

Learning to love the SAS LAG function

H. Ament, MSD, Oss, The Netherlands

ABSTRACT

The LAG and DIF functions (introduced in the SAS 1979 edition) are often used to get access to the previous value of a variable in a DATA step. For each defined LAG and DIF function in a data step a separate queue is maintained. Because programmers often don't realize this, sometimes the result of these functions are unexpected. When the LAG and DIF functions are well understood, they are powerful tools in SAS ® programming.

INTRODUCTION

In order to retrieve the value of a previous observation the function LAG or LAG1 is often used. The previous value is often compared to the most recent value.

In code:

```
DATA newset;
  SET oldest;
  IF VarValue = LAG(VarValue) THEN ...
RUN;
```

Below you will find other ways to retrieve the previous value of a variable.

1. By storing the value - at the end of the data step - in a variable that is retained:

```
DATA newset;
  SET oldest;
  IF PrevVarValue = VarValue THEN ...
  PrevVarValue = VarValue;
  RETAIN PrevVarValue;
RUN;
```

2. By storing the value in a new variable that is created before the first SET statement in the data step:

```
DATA newset;
  PrevVarValue = VarValue;
  SET oldest;
  IF PrevVarValue = VarValue THEN ...
RUN;
```

The three code examples above look all the same, however their behavior in a data step is different and as a result of this programmers experience their own specific problems. The rest of this paper tells what SAS ® is doing when the LAG and DIF functions are invoked.

EXAMPLE OF AN UNEXPECTED RESULT

Here is an example of a program giving 'unexpected' results.

```
DATA newset;
  SET oldset;
  BY assessment;
```

PhUSE 2011

```
IF NOT first.assessment THEN DO;
  IF subjid = LAG(subjid) THEN DO;
    ... code to be executed... e.g. increase a counting number for this subject
    count+1;
  ELSE DO;
    ... code to be executed... e.g. set a counting number for this subject to 1
    count = 1;
  END;
END;
RUN;
```

Because LAG(SUBJID) is executed conditionally, LAG(subjid) does not always contain the value of SUBJID of the previous observation. In the example above variable COUNT will not always be set to 1 if SUBJID differs from the previous observation. At the end of this paper you can understand why LAG(SUBJID) returns not the previous value for each second observation of each assessment.

When the previous value of SUBJID is stored in a new variable (LAGsubjid) at the beginning of the data step the problem is solved:

```
DATA newset;
  SET oldset;
  BY assessment;
  LAGsubjid = LAG(subjid);
  IF NOT first.assessment THEN DO;
    IF subjid = LAGsubjid THEN DO;
      ... code to be executed...
    END;
  END;
RUN;
```

LAG FUNCTION

From the SAS®¹ Help the following definition of LAG can be found:

The LAG functions, LAG1, LAG2, ..., LAGn return values from a **queue**. LAG1 can also be written as LAG. A LAGn function stores a value in a queue and returns a value stored previously in that queue. Each occurrence of a LAGn function in a program generates its own queue of values.

It is important to understand that for each LAG function a separate queue with a specific length is created, whereby the argument of the LAG function is entered into the queue. All values in the queue are moved one position forward and the oldest value entered will be returned into the expression.

Hence, for the first n executions of LAGn, missing values are returned, thereafter the lagged values of the argument begin to appear. Note that in case of an array name as the argument of LAGn function, a separate queue is maintained for **each** variable in the array.

To make more clear what is happening, an example with LAG3 - a queue of 3 values – is shown, whereby the queue has already been filled in previous data steps, with the values 3, 2 and 1. The variable A contains the value 5.

When LAG3 is executed the argument of LAG3 – in this case A - goes into the queue and last value in the queue – in this case 1 - will be returned.

```
* A contains 5;
```

```
Y = LAG3(A);
```



The contents of A which is 5 goes into the queue, values 3 and 2 move one position and the value 1 is returned.

PhUSE 2011

The result is:

Y contains 1

Lag Queue is

Last: 5

2nd Last: 3

3th Last: 2

Note that the numbers in the queue cannot be accessed in a data step.

DIF FUNCTION

From the SAS® Help¹:

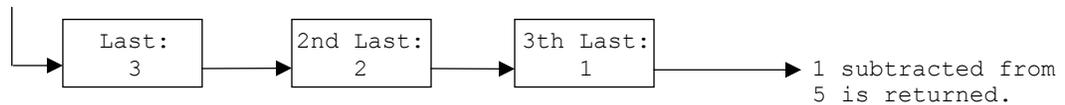
The DIF functions, DIF1, DIF2, ..., DIF100, return the first differences between the argument and its n^{th} lag. DIF1 can also be written as DIF. DIFn is defined as $\text{DIFn}(x)=x-\text{LAGn}(x)$.

The DIF function is almost the same as the LAG function. The difference is that returned value from LAGn is subtracted from the argument of the DIF function.

When we use the same example as for the LAG function the following will happen

```
* A contains 5;
```

```
Y = DIF3(A);
```



A - which is 5 - goes into the queue, value 3 and 2 move one position and the value 1 is subtracted from 5 and is returned.

The result is:

Y is 4

Last: 5

Second Last: 3

Third Last: 2

EXAMPLES

Below, four examples are shown of the use of the LAG function.

EXAMPLE 1, TWO QUEUES

Here are two separate identical queues. The content of both LAG queues are the same after each iteration of the data step.

The result is that X and Y contain the value of A from the previous observation.

```
DATA a;  
INPUT a;  
x = LAG(a);  
y = LAG(a);  
CARDS;  
1  
2  
;  
  
PROC PRINT;  
VAR a x y;
```

PhUSE 2011

RUN;

Output:

Obs	a	x	y
1	1		
2	2	1	1

EXAMPLE 2, ONE QUEUE EXECUTED IN A DO LOOP

Here is one queue. The value entered in the queue in the first iteration of J is returned in the second iteration. The result is that X has the same value as A and that the content of the queue is a missing value.

```
DATA b;
INPUT a;
DO J=1 to 2;
  x = LAG(a);
END;
CARDS;
1
2
;

PROC PRINT;
VAR a x;
RUN;
```

Output:

Obs	a	x
1	1	1
2	2	2

EXAMPLE 3, ONE QUEUE USED IN A LINK STATEMENT

Here is one queue, stored in a subroutine and called via LINK statement.

The value entered after the first call is returned in the second call. The result is the same as for EXAMPLE 2: X has the same value as A.

```
DATA c;
  INPUT a;
  LINK example;
  LINK example;
  RETURN;

  Example: x = LAG(a);
  RETURN;
CARDS;
1
2
;

PROC PRINT;
VAR a x;
RUN;
```

Output:

Obs	a	x
1	1	1
2	2	2

EXAMPLE 4, THREE QUEUES, ONE FOR EACH DISTINCT VALUE OF TREATMENT

Here the task is to store in the variable LAGSUBJ the previous value of SUBJID with the same treatment. The data is already sorted by SUBJID. There are three treatments A, B and C.

PhUSE 2011

Instead of using a PROC SORT, a LAG function for each treatment is sufficient to get the desired result. Therefore three queues are created, one for each treatment: A, B and C.

```
DATA a;
  INPUT subjid $ treatment $;
  SELECT (treatment);
    WHEN ('A') lagsubj = LAG (subjid);
    WHEN ('B') lagsubj = LAG (subjid);
    WHEN ('C') lagsubj = LAG (subjid);
END;
CARDS;
1 A
2 A
3 B
4 A
5 C
6 C
7 B
8 C
9 C
10 A
;
RUN;

PROC PRINT;
RUN;
```

Output (the dotted lines show where each value of SUBJID is returned from the LAG queue):

Obs	treatment	subjid	lagsubj
1	A	1	
2	A	2	1
3	B	3	
4	A	4	2
5	C	5	
6	C	6	5
7	B	7	3
8	C	8	6
9	C	9	8
10	A	10	4

EXAMPLE 5, ROW OF FIBONACI

Most of us know the golden ratio mentioned in the novel "The Da Vinci Code" by Dan Brown. The golden ratio is exactly $(1+\sqrt{5})/2$, which is approximately 1.618033988. The row of Fibonacci converges rapidly to this number.

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two. The quotient of the last 2 numbers converges to the golden ratio.²

This first 6 Fibonacci numbers are:

n	Fibonacci Number	Quotient
1	0	
2	1	n.a.
3	1	1 (= 1/1)
4	2	2 (= 2/1)
5	3	1,5 (= 3/2)
6	5	1,667 (= 5/3)

PhUSE 2011

Etc.

Here is an example of SAS code using the LAG function and omitting the first number zero. As you can see the golden ratio is calculated to 7 figures after 18 iterations.

```
DATA fib;
fib = 1;
OUTPUT;
DO UNTIL (fib > 2000);
  quotient = SUM(fib, LAG(fib)) /fib;
  fib      = SUM(fib, LAG(fib));
  OUTPUT;
END;
RUN;

PROC PRINT;
VAR fib quotient;
RUN;
```

Output:

<u>Obs</u>	<u>fib</u>	<u>quotient</u>
1	1	
2	1	1.0000000000
3	2	2.0000000000
4	3	1.5000000000
5	5	1.6666666667
6	8	1.6000000000
7	13	1.6250000000
8	21	1.6153846154
9	34	1.6190476190
10	55	1.6176470588
11	89	1.6181818182
12	144	1.6179775281
13	233	1.6180555556
14	377	1.6180257511
15	610	1.6180371353
16	987	1.6180327869
17	1597	1.6180344478
18	2584	1.6180338134

CONCLUSION

The conclusion is that the LAG and DIF function are powerful functions. If well understood they can be used in many ways.

It is advised is that if the previous value in a data step has to be retrieved and the code is simple, the LAG function can be used.

If the code is more complex, e.g. when previous values are used within a conditional section, the RETAIN statement is recommended.

REFERENCES

¹ On line SAS® HELP, <http://support.sas.com>

² On line Wikipedia encyclopedia: <http://en.wikipedia.org/wiki/Fibonacci>

CONTACT INFORMATION

Author name : H. Ament
Company : MSD
Address : PO BOX 20
5340 BH Oss
Email : h.ament@merck.com
Web : www.msd.com