Paper 1082-2017

Hash Objects: When LAGging Behind Just Doesn't Work

Andrea Wainwright-Zimmerman, Experis

ABSTRACT

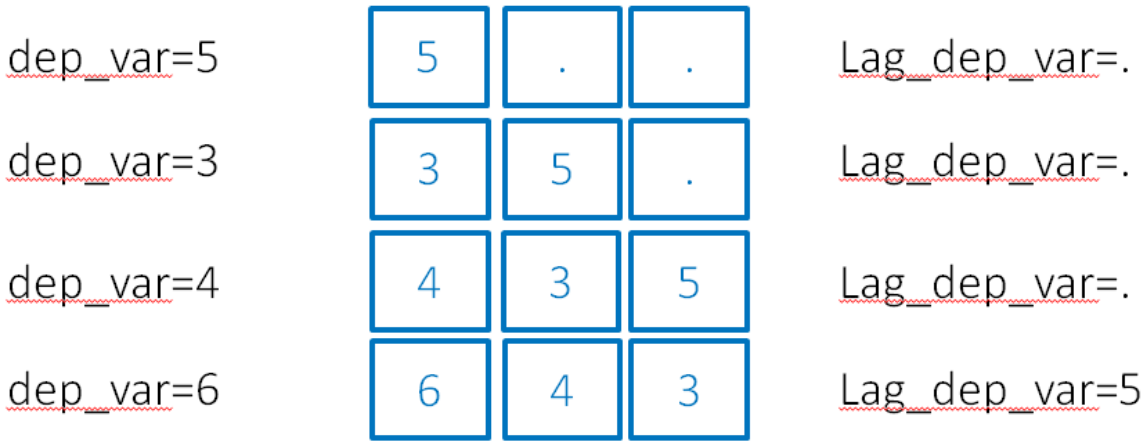
When modeling time series data, we often use a LAG of the dependent variable. The LAG function works great for this, until you try to predict out into the future and need the model's predicted value from one record as an independent value for a future record. This paper examines exactly how the LAG function works, and explains why it doesn't in this case. It also explains how to create a HASH object that will accomplish a LAG of any value, how to load the initial data, how to add predicted values to the HASH object, and how to extract those values when needed as an independent variable for future observations.

INTRODUCTION

I was asked to model a time series of data, but we wanted to take advantage of some of the newer modeling procs SAS® has to offer other than PROC TIMESERIES. The challenge I faced was trying to forecast out into the future. The LAG function wasn't working as I thought it would, so I decided to better understand the function.

THE LAG FUNCTION

The LAG function is called with LAG*n*(*variable_name*). When SAS sees this, it creates a queue in memory of length *n*. It is initially filled with missing values. Every time the function is called, SAS pulls the first data point out, and it pushes the existing value into the end of the queue. For example lag_dep_var=lag3(dep_var) would work like this:



Intuition tells us this should be easy. We want to take the predicted value from the previous observation, perform the calculation that produces our model score, and then move on to the next record.

The challenge is that the LAG function does the push and the pull simultaneously; there is no way to do any calculations between those two actions. Not even multiple calls of the same LAG function on the same variable work, each call results in a separate queue.

HASH

HASH object to the rescue!

First, we need to define the HASH object and load it with our starting data point(s):

```
data scored;
set  raw;
if _n_=1 then do;
   declare hash myLag(dataset: "raw(where=(QtySold=.))"
   myLag.definekey("date");
   myLag.definedata("lag_QtySold_1");
   myLag.definedone();
end;/*if _n_=1*/
```

We only want to define the object once. This is accomplished with IF _N_=1 THEN DO; and END;

We define the HASH object with the syntax DECLARE HASH followed by a name (in this example I used myLag). This name is used with different methods to interact with the HASH object.

We can load that HASH object with the DATASET: option by specifying the dataset inside double quotes. We can even use standard data step options to limit the rows, such as the WHERE statement in the example above. This is very important since the HASH object resides in active memory. I highly recommend thinking about how few rows and columns you really need. In this case, I only need the rows where my dependent variable is missing and I want to calculate a predicted value.

The DEFINEKEY method tells SAS what field(s) are going to identify the rows in our HASH object. SAS will not alter the values stored as key.

The DEFINEDATA method tells SAS what field(s) are going to be passed between the data step and the HASH object. SAS can alter the values of these fields in the HASH object, unlike the key fields.

The DEFINEDONE method completes the declare block and tells SAS we are done initializing our HASH object.

After scoring the Oct 19th record our HASH object and dataset look like this:

Snippet of data set SCORED

qtySold	date	product	dow	lag_qtySold_1	pred
15	16-Oct-04	Barbeque potato chips	7	6	.
7	17-Oct-04	Barbeque potato chips	1	15	.
5	18-Oct-04	Barbeque potato chips	2	7	.
.	19-Oct-04	Barbeque potato chips	3	5	3.5955
.	20-Oct-04	Barbeque potato chips	4	.	.
.	21-Oct-04	Barbeque potato chips	5	.	.
.	22-Oct-04	Barbeque potato chips	6	.	.
.	23-Oct-04	Barbeque potato chips	7	.	.

HASH

KEY	DATA
19-Oct-04	5
20-Oct-04	.
21-Oct-04	.
22-Oct-04	.
23-Oct-04	.

The Oct 20th record is where we will start to need our HASH object. We can push the predicted value from the Oct 19th to our HASH object to make it available for the next record:

```
/*scoring*/
if qtysold=. then do;
   pred=-0.19751+0.67111*lag_qtySold_1+0.14582*dow;
/*fill in HASH object so the next obs can pull from it*/
   rc=myLag.replace(key: Date, data: pred);
end;/*if qtysold=.*/
```

The REPLACE method takes the value from the data set, in this case PRED, and loads it to the HASH object for the record indicated by the key, in this case the same row with the same date.

Now our data set and HASH object look like this:

Snippet of data set SCORED					
qtySold	date	product	dow	lag_qtySold_1	pred
15	16-Oct-04	Barbeque potato chips	7	6	.
7	17-Oct-04	Barbeque potato chips	1	15	.
5	18-Oct-04	Barbeque potato chips	2	7	.
.	19-Oct-04	Barbeque potato chips	3	5	3.5955
.	20-Oct-04	Barbeque potato chips	4	.	.
.	21-Oct-04	Barbeque potato chips	5	.	.
.	22-Oct-04	Barbeque potato chips	6	.	.
.	23-Oct-04	Barbeque potato chips	7	.	.

HASH	
KEY	DATA
19-Oct-04	3.5955
20-Oct-04	.
21-Oct-04	.
22-Oct-04	.
23-Oct-04	.

When we move to the next row, that's when things get interesting. We need to reach back to the Oct 19th row in our HASH object and pull that value into our dataset:

```
if lag_QtySold_1=. then do;/*only pull from HASH if not already filled in*/
   rc=myLag.find(key: Date-1);
end;/*if lag=.*/
```

The FIND method looks for the row in the HASH object specified by the KEY (in this case the data rows with a date one less than the current row in our dataset) and then it will pull the data from the HASH object and store it in the dataset, replacing the same variable. When we defined myLag, we loaded the variable lag_qtySold_1, so the find method will replace that variable in our dataset, even though we loaded it with the value of pred from the previous observation. Our data set and HASH object now look like this:

Snippet of data set SCORED					
qtySold	date	product	dow	lag_qtySold_1	pred
15	16-Oct-04	Barbeque potato chips	7	6	.
7	17-Oct-04	Barbeque potato chips	1	15	.
5	18-Oct-04	Barbeque potato chips	2	7	.
.	19-Oct-04	Barbeque potato chips	3	5	3.5955
.	20-Oct-04	Barbeque potato chips	4	3.5955	
.	21-Oct-04	Barbeque potato chips	5	.	.
.	22-Oct-04	Barbeque potato chips	6	.	.
.	23-Oct-04	Barbeque potato chips	7	.	.

HASH	
KEY	DATA
19-Oct-04	3.5955
20-Oct-04	.
21-Oct-04	.
22-Oct-04	.
23-Oct-04	.

Now we continue to let SAS score the rest of the rows of data, pulling lag_qtySold_1 from the previous date's row in the HASH object, creating pred as a function of our lag and independent variables, and then loading that date's value of pred to the HASH object. Once all of this is done our dataset and HASH object look like this:

Snippet of data set SCORED					
qtySold	date	product	dow	lag_qtySold_1	pred
15	16-Oct-04	Barbeque potato chips	7	6	.
7	17-Oct-04	Barbeque potato chips	1	15	.
5	18-Oct-04	Barbeque potato chips	2	7	.
.	19-Oct-04	Barbeque potato chips	3	5	3.5955
.	20-Oct-04	Barbeque potato chips	4	3.5955	2.798746005
.	21-Oct-04	Barbeque potato chips	5	2.798746005	2.409856431
.	22-Oct-04	Barbeque potato chips	6	2.409856431	2.29468875
.	23-Oct-04	Barbeque potato chips	7	2.29468875	2.363218567

HASH	
KEY	DATA
19-Oct-04	3.5955
20-Oct-04	2.79874601
21-Oct-04	2.40985643
22-Oct-04	2.29468875
23-Oct-04	2.36321857

LAGS OTHER THAN 1

In this example we were just using a LAG1, but this method would work for any value of LAG*n*, just change the math in the FIND method to look back *n* records (key: Date-*n*).

TIME INTERVALS OTHER THAN DAYS

In this example we had daily data, but often that is not the case. It could be weekly, monthly, quarterly, yearly, or even smaller increments such as hourly. My recommendation is to create a field in your raw data that will be an integer such that each record is 1 apart. PROC SQL using SELECT INTO: can be used to take the minimum date and store it in a macro variable, for example min_date, then the INTCK function can be used to count the intervals between each value and that first date:

```
proc sql noprint;
select min(date) into: min_date
from sashelp.stocks;
quit;

data temp;
set sashelp.stocks;
num_mths=intck('MONTH',&min_date,date);
run;
```

COMPLETE CODE

This is the complete code used in this paper. It uses the SASHELP.SNACKS dataset available to all SAS users and can be copied, pasted, and run as is:

```
data pred_space;
format date date9.;
do date='19OCT2004'd to '31DEC2004'd;
   product="Barbeque potato chips";
   output;
end;
run;

data raw;
set sashelp.snacks(where=(product="Barbeque potato chips")keep=QtySold date
product) pred_space;
by date;
dow=weekday(date);
lag_qtySold_1=lag1(QtySold);
run;

data scored;
set  raw;

if _n_=1 then do;
   declare hash myLag(dataset: "raw(where=(QtySold=.))");
   myLag.definekey("date");
   myLag.definedata("lag_qtySold_1");
   myLag.definedone();
end;/*if _n_=1*/

if lag_qtySold_1=. then do;/*only pull from HASH if not filled in*/
   rc=myLag.find(key: Date-1);
end;/*if lag=.*/
```

```
/*scoring*/
if qtysold=. then do;
   pred=-0.19751+0.67111*lag_qtySold_1+0.14582*dow;
/*fill in HASH object so the next obs can pull from it*/
   rc=myLag.replace(key: Date, data: pred);
end;/*if qtysold=.*/

drop rc ;
run;
```

CONCLUSION

The LAG function is very useful in time series analysis, but it cannot help us when we try to use our model to predict out into the future if we are using the LAG of our dependent variable. The HASH object can be used for this purpose since we can load data to it, pull data from it, use that data in calculations, and then load that data into the HASH object for later use.

ACKNOWLEDGMENTS

Thank you to Experis for their career support and thank you to my supportive Verizon coworkers, especially Nish Herat who was the first to mention the HASH object as a possible solution.

RECOMMENDED READING

- *SAS® Hash Object Programming Made Easy – Michele M. Burlew*
- *SAS® Functions by Example – Ron Cody*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Andrea Wainwright-Zimmerman
Experis
AndreaNWZimmerman@gmail.com
www.linkedin.com/in/andreawainwrightzimmerman/