vectors are types of data. A vector is a single-column array that stores list of other data types, such as numbers or even other arrays. A range does not store data but specifies the first, second, and last values of a uniformly spaced set of real numbers.

Vector

$v := \text{stack}(1,2,3,4)$ $v = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$

Range Variable

$i := 1,2..4$ $i =$

1
2
3
4

The examples above show first creating a vector using the function stack to combine the values 1 through 4, then assigning the vector to the variable v. You can then create a range using the first..last notation and assign it to the variable i.

A variable that has a range for its value is called a range variable; it differs from a normal variable in that Mathcad treats it as an instruction to evaluate an expression for each value of the range.

Example of using a range variable

The Fibonacci sequence is "1,1,2,3,5,8,13,21 ...", where each number in the sequence is the sum of the preceding two elements. To start the sequence, define the first two elements to both be 1. For the nth number, the indices of the two preceding number are simply n-1 and n-2.

$fib_0 := 1$ first two numbers in the sequence (the 'seed' values)

$fib_1 := 1$

$n := 2..7$ range variable to create the next 6 numbers

$fib_n := fib_{n-1} + fib_{n-2}$ make the nth number equal to the sum of the preceding two numbers

$fib^T = (1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21)$ the resulting vector (transposed to make it easier to read)

You can use a range variable to quickly and conveniently generate arrays of data where each value is dependent on only the 'current' range variable value or elements of the array that have already been calculated. The notation occurs commonly in mathematics and requires no special programming skills. With a small amount of thought, the range variable can meet many iteration requirements.

Parallel calculations using range variables

One potential problem with range variables is that the values of several iterated variables may be mutually dependent. If you calculate them individually, you lose that dependence, as the range variable applies to only one region. However, if you put related expressions inside an array, you can calculate the variables in parallel.

For example, consider an infection model with four variables. The time development of this model is given by the equations below:

i number of individuals

s number susceptible

d number decreased (eliminated)

r number recovered (immune)

$t := 0..4$ time

initialize **iterate**

$$\begin{pmatrix} i_0 \\ s_0 \\ d_0 \\ r_0 \end{pmatrix} := \begin{pmatrix} 50 \\ 22000 \\ 0 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} i_{t+1} \\ s_{t+1} \\ d_{t+1} \\ r_{t+1} \end{pmatrix} := \begin{pmatrix} 0.0001 \cdot s_t \cdot i_t \\ s_t - 0.0001 \cdot s_t \cdot i_t \\ d_t + 0.55 \cdot i_t \\ r_t + 0.45 \cdot i_t \end{pmatrix}$$

$t =$	$i_t =$	$s_t =$	$d_t =$	$r_t =$
0	50	22000	0	0
1	110	21890	27.5	22.5
2	240.79	21649.21	88	72
3	521.291	21127.919	220.435	180.356
4	1101.38	20026.539	507.145	414.937

If you simply calculate i_t in isolation, you would hit problems immediately. The value of i at any time depends upon the value of s, so you need to know the values of s. However, the values of s depend upon the values of i, so you need to know the values of i.

The array method of synchronizing iteration provides an elegant way round this dilemma and, once more, looks like commonly occurring notation.

Breaking the Iteration

You can numerically estimate square roots using seeded iteration. It is a simple example but shows how you can repeatedly evaluate expressions to determine convergence. A vector is initialized with a guess value, then the range variable generates successive guess values from the convergence expression. The resulting vector lists all of the guesses and, hopefully, the desired result.

In this case, start with a positive real number X for which you want the square root and an initial guess value for the square root.

$$X := 1024 \qquad guess_0 := \frac{X}{10}$$

Next create a range variable; as this algorithm converges fairly quickly, choose a small range:

$$N := 10 \qquad i := 0 .. N$$

Then iterate over the range, updating each guess in terms of previous ones.

$$guess_{i+1} := \frac{1}{2} \cdot \left(guess_i + \frac{X}{guess_i} \right)$$

Convergence is very fast, but the number of iterations, N, can be increased to suit the needs of the problem.

$i =$	$guess_i =$	$(guess_i)^2 - X =$
0	102.4	$9.462 \cdot 10^3$
1	56.2	$2.134 \cdot 10^3$
2	37.21	360.608
3	32.365	23.479
4	32.002	0.132
5	32	$4.226 \cdot 10^{-6}$
6	32	0
7	32	0
8	32	0
9	32	0
10	32	0

However, if the convergence rate is unknown or unpredictable, you could end up choosing a range that is significantly large in terms of both execution time and storage. It would be convenient to be able to stop the iteration when the guess values converged to a constant value. The **until** function is helpful here.

The **until** function takes the form until(expr1, expr2), where expr1 is a test expression (usually involving a single range variable). When this expression becomes negative, the until function halts iteration. expr2 is the value returned by the until function at each iteration.

Ideally, you want to halt the square root iteration when two successive guesses are equal. However, to avoid problems with numerical round off causing values to alternate by small amounts, you can terminate the process when the square of the guess differs from X by one part in a million. Define a function, close_enough, and call it for expr1. The previous guess expression is expr2.

$$\text{close\_enough}(x, r) := \left|x - r^2\right| - 10^{-6}$$

initialize the new guess vector

$$\text{newguess}_0 := \text{guess}_0$$

iterate until convergence is reached

$$\text{newguess}_{i+1} := \text{until}\left[\text{close\_enough}\left(X, \text{newguess}_i\right), \frac{1}{2}\cdot\left(\text{newguess}_i + \frac{X}{\text{newguess}_i}\right)\right]$$

count the rows and define a new range variable based on the length of newguess

$$\text{rows}(\text{newguess}) = 7$$

$$k := 0 .. \text{last}(\text{newguess})$$

show the new values

$k =$	$\text{newguess}_k =$
0	102.4
1	56.2
2	37.21
3	32.365
4	32.002
5	32
6	32

The **until** function has successfully reduced the number of iterations.

Some Issues with Range Variables

As you see, range variables provide Mathcad with a flexible and easy-to-use method of iterating expressions. A single range variable can apply to multiple expressions, which makes it easy to vary the number of iterations according to the particular details of a problem.

Sometimes varying the iterations can be a problem. Consider the square root solution. Every time you need to calculate a square root, you must either change X, copy and paste the guess expressions as needed, or modify the expressions. It would be more convenient if you could wrap the root algorithm up in a function and simply pass numbers to it.

Furthermore, although you can terminate expressions by use of the until function, you still have to create a large enough range to guarantee convergence - just add 6 zeros onto the end of X above to see the problem.

The examples above are deliberately simple, but many real world problems involve long and detailed expressions. The single line limitation of a single line expression makes such expressions difficult to write, interpret, and modify.

In addition, there are classes of problem where only the final value is of interest, and where each value only depends upon its predecessor, for example, the square root algorithm; a brief examination shows that $guess_{i+1}$ only depends on $guess_i$. Consequently, there is no need to store the intermediate results in a vector at all. Unfortunately, you can't use the until function here, as it only works in the context of a range variable and otherwise generates an error:

nguess := 100

$$nguess := until\left[close\_enough(X, nguess), \frac{1}{2}\cdot\left(nguess + \frac{X}{nguess}\right)\right]$$

This 'break' must occur within a loop

for loop and while loop

Mathcad's programming language can be used to resolve some of these issues, by grouping related statements together. The for-loop and while-loops offer more control than the range variable.

Note: The programming operators **while, for, if,** and **return** should only be entered using the programming toolbar or a keyboard shortcut. They won't work if you type in the word.

The while loop

The while operator starts with a condition, evaluates it, and if it's true (not equal to zero), executes the subsequent, indented expressions. The operation repeats until the expression becomes false (zero). Look at the square root algorithm solved by a **while loop**. The major difference between the **while loop** and the **until** function, is that the **while loop** iterates while the guess isn't close enough. Modify the convergence test to account for this. Guess is now a a simple scalar.

define convergence function $\text{far\_away}(x, r) := \left| x - r^2 \right| > 10^{-6}$

initialize the guess
while guess isn't good enough
update it

$$\begin{aligned} &\text{newguess} \leftarrow X \div 4 \\ &\text{while far\_away}(X, \text{newguess}) \\ &\quad \text{newguess} \leftarrow \frac{1}{2} \cdot \left(\text{newguess} + \frac{X}{\text{newguess}} \right) \end{aligned} \qquad = 32$$

Note the use of the left arrow operator to perform assignment. Although a Mathcad program can refer to an externally defined variable, it cannot modify it. If a program tries to assign a value to a variable, Mathcad creates a new 'local' variable (if one of that name doesn't already exist). This is an important point - the variable newguess in the program above is completely independent of the previous newguess, as you can see by evaluating newguess.

$$\text{newguess}^{T} = (102.4 \quad 56.2 \quad 37.21 \quad 32.365 \quad 32.002 \quad 32 \quad 32)$$

The local variable newguess in the program only exists within the context of the program, that is, it is 'local' to the program.

You don't have to worry about how many iterations the program needs to take, and it uses far less memory than multiple vectors.

An advantage of the while loop is that you don't have to worry about how many iterations the program needs to take, and it uses far less memory than multiple vectors.

The program also encapsulates the whole algorithm in one region, making it easier to copy and paste for different values. Even more usefully, you can convert the program to a function and call that as you would any other function.

$$\begin{aligned} \text{sqroot}(X) := \; &\text{newguess} \leftarrow X \div 4 \\ &\text{while far\_away}(X, \text{newguess}) \\ &\quad \text{newguess} \leftarrow \frac{1}{2} \cdot \left(\text{newguess} + \frac{X}{\text{newguess}} \right) \end{aligned}$$

$\text{sqroot}(81) = 9$

$\text{sqroot}(4096) = 64$

Now take a look at the **for loop**, but first review what a sequence is.

Sequences

A sequence is a comma-separated list of values, where a value is a scalar, string, array, sequence, range, function name, or an expression that evaluates to one of these types of values (for example an evaluated function or a range variable). The argument $1,2,3,4$ to the stack function is a sequence.

Note that a sequence is not a vector, and it isn't possible to pick out a single element from within it. A sequence also differs from a range in that it can't be assigned to a variable, which makes statements like the one below, flag up an error.

$s := 0, 0.1, 0.2$

$s :=$ ▮

Sequences are often used as arguments to functions. However, sequences are at their most useful as the argument to a **for loop**.

For Loops

A **for loop** takes the form

for $x \in s$

  z

where x is a variable, known as the iteration variable, s is a sequence and z is one or more lines that Mathcad evaluates for each iteration. The key to the **for loop** is that x works its way from left to right through the sequence, taking on each value in turn.

The first example uses a simple range that makes x take on the values 1 through 5; the local variable i indexes the vector v and assigns each value of k to successive elements of v.

$$\begin{array}{l} i \leftarrow 0 \\ \text{for } k \in 1..5 \\ \quad v_i \leftarrow k \\ \quad i \leftarrow i + 1 \\ v \end{array} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

The second example is slightly more complex; the sequence comprises the two scalars 1 and 2, and the range 3,5..9 consequently takes on the values 1, 2, 3, 5, 7, 9.

```
i ← 0
for k ∈ 1, 2, (3, 5 .. 9)
    v_i ← k
    i ← i + 1
v
```

$$= \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 7 \\ 9 \end{pmatrix}$$

The third example uses a matrix as the sequence. Mathcad scans the matrix element by element, working down each successive column. Note the different approach to indexing the vector v. With a zero-based array, the function rows always return a value one greater than the index of the last element of a vector. If you use rows as an index, it effectively points to the next 'free' element of the vector. To get it started, define v as a scalar. A scalar has zero rows, so writing to v_0 causes the first iteration to transform v into a vector and subsequent iterations extend v.

```
v ← 0
for k ∈ [1 2; 3 4]
  v_rows(v) ← k
v
```

$$= \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}$$

Referring back to the Fibonacci sequence, you now have a means of encapsulating the entire algorithm in a single region, using the multiline capability of programming to initialize the sequence and the for-loop to perform the iteration:

```
Fib(n) :=
    fib_0 ← 1
    return fib if n ≤ 1
    fib_1 ← 1
    return fib if n = 2
    for i ∈ 2 .. n
      fib_i ← fib_(i−1) + fib_(i−2)
    fib
```

$$\text{Fib}(7)^{T} = (1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21)$$

The function Fib(n), as defined above, returns a vector containing the first n + 1 Fibonacci numbers. The function also shows some other important aspects of programming. Unlike the range variable version, you can check that the function returns a valid result for n less than or equal to 1 and for n equal 2. You can use the **if** operator to check the condition and the **return** operator to stop evaluating the result.

The **if** operator tests a condition. If the condition is true (not zero), it evaluates an associated clause. The **if** operator may look a little strange since the condition appears on the right-hand side of the operator 'if', but the clause appears on the left-hand side. When the clause is spread over two or more lines, Mathcad drops the clauses underneath the **if** operator and indents them.

```
if x = 0
    y ← 2
    z ← 3
```

A final point is that if fib were not the last line of the function Fib, Mathcad returns fib_n; by default, Mathcad returns the last value calculated, which in this instance is the final value of the sequence. Adding fib ensures that you return the entire vector.

About the Author

Stuart Bruff has 25 years experience in engineering, both as a Royal Air Force communications engineer officer and subsequently as a systems engineer in the United Kingdom aerospace industry. He has used Mathcad since version 7 and is a regular contributor to the **Mathcad User's Forums**. He also works as a Mathcad consultant.