PharmaSUG China 2018 – Paper AD-01

Practical Uses of the DOW Loop in Pharmaceutical Programming

Richard Read Allen, Peak Statistical Services, Evergreen, CO, USA

ABSTRACT

The DOW-Loop was originally developed by Don Henderson and popularized the past few years on the SAS-L listserv by Paul Dorfman and Ian Whitlock, among others. It is a programming technique that involves taking control of the implicit DO loop inherent in the DATA step by identifying and storing a value in a variable that is retained until all records for a by-group have been processed. The DOW-Loop relies on the fact that the values of assigned variables, or variables created by assignment statements in the DATA step, are not reset to missing until SAS returns to the top of the DATA step. It has a variety of applications, including streamlining LOCF and change from baseline calculations as well as transposing multiple variables for a by-group. Some examples of these applications will be presented.

INTRODUCTION

The DOW loop is a powerful technique that moves the DATA step SET statement inside of an explicitly-coded DO-loop. This gives the programmer complete control over retention of variable values and the population of the Program Data Vector (PDV) by allowing a natural isolation of DO-loop instructions related to a certain break-event. The DOW isolates actions performed before and after the DO-loop from the instructions inside the loop and eliminates the necessity of retaining or reinitializing variables in most applications. In its most basic and well-known form using a DO UNTIL (LAST.ID) construct, it naturally lends itself to BY-processing of grouped data.

THE DOW LOOP

The basic structure of the DOW loop is as follows [4: Dorfman/Shajenko]:

```
data ... ;
     <stuff done before break-event> ;
     do <index specs> until ( break-event ) ;
          set … ;
          by …;
          <stuff done for each record> ;
     end ;
    <stuff done after break-event... > ;
run ;
```

There are three separate sections of code in this structure where instructions can be grouped for different types of processing, depending on how you need to handle the data.

1. Between the top of the implied data step loop and before the first record in the by-group is read if the action needs to be done before the by-group is processed.
2. Inside the DOW-loop, for each record in the by-group, if the action needs to be done to each record.
3. After the last record in the by-group has been processed and before the bottom of the implied data step loop, if the action such as summarizing needs to be done after the by-group is processed.

Here's a brief description of how the inner loop works:

- The DOW-Loop itself begins with the DO UNTIL statement and takes control from the traditional implicit DATA step loop.
- Because the SET statement is inside of the DOW-loop, the loop is not exited until after the last record for the break-event has been processed.
- Variables populated inside of the loop are retained while all of the records for the break-event are read in.
- A single record for the break-event, containing the filled arrays, is output at the completion of the DOW-loop by the "implied" output at the end of the data step.

The Dorfman/Shajenko paper [4] has some detailed explanations of the inner workings of the DOW loop and its

variations, including using multiple DOW loops in the same data step and using the DOW with the data step hash object. We will look briefly at using the double DOW in an example.

APPLICATIONS OF THE DOW

The compact data step structure of the DOW loop lends itself to a number of interesting programming applications. As an introduction to this type of coding we will look at some examples of the following types of applications of the DOW.

- Transposing multiple variables simultaneously
- Change from Baseline Calculations
- LOCF (Last Observation Carried Forward)
- Change from Baseline Calculations using a Double DOW
- LOCF and Change from Baseline Calculations using a Double DOW

EXAMPLE 1: TRANSPOSING

To transpose a number of variables in a by-group using PROC TRANSPOSE is not an easy task. One would have to perform multiple transposes (or split one transpose output) and a merge. This can all be done in one data step using the DOW-loop. The Brucken paper [1] has a very nice example of this. I use this technique often to transpose data for presentation purposes as follows:

Suppose we have the following summarized data:

Disease	analyte	Group	n	mean
DL	HDL	1	509	55.408
DL	HDL	2	303	53.191
DL	HDL	3	511	56.447
DL	LDL	1	506	101.631
DL	LDL	2	300	88.226
DL	LDL	3	506	96.541
DL	Total	1	515	185.097
DL	Total	2	306	170.975
DL	Total	3	513	182.161
DM	A1c	1	287	6.822
DM	A1c	2	302	7.003
DM	A1c	3	335	6.780

We'd like to have the final output look like the following:

Disease	analyte	n1	n2	n3	Mean1	Mean2	Mean3	Diff13	Diff23
DL	HDL	509	303	511	55.408	53.191	56.447	1.03836	3.2556
DL	LDL	506	300	506	101.631	88.226	96.541	-5.09048	8.3149
DL	Total	515	306	513	185.097	170.975	182.161	-2.93591	11.1860
DM	A1c	287	302	335	6.822	7.003	6.780	-0.04239	-0.2234

If we use proc transpose to do this as follows:

```
proc transpose data=Stats out=Raw_MeansDiffs_Trans;
    by Disease Analyte;
    var n mean;
run;
```

We get this output:

Disease	analyte	_NAME_	COL1	COL2	COL3
DL	HDL	n	509.000	303.000	511.000
DL	HDL	mean	55.408	53.191	56.447
DL	LDL	n	506.000	300.000	506.000

```
DL        LDL       mean     101.631     88.226     96.541
DL        Total     n        515.000    306.000    513.000
DL        Total     mean     185.097    170.975    182.161
DM        A1c       n        287.000    302.000    335.000
DM        A1c       mean       6.822      7.003      6.780
```

The transpose output needs to be split and merged back together by Disease and Analyte to obtain the desired results.

However, this can be done in one data step using the following DOW code:

```
data Raw_MeansDiffs(drop=n Group mean);
  array _en n1-n3;
  array _mn Mean1-Mean3;
  do until (last.Analyte);
    set Stats;
    by Disease Analyte;
    _en(Group)=n;
    _mn(Group)=mean;
  end;
  Diff13=Mean3-Mean1;
  Diff23=Mean3-Mean2;
  output;
run;
```

One can even get fancy and use a multi-dimensional array inside the DOW loop. The following is a simple sample of how this can be done with the above data.

```
data Raw_MeansDiffs_2dim(drop=n Group mean);
  array _grp(3,2) n1 Mean1
                  n2 Mean2
                  n3 Mean3;
  do until(last.Analyte);
    set Stats;
    by Disease Analyte;
    _grp(Group,1)=n;
    _grp(Group,2)=mean;
  end;
  Diff13=Mean3-Mean1;
  Diff23=Mean3-Mean2;
  output;
run;
```

EXAMPLE 2: CHANGE FROM BASELINE

Change from baseline calculations can be simplified considerably using the DOW. The second Brucken paper [2] has a great example of this and explains how stepping through the DOW loop changes the PDV from record to record.

Let's say we have the following blood pressure data. We'd like to calculate the baseline value as the last value on or before the treatment start date and the change from baseline for all other visits. Most commonly the baseline values would be created in a separate dataset for each subject and merged with the original dataset. The change from baseline calculation can then be done in the resultant data. This can all be done in one data step using DOW logic.

```
data BP;
  input subject visit @7 vsdt mmddyy10. @18 tmtstdt mmddyy10. sysbp diabp @;
  datalines;
1  1  09/15/2007 10/15/2007 140 90
1  3  10/15/2007 10/15/2007 138 94
1  4  10/29/2007 10/15/2007 142 86
1  5  10/15/2007 10/15/2007 130 80
1  6  11/13/2007 10/15/2007 132 84
1  7  11/29/2007 10/15/2007 130 83
1  8  12/15/2007 10/15/2007 131 81
1  9  12/30/2007 10/15/2007 128 78

2  1  09/01/2007 09/30/2007 122 70
2  2  09/15/2007 09/30/2007 125 73
2  3  09/30/2007 09/30/2007 128 80
2  4  10/14/2007 09/30/2007 130 78
2  5  10/30/2007 09/30/2007 128 82
2  6  11/13/2007 09/30/2007 126 74
2  7  11/28/2007 09/30/2007 128 76
2  9  12/29/2007 09/30/2007 124 80
 ;;;;
run;
```

The following is the DOW loop code to perform these changes from baseline calculations, where baseline is the last observation on or before treatment start date:

```
data Change;
   do until(last.subject);
      set BP;
      by subject visit;
      if vsdt<=tmtstdt then do;
         b_sysbp=sysbp;
         b_diabp=diabp;
      end;
      else do;
         c_sysbp=sysbp-b_sysbp;
         c_diabp=diabp-b_diabp;
      end;
      if vsdt<tmtstdt then do;
         b_sysbp=.;
         b_diabp=.;
      end;
      output;
   end;
run;
```

The last do loop blanks out the baseline values for any record before treatment start. If this is not desired, the do loop should be removed and all records will contain the baseline information.

RESULTS:

subject	visit	vsdt	tmtstdt	sysbp	diabp	b_sysbp	b_diabp	c_sysbp	c_diabp
1	1	09/15/2007	10/15/2007	140	90	.	.	.	.
	3	10/15/2007	10/15/2007	138	94	138	94	.	.
	4	10/29/2007	10/15/2007	142	86	138	94	4	-8
	5	11/13/2007	10/15/2007	130	80	138	94	-8	-14
	6	11/29/2007	10/15/2007	132	84	138	94	-6	-10
	7	12/15/2007	10/15/2007	130	83	138	94	-8	-11
	8	12/30/2007	10/15/2007	131	81	138	94	-7	-13
	9	01/13/2008	10/15/2007	128	78	138	94	-10	-16
2	1	09/01/2007	09/30/2007	122	70	.	.	.	.
	2	09/15/2007	09/30/2007	125	73	.	.	.	.
	3	09/30/2007	09/30/2007	128	80	128	80	.	.
	4	10/14/2007	09/30/2007	130	78	128	80	2	-2
	5	10/30/2007	09/30/2007	128	82	128	80	0	2
	6	11/13/2007	09/30/2007	126	74	128	80	-2	-6
	7	11/28/2007	09/30/2007	128	76	128	80	0	-4
	9	12/29/2007	09/30/2007	124	80	128	80	-4	0

EXAMPLE 3: LOCF

We can use the DOW-loop and the implicit retain that is one of its most useful properties to greatly simplify LOCF (Last Observation Carried Forward) calculations. The Chakravarthy article [3] is a great source for learning more about this method.

Let's say we have some data with missing values for some visits that we'd like to fill in with the most recent value for the missing values. Below is some sample data:

```
data lab;
   input PT:$3. visit labstd;
   cards;
101      1        0.1
101      2        0.2
101      3        0.3
101      5        0.5
101      6        0.6

102      2        0.1
102      3        0.2
102      5        0.3
102      7        0.5
102      8        0.6
;;;;
run;
```

These are the results we'd like to get with most recent value added as the LOCF values to each visit for each subject when that data is missing.

PT	visit	labstd	locf LabStd
101	1	0.1	0.1
101	2	0.2	0.2
101	3	0.3	0.3
101	4	.	0.3
101	5	0.5	0.5
101	6	0.6	0.6
101	7	.	0.6
101	8	.	0.6

```
102      1         .         .
102      2       0.1       0.1
102      3       0.2       0.2
102      4         .       0.2
102      5       0.3       0.3
102      6         .       0.3
102      7       0.5       0.5
102      8       0.6       0.6
```

We start by creating a shell dataset for the visits that we should have for each subject. There are multiple ways to do this, including the following:

```
Data VisitFrame;
  set lab(keep=pt);
  by pt;
  if first.pt then do visit=1 to 8;
    output;
  end;
run;
```

Another method for getting this shell dataset is as follows:

```
Data ClassData;
  do visit=1 to 8;
    output;
  end;
run;
proc summary data=lab nway classdata=ClassData;
  by pt;
  class visit;
  output out=VisitFrame(drop=_:);
run;
```

We then use the VisitFrame data set created above with each patient (pt) and the visits they should have (visits 1-8) to update the lab data and create the LOCF variable in one data step

The following is the DOW data step that accomplishes this update

```
data work.locf;
   do until(last.pt);
      merge VisitFrame
                Lab;
      by pt visit;
      if missing(labstd)=0 then locfLabStd=labstd;
      output;
   end;
run;
```

Another possible way to do this not using DOW logic is as follows:

```
data work.locf2;
  merge work.visitFrame
        work.lab;
  by pt visit;
  retain locfLabStd;
  if missing(labstd)=0 then locfLabStd=labstd;
  if first.pt & missing(labstd) then locfLabStd=.;
run;
```

Note that when using the DOW It is not necessary to issue a retain statement and the adjustment for the first patient record being missing is not necessary.

EXAMPLE 4: CHANGE FROM BASELINE USING DOUBLE DOW LOOP

This example shows how one can do the change from baseline calculations using a double DOW loop. In this solution two DOW loops are used in the same data step. The first loop creates the baseline variables using any method you choose, while the second loop does the change from baseline calculations. We'll use the same sample data as in example 2.

Below is the code for using the last value on or before treatment as baseline

```
data Change;
  do until(last.subject);
    set BP;
    by subject visit;
    if vsdt<=tmtstdt then do;
      b_sysbp=sysbp;
      b_diabp=diabp;
    end;
  end;

  do until(last.subject);
    set BP;
    by subject visit;
    if vsdt>tmtstdt then do;
      c_sysbp=sysbp-b_sysbp;
      c_diabp=diabp-b_diabp;
    end;
    output;
  end;
run;
```

This code produces exactly the same output data as example 2 would if we eliminated the step that clears the baseline values from the visits prior to the baseline visits. The first DOW loop outputs the last value on or before treatment start date as the baseline for each BP measure for each subject. These values are then retained for each record for each subject in the second DOW loop which then uses them for the change from baseline calculations.

Sometimes one wishes to use the mean of all values on or before treatment start or some other method for calculating baseline values rather than just taking the most recent value, In these cases, the desired algorithm for determining the baseline value should be put into the first DOW loop in the data step. Below is the code for using the mean of all values on or before treatment start as baseline:

```
data Change;
  do until(last.subject);
    set BP;
    by subject visit;
    if first.subject then denom=0;
    if vsdt<=tmtstdt then do;
      denom+1;
      s_sysbp=sum(s_sysbp,sysbp);
      s_diabp=sum(s_diabp,diabp);
    end;
  end;
  b_sysbp=s_sysbp/denom;
  b_diabp=s_diabp/denom;
```

```
  do until(last.subject);
    set BP;
    by subject visit;
    if vsdt>tmtstdt then do;
      c_sysbp=sysbp-b_sysbp;
      c_diabp=diabp-b_diabp;
    end;
    output;
  end;
run;
```

One note of caution in using more than one DOW loop in the same data step is that both loops must read exactly the same number of observations from their respective datasets in order for the loops to be synched up correctly. The details of this synchronization and what occurs if it is violated can be found in the Dorfman/Shajenko paper [4].

EXAMPLE 5: LOCF AND CHANGE FROM BASELINE USING DOUBLE DOW LOOP

As an extension of the above example let's now look at a data step that will do both the LOCF and the change from baseline calculations using a double DOW loop. Here we modify the data for example 2 so that some values are missing.

```
data BP;
  input subject visit @7 vsdt mmddyy10. @18 tmtstdt mmddyy10. sysbp diabp @;
  datalines;
1  1  09/15/2007 10/15/2007 140 90
1  3  10/15/2007 10/15/2007 138 94
1  4  10/29/2007 10/15/2007 142 .
1  5  11/13/2007 10/15/2007 130 80
1  6  11/29/2007 10/15/2007 132 84
1  7  12/15/2007 10/15/2007   . 83
1  8  12/30/2007 10/15/2007 131 81
1  9  01/13/2008 10/15/2007 128  .
2  1  09/01/2007 09/30/2007 122 70
2  2  09/15/2007 09/30/2007 125 73
2  3  09/30/2007 09/30/2007   . 80
2  4  10/14/2007 09/30/2007 130 78
2  5  10/30/2007 09/30/2007 128  .
2  6  11/13/2007 09/30/2007 126  .
2  7  11/28/2007 09/30/2007   . 76
2  9  12/29/2007 09/30/2007 124 80

;;;;
run;
```

Similar to example 3 we need to create a shell for the visits to carry values forward into as follows:

```
data shell;
  set BP(keep=subject tmtstdt);
  by subject;
  if first.subject then do visit=1 to 9;
    output;
  end;
run;
```

Now we apply the double DOW logic to the case where we use the last value on or before treatment as the baseline and wish to carry forward values from previous observations in the cases of missing values. Once again the baseline calculations are done in the first DOW loop. Inside the loop the BP dataset is merged with the shell so that we have the same number of observations in this loop as we'll need to the LOCF to be done in the second DOW loop in the data step.

```
data Change;
  do until(last.subject);
    merge Shell
          BP;
    by subject visit;
    if .<vsdt<=tmtstdt then do;
      if missing(sysbp)=0 then b_sysbp=sysbp;
      if missing(diabp)=0 then b_diabp=diabp;
    end;
  end;

  do until(last.subject);
    merge Shell
          BP;
    by subject visit;
    if vsdt>tmtstdt then do;
      c_sysbp=sysbp-b_sysbp;
      c_diabp=diabp-b_diabp;
    end;
    else do;
      c_sysbp=.;
      c_diabp=.;
    end;
    if missing(sysbp)=0 then l_sysbp=sysbp;
    if missing(diabp)=0 then l_diabp=diabp;
    if missing(c_sysbp)=0 then l_c_sysbp=c_sysbp;
    if missing(c_diabp)=0 then l_c_diabp=c_diabp;
    output;
  end;

run;
```

Note that the LOCF is done in the second DOW loop after the change from baseline calculations are done so that these can also be carried forward when missing. The "else do" loop to set the changes to missing is necessary to keep these values from being retained on visits created in shell for LOCF values.

The outcome for this data step is as follows:

SUBJECT	TMTSTDT	VISIT	VSDT	SYSBP	DIABP	B_SYSBP	B_DIABP	C_SYSBP	C_DIABP	L_SYSBP	L_DIABP	L_C_SYSBP	L_C_DIABP
1	10/15/2007	1	09/15/2007	140	90	138	94	.	.	140	90	.	.
	10/15/2007	2	.	.	.	138	94	.	.	140	90	.	.
	10/15/2007	3	10/15/2007	138	94	138	94	.	.	138	94	.	.
	10/15/2007	4	10/29/2007	142	.	138	94	4	.	142	94	4	.
	10/15/2007	5	11/13/2007	130	80	138	94	-8	-14	130	80	-8	-14
	10/15/2007	6	11/29/2007	132	84	138	94	-6	-10	132	84	-6	-10
	10/15/2007	7	12/15/2007	.	83	138	94	.	-11	132	83	-6	-11
	10/15/2007	8	12/30/2007	131	81	138	94	-7	-13	131	81	-7	-13
	10/15/2007	9	01/13/2008	128	.	138	94	-10	.	128	81	-10	-13
2	09/30/2007	1	09/01/2007	122	70	125	80	.	.	122	70	.	.
	09/30/2007	2	09/15/2007	125	73	125	80	.	.	125	73	.	.
	09/30/2007	3	09/30/2007	.	80	125	80	.	.	125	80	.	.
	09/30/2007	4	10/14/2007	130	78	125	80	5	-2	130	78	5	-2
	09/30/2007	5	10/30/2007	128	.	125	80	3	.	128	78	3	-2
	09/30/2007	6	11/13/2007	126	.	125	80	1	.	126	78	1	-2
	09/30/2007	7	11/28/2007	.	76	125	80	.	-4	126	76	1	-4
	09/30/2007	8	.	.	.	125	80	.	-4	126	76	1	-4
	09/30/2007	9	12/29/2007	124	80	125	80	-1	0	124	80	-1	0

CONCLUSION

The DOW-Loop is an extremely powerful programming technique which

1. gives the programmer more control over the input and retention of variables
2. allows the programmer to create a number of variables for each by-group and output these for the last record of the by-group.
3. can reduce the number of passes through a dataset required by a program resulting in faster and more efficient programs.

Though these methods may at first seem peculiar, especially to the novice programmer, once they have been understood and mastered they can provide a very simple technique for programming a variety of situations involving by-group processing.

REFERENCES

1. *2 PROC TRANSPOSEs = 1 DATA Step DOW-Loop*, Nancy Brucken, PharmaSUG 2007 Proceedings

2. *One-Step Change from Baseline Calculations*, Nancy Brucken, PharmaSUG 2008 Proceedings

3. *The DOW (not that DOW!!!) and the LOCF in Clinical Trials*, Venky Chakravarthy, SUGI 28 Proceedings

4. *The DOW loop unrolled,* Paul Dorfman & Lessia Shajenko, PharmaSUG 2008 Proceedings

ACKNOWLEDGMENTS

Thanks to Nancy Brucken for bringing this technique to my attention.

RECOMMENDED READING

SAS-L archives, especially posts by Paul Dorfman, Ian Whitlock, Richard DeVenezia, Ken Borowiak, Howard Schreier and Muthia Kachirayan – among others – on this subject.

CONTACT INFORMATION

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

Richard Read Allen
Peak Statistical Services
5691 Northwood Drive
Evergreen, CO 80439
E-mail: rrallen@peakstat.com
Web: www.peakstat.com