PharmaSUG 2016 – Paper BB08

Novel Programming Methods for Change from Baseline Calculations

Mina Chen, Roche Product Development in Asia Pacific, Shanghai, China
Peter Eberhardt, Fernwood Consulting Group Inc.

ABSTRACT

In many clinical studies, change from baseline is a common measure of safety and/or efficacy in in the clinical data analysis. There are several ways to calculate changes from baseline in a vertically structured data set such as Retain statement, Arrays, DO-Loop in DATA steps or PROC SQL. However, most of these techniques require operations such as sorting, searching, and comparing. As it turns out, these types of techniques are some of the more computationally intensive and time-consuming. Consequently, an understanding of these techniques and a careful selection of the specific method can often save the user a substantial amount of computing resources.

This paper will demonstrates a novel way of calculating change from baseline using Hash objects.

INTRODUCTION

In many clinical trials, the primary focus is whether treatment groups differ with respect to the change from baseline to end of therapy in a continuous response variable. In a randomized clinical trial, we often use a repeated measures design in which subjects are measured at fixed times throughout the study. Then we usually use the change-from-baseline data set to see what effect some therapeutic intervention had on some kind of diagnostic measure, e.g. temperature, blood pressure. A measure is taken before and after therapy, and a difference is calculated for each post-baseline measure.

For years, the most common method for computing change from baseline was done in SAS® by first physically dividing the original data set into baseline and post-baseline data sets. The baseline value may then be obtained from a single point in time, often from the last value measured before the first dose of study drug, or may be a composite of several observations perhaps the mean of the last two measurements before the start of study drug administration. A data set containing the baseline value for every subject is created, and then merged with the post-baseline data set by subject and diagnostic parameter. Then calculate the change from baseline line per patients and parameters.

In SAS 9.2, the hash object was introduced, giving users another method for quickly calculating change from baseline value. This paper will introduce how hash object works, the syntax required, and simple applications of it use in real studies.

Example Tables

We are going to refer to the vital signs data set. In this example dataset, we collected the measured results of diastolic blood pressure and systolic blood pressure for each subject at each visit. The change from baseline value will help our clinical scientist to find out if there are some abnormal changes of diagnostic parameters from baseline.

So, the original dataset looks like this (Figure 1):

usubjid	PARAMCD	PARAM	VISITNUM	VISIT	AVAL
11110	DIABP	Diastolic Blood Pressure	1	SCREENING	86
11110	DIABP	Diastolic Blood Pressure	2	BASELINE	86
11110	DIABP	Diastolic Blood Pressure	3	WEEK 1	81
11110	DIABP	Diastolic Blood Pressure	4	WEEK 2	82
11110	DIABP	Diastolic Blood Pressure	5	WEEK 3	82
11110	DIABP	Diastolic Blood Pressure	6	WEEK 4	83
11110	DIABP	Diastolic Blood Pressure	7	WEEK 5	82
11110	DIABP	Diastolic Blood Pressure	37	FOLLOW UP	84
11110	SYSBP	Systolic Blood Pressure	1	SCREENING	120
11110	SYSBP	Systolic Blood Pressure	2	BASELINE	117
11110	SYSBP	Systolic Blood Pressure	3	WEEK 1	121
11110	SYSBP	Systolic Blood Pressure	4	WEEK 2	123
11110	SYSBP	Systolic Blood Pressure	5	WEEK 3	126
11110	SYSBP	Systolic Blood Pressure	6	WEEK 4	125
11110	SYSBP	Systolic Blood Pressure	7	WEEK 5	121
11110	SYSBP	Systolic Blood Pressure	37	FOLLOW UP	126

Figure 1- The original data

The final data set (Figure 2) contains change-from-baseline values looks something like this:

usubjid	PARAMCD	PARAM	VISITNUM	VISIT	AVAL	base	chg
11110	DIABP	Diastolic Blood Pressure	1	SCREENING	86	.	.
11110	DIABP	Diastolic Blood Pressure	2	BASELINE	86	86	0
11110	DIABP	Diastolic Blood Pressure	3	WEEK 1	81	86	-5
11110	DIABP	Diastolic Blood Pressure	4	WEEK 2	82	86	-4
11110	DIABP	Diastolic Blood Pressure	5	WEEK 3	82	86	-4
11110	DIABP	Diastolic Blood Pressure	6	WEEK 4	83	86	-3
11110	DIABP	Diastolic Blood Pressure	7	WEEK 5	82	86	-4
11110	DIABP	Diastolic Blood Pressure	37	FOLLOW UP	84	86	-2
11110	SYSBP	Systolic Blood Pressure	1	SCREENING	120	.	.
11110	SYSBP	Systolic Blood Pressure	2	BASELINE	117	117	0
11110	SYSBP	Systolic Blood Pressure	3	WEEK 1	121	117	4
11110	SYSBP	Systolic Blood Pressure	4	WEEK 2	123	117	6
11110	SYSBP	Systolic Blood Pressure	5	WEEK 3	126	117	9
11110	SYSBP	Systolic Blood Pressure	6	WEEK 4	125	117	8
11110	SYSBP	Systolic Blood Pressure	7	WEEK 5	121	117	4
11110	SYSBP	Systolic Blood Pressure	37	FOLLOW UP	126	117	9

Figure 2- The results.

Dynamic method using Hash Object

1. Simple Change from Baseline

As statistical programmers we are always interested in learning techniques that will help us improve the performance of data step operations and work efficiency. SAS software supports a DATA step programming technique known as a hash object to associate a key with one or more values. For our purposes, think of the hash table as a look-up table.
The following code makes use of the HASH OBJECT to compute change from baseline and generate a data set containing change from baseline for all post-baseline measurements in a single DATA step:

```
      data _vs;
❶     set vsdata;
      if _n_=1
      then
        do;
❷           declare hash base(dataset:"vsdata(where=(visitnum=2)
                                          rename=(aval=baseline))");
                  base.definekey('paramcd','usubjid');
                  base.definedata('baseline');
                  base.definedone();
        end;
```

```
❸      rc= base.find();
       if rc=0
❹         then chg=(aval-baseline);
       drop rc;
       run;
```

In the program above, we have one hash tables: BASE. The BASE hash table contains only one entry (i.e. one "row") per diagnostic parameter (`base.definekey('paramcd'))` with one data variable - SEQ (`base.definedata('seq'`).

1. `if_n_ = 1`
   The HASH object should only be declared once in the DATA step program. We do this in the _n_ = 1 block.

2. ```
   declare hash base(dataset:"vsdata(where=(visitnum=2)
                                      rename=(aval=baseline))");
               base.definekey('paramcd','usubjid');
               base.definedata('baseline');
   ```
   here we are declaring the HASH object and loading data in one step. We can use dataset options to control the data; here we are using a WHERE clause to only load baseline visits (`visitnum=2`), and the RENAME option to rename the data measure of interest to `baseline`. We need to rename it so we do not overwrte the valueof `aval` in the PDV when we retrieve a record from the hash table. The hash table has two keys (`base.definekey('paramcd','usubjid');`) and one data value (`base.definedata('baseline');`). Each time we look up the parameter and subject in the hash table we wil get the baseline value returned.

3. ```
   rc= base.find();
   if rc=0
   ```
   the `base.find()` method will use the current value in the PDV for paramcd and usubjid to search the hash table. After the lookup, we check if key values were found (`if rc=0`). Note here that we are checking for a value of zero (0) for success; this is different than usual SAS processing were we use zero to indicate faliure and non-zero to indicate success,

4. `then chg=(aval-baseline);`
   if the subject/parameter was found in the hash table, the current value of baseline in the PDV is overwritten; we use this value to calculate the change. If the subject/parameter was not found the value for baseline in the PDV would not be changed – it would have the value from the last successful lookup. Note that if we did not check the return code and the subject/parameter was not found, we would calculate a change based in incorrect data

In summary, we created a hash object that contained only baseline visit values with one entry for each parameter/subject. We used dataset options to limit the hash to baseline visits, and to rename the relevant measure to 'baseline'. The data step will iterate over all rows in the visit dataset (`set vsdata`) with each iteration updating the PDV with values from the dataset. The `base.find()` method will take the current values in the PDV of the hash key and search the hash table. If the search is successful, the value of `baseline` in the PDV is overwritten; if the search is not successful the value for `baseline` in the PDV is not overwritten. This highlights the importance of checking the return code from the hash methods before using data values returned from the hash table.

2. Repeated Measure Change from Baseline

Sometimes we use repeated measures design in which subjects are measured triple at each visit to avoid any random errors. In this situation, we usually use means procedure to calculate the means for baseline and post-baseline visits separately, and then merge the two dataset together in the end and calculated changes.

Although it is easy to understand (as it should be as being traditional), it requires multiple DATA and PROC steps. The number of steps does not stay the same – rather it varies depending on the problem. Below is an extract of a table with repeated measures. In this example we need to first calculate the average of the parameter for each visit, then use the average values to calculate the change from baseline.

usubjid	paramcd	param	visit	visitnum	aval
11110	DIABP	Diastolic Blood Pressure	Screening	1	86
11110	DIABP	Diastolic Blood Pressure	Screening	1	84
11110	DIABP	Diastolic Blood Pressure	Screening	1	83
11110	DIABP	Diastolic Blood Pressure	BASELINE	2	86
11110	DIABP	Diastolic Blood Pressure	BASELINE	2	86
11110	DIABP	Diastolic Blood Pressure	BASELINE	2	82
11110	DIABP	Diastolic Blood Pressure	WEEK 1	3	81
11110	DIABP	Diastolic Blood Pressure	WEEK 1	3	83
11110	DIABP	Diastolic Blood Pressure	WEEK 1	3	85
11110	DIABP	Diastolic Blood Pressure	WEEK 2	4	82
11110	DIABP	Diastolic Blood Pressure	WEEK 2	4	88
11110	DIABP	Diastolic Blood Pressure	WEEK 2	4	87
11110	DIABP	Diastolic Blood Pressure	WEEK 3	5	82
11110	DIABP	Diastolic Blood Pressure	WEEK 3	5	86
11110	DIABP	Diastolic Blood Pressure	WEEK 3	5	82

The following code demonstrate how to calculate average values for each visit within one data step.

```
data _vs;
      set vsdata1 end=done;
      if _n_ = 1 then
      do;
      declare hash h(dataset:'vsdata1',multidata:'y');
❶           h.definekey('paramcd','visitnum','usubjid');
            h.definedata(all:'y');
            h.definedone();
❷     call missing (meanval);
      declare hash avg( ORDERED:'A');
            avg.definekey('usubjid','paramcd','visitnum');
            avg.definedata('meanval','paramcd','visitnum','usubjid');
      /*step through the hash object once we've cteated it*/
❸     declare hiter hi_avg('avg');
            avg.defneDone();
❹     declare hash baseline();
            baseline.definekey('paramcd','usubjid');
            baseline.definedata('baseval','paramcd','usubjid');
            baseline.definedone();
      end;
      sum = 0;
      count = 0;
❺     if ~h.find() then
      do;
            sum + aval;
            count + 1;
      end;
❻     do rc = h.find_next() while(~rc);
❼           do while(rc=0 );
                  count+1;
                  sum+aval;
                  rc = h.find_next();
            end;
      end;
      meanval = sum / count;
❽     rc=avg.add();
❾     if visitnum=2 then
      do;
            baseval=meanval;
            rc=baseline.add();
      end;
      if done then
❿     do;
            avg.output(dataset:"avg");
⓫           baseline.output(dataset:"baseline");
⓬           rc=hi_avg.first();
            do while (rc = 0);
```

```
⓭                   rc=baseline.find();
                    chg=meanval-baseval;
                    output;
⓮                   rc = hi_avg.next();
              end;
       end;
       drop rc count sum aval;
Run;
```

In the above code we defined a hash object h, which we use to calculate the average measure in one pass of the table.

1. `declare hash h(dataset:'vs',multidata:'y');`

   Since we can have multiple measure for each visit, we cannot use `paramcd` and `visitnum` alone to uniquely identify a row. We can use the `multidata` option to tell SAS we can have more than one row for each key element. In addition, we are telling SAS to read the table `vs` into the hash table once it is created

   `h.definedata(all:'y');`

   this is a shortcut telling SAS we want all the variables in the table to be data elements of the hash table. Although this is a useful shortcut, you need to be careful you do not use it when there are many variables in the table and you only need a few.

2. `sum = 0; count = 0;`

   initialize the variables for calculating the mean,

3. `if ~h.find() then do;`

   the find method will use the current value of the key fields in the PDV and return the data values to the PDV id the key was found in the hash table. The hash table returns a 0 if the key was found; the `~h.find()` is saying “if we found the key fields, do the following block”. In the block we are accumulation aval and incrementing the count for each paramenter/visit date measure.

4. `do rc = h.find_next() while(~rc);`

   The find_next() method will look for the next value of the key fields (paramenter/visit date) and return the data elements to the PDV if the find was successful. Since a return code of 0 indicates success, the `while(~rc)` tells SAS to keep executing the find_next as long as there are more records for the key values.

5. `do while(rc=0 );`

   this loop accumulates the value and increments the count, then looks for another entry in the hash table.

6. `do rc = h.find_next() while(~rc);`

   this initiates the pass through the data. Sine we have can have multiple key values (multiple measures for each visitnum) we use the find_next() method.

7. do while(rc=0);

   as long as there are more key values (rc=0) then iterate to accumulate the values and count;

8. ```
   meanval = sum / count;
   rc=avg.add();
   ```

   calculate the average and add it to the avg hash

9. `if visitnum=2 then`

   if the current row is the baseline, then save the values in the baseline hash.

   ```
   baseval=meanval;
   rc=baseline.add();
   ```

10. `if done then`

    when all the rows of the input table are read (`set vsdata1 end=done;`) complete the processing
   ```

11. `avg.output(dataset:"avg");`
    `baseline.output(dataset:"baseline");`

    write the contents of the avg and the baseline hashes to a table; this is not necessary, but a useful validation check.

12. `rc=hi_avg.first();`

    start to read over the avg iterator

13. `rc=baseline.find();`

    use the values in the avg to get the matching baseline values. Use these to calculate the change from baseline.

14. `rc = hi_avg.next();`

    Continue the traversal of the avg iterator.

How Does a Hash Object Work?

A hash object is an in-memory structure that encapsulates data, and methods to access these data. As SAS programmers, the key to the above statement is “in memory”. In memory access to data means fast access to data. In memory access also means transient access to data; that is, like a SAS array, the hash object itself is not saved. Since the hash object has both data and methods to access the data it can exist only in a programming environment; this means the hash object is available only in the DATA step.

In simple terms, we can think of the hash object as a table in a normalized relational database system: there are key columns and associated data columns. To access the data columns we need to look-up the row using the key columns. In the case of the hash object, there are methods used to perform this look-up; as we saw above, the most common method is the `find()` method. We also saw that in one DATA step we can have multiple hash objects, and each has its own `find()` method.

From the examples above, we can also see the hash object is tightly coupled with the PDV. The hash object knows about itself, so when we invoke `base.find()`, the base object knows you want to retrieve the value of `seq` for a specific `paramcd;` since the method needs a value for `paramcd,` it uses the current value of `paramcd` in the PDV to do the lookup. When this `paramcd` is found in the hash object, the value of `seq` is copied from the hash object to the `seq` variable in the PDV, overwriting the PDV value; the method lets the program know it was successful by returning a code of 0 (zero) . On the other hand, if the value of `paramcd` was not found the value of the `seq` variable in the PDV is not overwritten; the method lets the program know it was not successful by returning a non-zero value. This means it is very important that the return code be checked after method calls to ensure the appropriate action is taken after the method returns,

This has been a very summary overview of how the hash object works. For more detailed information see Eberhardt 2010 and Dorfman and Eberhardt 2010.

Why Hashing

In SAS, it is rare there is only one way to solve a problem; the hash object provides the SAS programmer with a new tool to solve problems. Because it is in memory, it is fast. Because it is based on object oriented concepts, it is a flexible and powerful tool. If your applications demand performance, then you should consider hashing.

The variety of methods available means you can perform complex, cross record comparisons and calculations with only one pass of the data. Because hash methods return success/failure codes, the DATA step language allows us to more easily implement complex logic and validation checks. If your applications require complex multi-pass processing, you should consider hashing.

Our data are diverse, different sources, different keys yet we need to join them. DATA step merge requires sorting and possibly renaming variables to effectively join tables. SQL can join many tables in one pass, but can only create one output table. If you need to join many diverse tables and create multiple output tables, you should consider hashing.

If you want to write better, more robust programs, you should consider hashing,

Conclusion

By using not just one, but multiple hash objects we can read a table into memory then traverse the table to create our summary statistics. In addition, we can then spread these values across all the rows in the dataset If needed.

All of this can be done on many ways, usually with many steps. The hash object allows us to use one data step to do all of this.

References

Dorman, Paul and Peter Eberhardt 2010, "*Two Guys on Hash*", Proceedings of the South East SAS User Group 2010 Conference.

Eberhardt, Peter. 2010. "*The SAS Hash Object: It's Time to .find() Your Way Around",* Proceedings of the SAS Global Forum 2010 Conference

SAS Institute Inc. (2012), SAS 9.3 Language Reference: Concepts, Second Edition. Available at: http://support.sas.com/documentation/cdl/en/lrcon/65287/HTML/default/viewer.htm#n1b4cbtmb049xtn1vh9x4waiioz4.htm

Leads and Lags: Static and Dynamic Queues in the SAS® DATA STEP. Available at: http://support.sas.com/resources/papers/proceedings15/3372-2015.pdf

Contact Information

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

Name: Mina Chen
Enterprise: Roche (China) Holding Ltd.
Address: 6th Floor, 3rd Tower, German Center, No. 88, Ke Yuan Road
City, State ZIP: Shanghai, China
Work Phone: +86-021-2892 3475, 201203
E-mail: mina.chen@roche.com

Name: Peter Eberhardt
Enterprise: Fernwood Consulting Group Inc/
Address: 301-50 Prince Arthur Ave, Toronto, ON M5R 1B5 Canada
Work Phone: (416)429-6705
Email: peter@fernwood.ca
Web: www.fernwood.ca