Loops in SAS

Looping is essential to statistical programming. Whether you need to iterate over parameters in an algorithm or indices in an array, a loop is often one of the first programming constructs that a beginning programmer learns. Today is the first anniversary of this blog, which is named The DO Loop

### Loops in SAS

Loops are fundamental to programming because they enable you to repeat a computation for various values of parameters. Different languages use different keywords to define the iteration statement. The most well-known statement is the "for loop," which is used by C/C++, MATLAB, R, and other languages. Older languages, such as FORTRAN and SAS, call the iteration statement a "do loop," but it is exactly the same concept.

### DO loops in the DATA step

The basic iterative DO statement in SAS has the syntax DO value = start TO stop. An END statement marks the end of the loop, as shown in the following example:

|  |
| --- |
| **data** A;  input v1 v2;  datalines;  1 2  3 4  ;  **run**;  data A;  do i = **1** to **5 by 2**;  y = i*\*\*2;* */\* 1\*1 3\*3 5\*5\*/*  output;  end;  run;  data A;  do i = **5** to **1 by -1**;  y = i*\*\*2;*  output;  end;  run;  data A;  do i = **1** to **5 by .2**;  y = i*\*\*2;*  output;  end;  run; |

By default, each iteration of a DO statement increments the value of the counter by 1, but you can use the BY option to increment the counter by other amounts, including non-integer amounts. For example, each iteration of the following DATA step increments the value i by 0.5:

|  |
| --- |
| **data** A;  do i = **1** to **5** by **0.5**;  y = i*\*\*2;* */\* values are 1, 2.25, 4, ..., 16, 20.25, 25 \*/*  output;  end;  **run**;  **data** A;  do i = **1** to **5** by **0.5**;  y = i*\*\*2;* */\* values are 1, 2.25, 4, ..., 16, 20.25, 25 \*/*  output;  end;  **run**; |

You can also iterate "backwards" by using a negative value for the BY option: do i=5 to 1 by -0.5.

### DO loops in SAS/IML Software

A basic iterative DO statement in the SAS/IML language has exactly the same syntax as in the DATA step, as shown in the following PROC IML statements:

|  |
| --- |
| D = 1:10 {1,2,3,…10}  proc iml;  x = 1:4; /\* vector of values {1 2 3 4} \*/  do i = 1 to 5;  z = sum(x##i); /\* 10, 30, 100, 354, 1300 \*/  end; |

In the body of the loop, z is the sum of powers of the elements of x. During the ith iteration, the elements of x are raised to the ith power. As mentioned in the previous section, you can also use the BY option to increment the counter by non-unit values and by negative values.

### Variations on the DO loop: DO WHILE and DO UNTIL

On occasion, you might want to stop iterating if a certain condition occurs. There are two ways to do this: you can use the WHILE clause to iterate as long as a certain condition holds, or you can use the UNTIL clause to iterate until a certain condition holds. You can use the DO statement with a WHILE clause to iterate while a condition is true. The condition is checked before each iteration, which implies that you should intialize the stopping condition prior to the loop. The following statements extend the DATA step example and iterate as long as the value of y is less than 20. When i=4, the WHILE condition is not satisfied, so the loop iterates again.

|  |
| --- |
| **data** A;  y = **0**;  do i = **1** to **5** by **0.5** while(y < **20**);  y = i*\*\*2;* */\* values are 1, 2.25, 4, ..., 16, 20.5 \*/*  output;  end;  **run**; |

You can use the iterative DO statement with an UNTIL clause to iterate until a condition becomes true. The UNTIL condition is evaluated at the end of the loop, so you do not have to initialize the condition prior to the loop. The following statements extend the PROC IML example. The iteration stops after the value of z exceeds 200.

|  |
| --- |
| proc iml;  x = 1:4;  do i = 1 to 5 until(z > 200);  z = sum(x##i); /\* 10, 30, 100, 354 \*/  end; |

In these examples, the iteration stopped because the WHILE or UNTIL condition was satisfied. If the condition is not satisfied when i=5 (the last value for the counter), the loop stops anyway. Consequently, the examples have two stopping conditions: a maximum number of iterations and the WHILE or UNTIL criterion. SAS also supports a DO WHILE and DO UNTIL syntax that does not involve using a counter variable. It is worth noting that a DO loop with an UNTIL clause always executes at least one time because the condition is evaluated at the end of the loop. To prevent this behavior, use a DO loop with a WHILE clause.

### Looping over a set of items (foreach)

Some languages support a "foreach loop" that iterates over objects in a collection. SAS doesn't support that syntax directly, but there is a variant of the DO loop in which you can iterate over values in a specified list. The syntax in the DATA step is to specify a list of values (numeric or character) after the equal sign. The following example iterates over a few terms in the Fibonacci sequence:

|  |
| --- |
| **data** A;  do v = **1**, **1**, **2**, **3**, **5**, **8**, **13**, **21**;  y = v/lag(v);  output;  end;  **run**; |

The ratio of adjacent values in a Fibonacci sequence converges to the golden ratio, which is 1.61803399.... The SAS/IML language does not support this syntax, but does enable you to iterate over values that are contained in a vector (or matrix). The following statements create a vector, v, that contains the Fibonacci numbers. An ordinary DO loop is used to iterate over the elements of the vector. At the end of the loop, the vector z contains the same values as the variable Y that was computed in the DATA step.

|  |
| --- |
| proc iml;  v = {1, 1, 2, 3, 5, 8, 13, 21};  z = j(nrow(v),1,.); /\* initialize ratio to missing values \*/  do i = 2 to nrow(v);  z[i] = v[i]/v[i-1];  end; |

### Avoid unnecessary loops in the SAS/IML Language

I have some advice on using DO loops in SAS/IML language: look carefully to determine if you really need a loop. The SAS/IML language is a matrix/vector language, so statements that operate on a few long vectors run much faster than equivalent statements that involve many scalar quantities. Experienced SAS/IML programmers rarely operate on each element of a vector. Rather, they manipulate the vector as a single quantity. For example, the previous SAS/IML loop can be eliminated:

|  |
| --- |
| proc iml;  v = {1, 1, 2, 3, 5, 8, 13, 21};  idx = 2:nrow(v);  z = v[idx]/v[idx-1]; |