PharmaSUG 2019 - Paper DV-005

Back to the Future: Heckbert's Labeling Algorithm

Christopher J. Smith, Cytel Inc.

ABSTRACT

There are several different axis tick mark algorithms in existence. We will discover one of these by time travelling back to 1990 to learn about Heckbert's nice numbers algorithm for labeling graph axes. Then, we will travel back to the future to proactively apply this algorithm to clinical data, using ODS Graphics and DYNAMIC variables from the Graph Template Language (GTL). Lastly, we explore other uses of DYNAMIC variables to provide data-driven solutions to graphical outputs. SAS® 9.4 M2 was used in the examples presented. This paper is written for the intermediate to advanced SAS users. In particular, it assumes familiarity with the SGPLOT procedure and GTL.

INTRODUCTION

When developing programs to analyze clinical trial data, we often do not have all the data at the time of program development. Therefore, we must create robust programs that implement defensive coding techniques to handle incoming data. In particular with graphical outputs, SAS does not always choose the most aesthetically pleasing tick mark values. In addition, we will not have the full range of data to properly hardcode the tick mark values ourselves. In this paper, we will learn about Heckbert's labeling algorithm to dynamically calculate nicely-numbered, data-driven tick marks for either the x-axis or y-axis. Then, we will use these calculated tick marks in various examples using the SGPLOT procedure and using DYNAMIC variables from the Graph Template Language (GTL). Beyond Heckbert's algorithm, we will discuss a few other data-driven applications of DYNAMIC variables.

HECKBERT'S LABELING ALGORITHM

When selecting tick marks ourselves, we tend to gravitate towards increments 1, 2, or 5 in powers of ten. For example, a temperature data range from 97.9 to 100.4, we may choose tick marks from 97.5 to 100.5 in increments of 0.5. For alanine aminotransferase (ALT), a data range might be 7 to 56 and we choose tick marks from 0 to 60 in increments of 10. Let's get in the DeLorean and travel back to 1990 when Paul Heckbert developed an algorithm to calculate these tick marks in a data-driven fashion (Heckbert, 1990, pp. 61-63). He called them "nice numbers." Next, we will present an implementation of this algorithm using SAS software.

First, we need to calculate the minimum and maximum of the data we want to graph. Suppose we already calculated the Mean ± Standard Deviation (SD) of plutonium levels at each visit in a data set called _STATS shown in Table 1.

PARAMCD	PARAM	TRTP	AVISITN	MEAN	MEAN_LOWER	MEAN_UPPER
PLUTO	Plutonium (g)	Active	0	60.19	56.767	63.611
PLUTO	Plutonium (g)	Active	2	58.29	54.443	62.132
PLUTO	Plutonium (g)	Active	3	59.99	56.609	63.385
PLUTO	Plutonium (g)	Active	…	…	…	…

Table 1. Statistic Calculations in _STATS

Ultimately, we will be doing a series plot of the mean with error bars for ± SD. To find the range of the data being plotted, we need to find the maximum of MEAN_UPPER and the minimum of MEAN_LOWER across all visits and all treatment groups. There are many ways to accomplish this including the SQL, MEANS, and UNIVARIATE procedures. Here, we chose to use SQL:

```
PROC SQL;
   CREATE TABLE minmax AS
   SELECT MIN(mean_lower) AS statmin, MAX(mean_upper) AS statmax
   FROM _STATS(WHERE=(paramcd='PLUTO'));
QUIT;
```

This SQL procedure will yield a data set containing one observation with two variables representing the overall minimum and maximum of the data to be plotted, see Table 2.

STATMIN	STATMAX
51.982	65.595

Table 2. Minimum and Maximum of Points to be Plotted

Heckbert's nice number algorithm requires a minimum, a maximum, and a desired number of tick marks as inputs. The results of using this algorithm are a new minimum, a new maximum, and a step increment. The following DATA step carries out Heckbert's algorithm:

```
DATA _NULL_;
   SET minmax;

   desired_num_of_ticks=9;  ❶

   data_range= statmax -statmin;
   exponent=FLOOR(LOG10(data_range));
   fraction_part=data_range/(10**exponent);
        if fraction_part <=1 then new_fraction_part=1;        ❷
   else if fraction_part <=2 then new_fraction_part=2;
   else if fraction_part <=5 then new_fraction_part=5;
   else new_fraction_part=10;
   intermediate_range=new_fraction_part*(10**exponent);

   intermediate_tick_step=intermediate_range/(desired_num_of_ticks-1);
   tick_exponent=FLOOR(LOG10(intermediate_tick_step));
   tick_fraction_part=intermediate_tick_step/(10**tick_exponent);
        if tick_fraction_part <1.5 then new_tick_fraction_part=1;     ❸
   else if tick_fraction_part <3 then new_tick_fraction_part=2;
   else if tick_fraction_part <7 then new_tick_fraction_part=5;
   else new_tick_fraction_part=10;
   nice_tick_increment= new_tick_fraction_part*(10**tick_exponent);

   nicemin=FLOOR(statmin/nice_tick_increment)*nice_tick_increment;  ❹
   nicemax=CEIL(statmax /nice_tick_increment)*nice_tick_increment;

   CALL SYMPUTX('graph_min', nicemin);
   CALL SYMPUTX('graph_max', nicemax);                 ❺
   CALL SYMPUTX('graph_by', nice_tick_increment);
RUN;
```

❶ Heckbert's algorithm requires a desired number of tick marks. Here, we chose 9. Note that, the desired number of tick marks does not guarantee the exact number of tick marks.

❷ Next, we calculate the actual range, 13.613. Then, find a temporary range that is a power of ten of 1, 2, or 5 which covers the size of the actual range. The temporary range is 20.

❸ Using the temporary range of 20, we use the desired number of tick marks to divide it into small segments of 2.5. This number gets compared against key cutoff values of 1.5, 3, and 7 to determine the step increment. For our example, the step increment is 2. Note that, the pieces of code using

FLOOR, LOG10, and 10$^{tick\_exponent}$ are adjusting for magnitude. That is, if the temporary range was 20, 200, or 2000, the step increment will become 2, 20, or 200 respectively.

❹ We use the tick mark increment to derive nice minimum and nice maximum. Using the FLOOR and CEIL functions ensures the minimum and maximum, respectively, are in stride with the nice tick mark increment. For our example, the nice minimum and nice maximum are 50 and 66, respectively.

❺ In order to use the nice minimum, maximum, and tick increment values in the SGPLOT procedure, we used CALL SYMPUTX to store them as macro variable values.

Let's back up the DeLorean and suppose we used the data in Table 1 to graph a series plot of the plutonium levels using the SGPLOT procedure shown in Figure 1. Notice that, the default y-axis tick marks are not that desirable. Although, the increments are nice number increments of 5, our data extends far beyond the minimum tick value being plotted. Thus, we don't have a good reference point for the values plotted near the bottom of the graph.

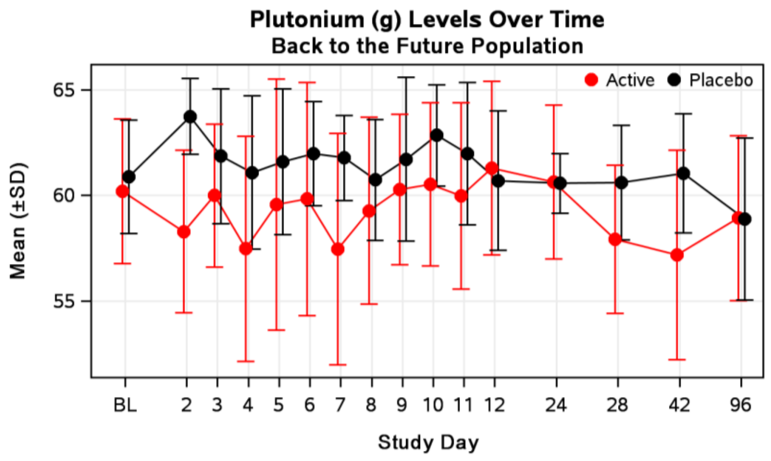

Figure 1. Default Tick Marks

We could slightly improve on the SAS default tick values by using the options THRESHOLDMIN=1 and THRESHOLDMAX=1 on the YAXIS statement to guarantee the tick marks extend beyond the data values:

```
YAXIS <other yaxis options> THRESHOLDMIN=1 THRESHOLDMAX=1;
```

However, as we see in Figure 2 below, this creates a fairly large amount of white space above and below the true data range.

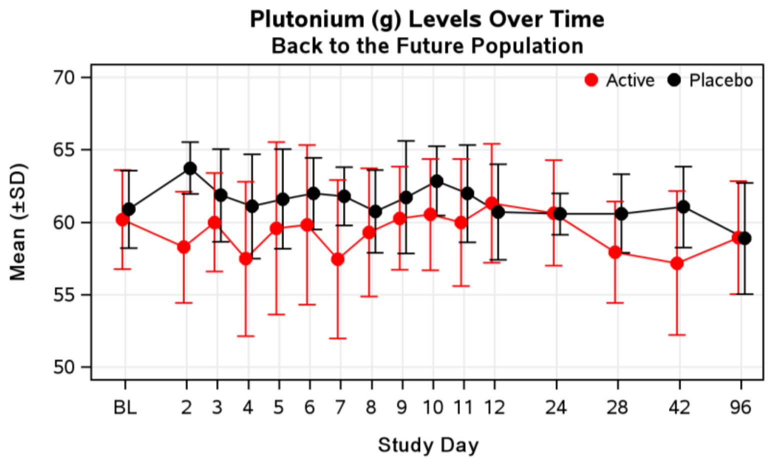

Figure 2. THRESHOLDMIN and THRESHOLDMAX Options

To apply Heckbert's nice number tick marks, we can add the VALUES= option on the YAXIS statement in the SGPLOT procedure and reference our macro variables created in the DATA _NULL_ step above:

```
      YAXIS <other yaxis options>
            VALUES=(&graph_min. TO &graph_max. BY &graph_by.);
```

Remember, the nice number tick marks are from 50 to 66 with a nice step increment of 2. Figure 3 below shows our data are bounded by the y-axis with some of the white space minimized compared to using the THRESHOLDMIN= and THRESHOLDMAX= options. For a more thorough look into the axis threshold options, see (SAS User's Guide, 2016, pp. 108-112).

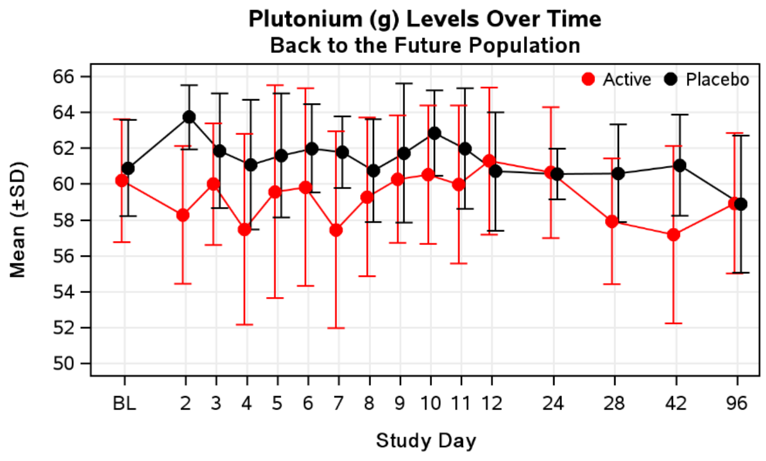

Figure 3. Heckbert's Nice Numbers Applied to Y-axis.

For figure outputs that are one page, our approach to implementing Heckbert's algorithm is sufficient. However, we might be tasked with creating multi-page figure report (e.g. all hematology lab tests, one page per lab parameter). Suppose we wanted to do BY group processing (e.g. `BY PARAM;`). Looking back at our VALUES= option of the YAXIS statement above, it is not possible to provide different tick mark values for different parameters. One approach might be to create a SAS macro and %DO loop through each parameter one by one, calculate tick marks, create the plot, and iterate to the next parameter. But, we will take a more interesting approach; we will travel back to the future to learn about DYNAMIC variables in GTL in hopes of a data-driven solution.

USING HECKBERT WITH GRAPH TEMPLATE LANGUAGE

The TEMPLATE procedure gives us access to GTL. We will mainly discuss aspects of GTL to allow us to accomplish the goal of data-driven axis tick marks by lab parameter. To get started, using GTL is a two-step process (Matange, 2013, p. 11). First, we need to compile a template. The usual structure is shown below using STATGRAPH template:

```
   PROC TEMPLATE;
      DEFINE STATGRAPH template-name;
         BEGINGRAPH / <options>;
            <Graph Template Language Statements Defining the Graph>
         ENDGRAPH;
      END;
   RUN;
```

Second, we need to render the template using the SGRENDER procedure:

```
PROC SGRENDER DATA=data-set-name TEMPLATE=template-name;
   <optional statements>
RUN;
```

This step creates the image; the first step only saves a template to be used later:

Our previous SGPLOT procedure efforts were not all for naught. Under the hood, all the SG- procedures including SGPLOT, SGPANEL, and SGSCATTER are actually GTL! That is, when we write SGPLOT code and options, we are tweaking the underlying GTL syntax. To convert SGPLOT procedure code to GTL, we could use the TMPLOUT= option on the PROC SGPLOT statement to write the GTL code out to a separate file. This could be used as a starting point for writing a graph template using GTL.

DYNAMIC VARIABLES

When we create templates, we are often providing variable names and hardcoded strings in the template. To make the template more reusable, we introduce DYNAMIC variables. To utilize DYNAMIC variables, we need to make updates to both the TEMPLATE and SGRENDER procedures. The code below illustrates the use of dynamic variables:

```
PROC TEMPLATE;
   DEFINE STATGRAPH __template_name;
      BEGINGRAPH /;
         DYNAMIC _var; ❶
         LAYOUT OVERLAY /;
            HISTOGRAM _var; ❷
         ENDLAYOUT;
      ENDGRAPH;
   END;
RUN;

PROC SGRENDER DATA=sashelp.cars TEMPLATE=__template_name;
   DYNAMIC _var="mpg_city"; ❸
RUN;
```

❶ _VAR is declared as a DYNAMIC variable in the template.

❷ _VAR is referenced in the HISTOGRAM statement of the template.

❸ The variable MPG_CITY in SASHELP.CARS is initialized to the DYNAMIC _VAR variable. Note that, the SAS GTL User's Guide for the DYNAMIC statement reveals:

For dynamic variables that resolve to column names or strings, enclose the value in quotation marks. For dynamic variables that resolve to numeric values, specify the value without quotation marks. (SAS User's Guide, 2016, p. 554)

Since MPG_CITY is a column name, it belongs in quotes.

For referencing numeric or character DYNAMIC variable lists, we may encounter syntax errors like the one shown in Output 1:

```
DYNAMIC __COLOR;
BEGINGRAPH /DATACONTRASTCOLORS=(__COLOR) DATALINEPATTERNS=(solid);
                               ________
                                 999
ERROR 999-580: Syntax error: value is not a valid color or style reference.
WARNING: Object will not be saved.
```

Output 1. Log from TEMPLATE Procedure

When supplying a list to a DYNAMIC variable, parentheses are not used around the reference (SAS Reference, 2016, p. 1352). The correct syntax is:

```
BEGINGRAPH /DATACONTRASTCOLORS=__COLOR DATALINEPATTERNS=(solid);
```

Note that, in the SGRENDER procedure, the DYNAMIC statement will have the list of colors enclosed in quotation marks since it is a string. For example:

```
PROC SGRENDER DATA=sashelp.cars TEMPLATE=__template_name;
   DYNAMIC __COLOR="RED BLUE";
RUN;
```

BACK TO THE FUTURE

With that crash course now in the past, let's revisit our graph of plutonium levels. We will try to implement nice number tick marks from variables stored in a data set. The GTL statement of interest is the YAXISOPTS= and the LINEAROPTS= suboptions. In particular, the TICKVALUESEQUENCE= suboptions. The structure of this option is:

```
TICKVALUESEQUENCE=(start=number end=number increment=number)
```

First, as a stepping stone, we will setup DYNAMIC variables with hardcoded values.

```
PROC TEMPLATE;
  DEFINE STATGRAPH __pluto;
    BEGINGRAPH /;
      DYNAMIC _graph_min _graph_max _graph_by; ❶
      LAYOUT OVERLAY /
        <other overlay  options>
        YAXISOPTS=(<other yaxisopts options>
                   LINEAROPTS=(TICKVALUESEQUENCE=(START=_graph_min
                                                  END=_graph_max
                                                  INCREMENT=_graph_by)
                               VIEWMIN=_graph_min VIEWMAX=_graph_max )); ❷
       <Graph Template Language Plot Statements>
      ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;

PROC SGRENDER DATA=final TEMPLATE=__pluto;
   DYNAMIC _graph_min=50 _graph_max=66 _graph_by=2; ❸
RUN;
```

❶ Three DYNAMIC variables are declared _GRAPH_MIN, _GRAPH_MAX, and _GRAPH_BY.

❷ The three DYNAMIC variables are referenced in the sequence options of TICKVALUESEQUENCE=. Also note that, _GRAPH_MIN and _GRAPH_MAX are referenced in VIEWMIN= and VIEWMAX= options. Without explicitly using these options, SAS determines whether to display the start and end tick marks (SAS Reference, 2016, p. 958). Generally, tick marks are only displayed if they fall within the actual data range. Using VIEWMIN= and VIEWMAX= will ensure the presence of the tick marks.

❸ Hardcoded values are initialized to DYNAMIC variables _GRAPH_MIN, _GRAPH_MAX, and _GRAPH_BY. Since they resolve to numeric values, quotation marks are not used.

As expected, this will create the same graph as Figure 3 above. But, this will set us up to blast into the 21st century in extraordinary fashion. Now, we want to try to pass column variables storing this information instead of the numeric values hardcoded in the DYNAMIC statement of the SGRENDER procedure. Suppose instead of the DATA _NULL_ step above, we kept a WORK data set and merged the nice numbers information back onto our _STATS data set (see Table 3 for the structure of this new SAS data set). Notice, the nice number information is present on every observation.

TRTP	AVISIT	MEAN	MEAN_ LOWER	MEAN_ UPPER	NICEMIN	NICEMAX	NICE_TICK_ INCREMENT
Active	0	60.19	56.767	63.611	50	66	2
Active	2	58.29	54.443	62.132	50	66	2
Active	3	59.99	56.609	63.385	50	66	2
Active	…	…	…	…	50	66	2

Table 3. Statistic Calculations with Nice Number Tick Marks

Now, we will attempt to supply these DATA step variables into our DYNAMIC variables. Recall for column names we must enclose in quotes:

```
PROC SGRENDER DATA=final TEMPLATE=__pluto;
   DYNAMIC graph_min="nicemin"
           graph_max="nicemax"
           graph_by="nice_tick_increment";
RUN;
```

Unfortunately, we fall short of glory and end up back where we started with the default SAS tick marks (i.e. 55, 60, 65) found in Figure 1. Great Scott, what happened? It turns out; we are using an incorrect DYNAMIC variable reference for START= and END= GTL options. The options are ignored and resort to the default because we are trying to supply a variable **name** to options that are expecting numeric values (SAS User's Guide, 2016, p. 555). While our program is ERROR and WARNING free, SAS does provide a hint of resorting back to the default in the log shown in Output 2:

```
NOTE: START=nicemin is invalid. The value cannot be converted to a number.
The default will be used.
NOTE: END=nicemax is invalid. The value cannot ...
```

Output 2. Log from SGRENDER Procedure

If this paper has any crescendo, it is what comes next. Enter Special DYNAMIC Variables! These little gems are greatly undersold in the documentation; "special" is an understatement. We are going to make use of the _BYVAL_ and _BYVAL*n*_ special DYNAMIC variables which represent the **values** of variables specified in the BY statement of the SGRENDER procedure:

```
PROC TEMPLATE;
  DEFINE STATGRAPH __pluto;
    BEGINGRAPH /;
      DYNAMIC _BYVAL_ _BYVAL2_ _BYVAL3_; ❶
      LAYOUT OVERLAY /
        <other overlay  options>
        YAXISOPTS=(<other yaxisopts options>
                   LINEAROPTS=(TICKVALUESEQUENCE=(START=_BYVAL_
                                                  END=_BYVAL2_
                                                  INCREMENT=_BYVAL3_)
                               VIEWMIN=_BYVAL_ VIEWMAX=_BYVAL2_)) ❷
      ;
         <Graph Template Language Plot Statements>
      ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;

PROC SGRENDER DATA=final TEMPLATE=__pluto;
  BY nicemin nicemax nice_tick_increment; ❸
RUN;
```

❶ Special DYNAMIC variables are defined.

❷ The three special DYNAMIC variables are referenced as sequence options of TICKVALUESEQUENCE=.

❸ Data set variables NICEMIN, NICEMAX, and NICE_TICK_INCREMENT are provided in the BY statement of the SGRENDER procedure. These variable values will be used in the place of the _BYVAL_ and _BYVAL*n*_ references. Note that, special DYNAMIC variables do not have to be defined in the SGRENDER procedure (SAS User's Guide, 2016, p. 560).

We now have all the weapons in our arsenal to create a multi-page figure report. We will go back and make a few edits to allow for the by-group processing. First, we need calculate the minimum and maximum data ranges for each parameter. We can do this by incorporating a GROUP BY clause in the SQL procedure that was calculating the minimum and maximum values:

```
PROC SQL;
   CREATE TABLE minmax AS
   SELECT PARAMN, MIN(mean_lower) AS statmin, MAX(mean_upper) AS statmax
   FROM _STATS
   GROUP BY PARAMN;
QUIT;
```

Next, we need to include the GROUP BY variable, PARAMN, in the BY statement of SGRENDER to ensure the parameter tick marks do not intermix:

```
PROC SGRENDER DATA=final TEMPLATE=__pluto;
  BY paramn nicemin nicemax nice_tick_increment;
RUN;
```

Then, we will slightly adjust our TEMPLATE procedure for the extra special DYNAMIC variable _BYVAL4_.

```
DYNAMIC _BYVAL_ _BYVAL2_ _BYVAL3_ _BYVAL4_;
```

Lastly, we adjust the references to these variables in the TICKVALUESEQUENCE= sequence options.

```
YAXISOPTS=(<other yaxisopts options>
                   LINEAROPTS=(TICKVALUESEQUENCE=(START=_BYVAL2_
                                                  END=_BYVAL3_
                                                  INCREMENT=_BYVAL4_)
                               VIEWMIN=_BYVAL2_ VIEWMAX=_BYVAL3_))
```

Figure 4 below shows examples of several y-axes for different lab parameters using Heckbert's algorithm in combination with using special DYNAMIC variables.

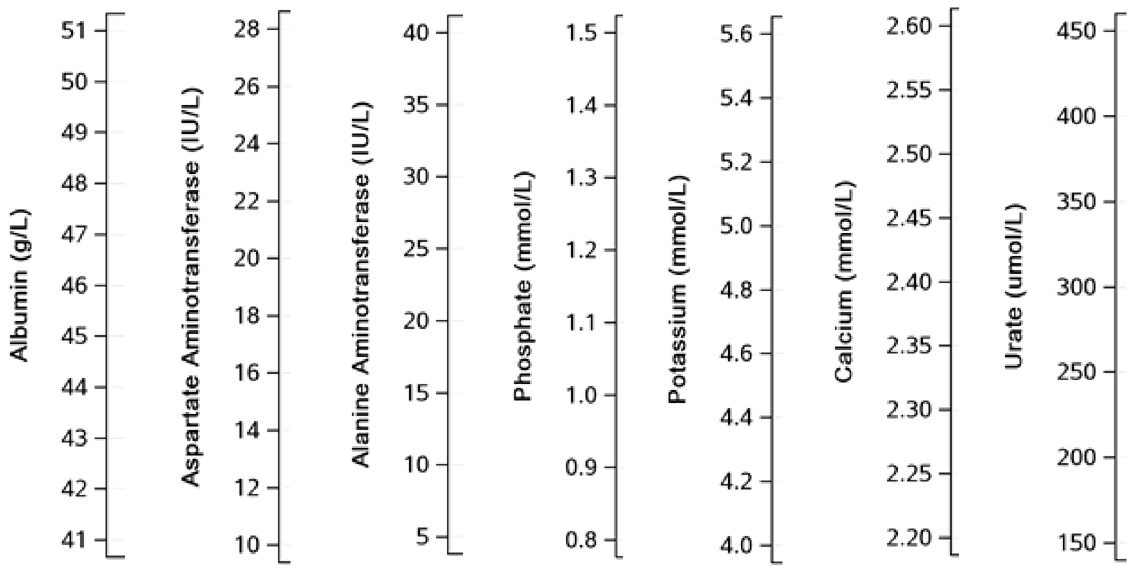

Figure 4. Y-axis Tick Values for Several Lab Parameters

There are a few things to note here. Despite the desired number of tick marks being 9, we got as few as 7 for Urate, and as many as 11 for Albumin. We can also see how all of the increments are 1, 2, or 5 in a power of 10. For example, the Calcium increment is 0.05, Phosphate increment is 0.2, and Urate is 50.

BEYOND HECKBERT

Here, we offer up some additional situations where it is useful to use data-driven options via special DYNAMIC variables.

TREATMENT COLORS

Not until SAS 9.4 M3, are we able to use discrete attribute map variables with LINECOLORGROUP=, LINEPATTERNGROUP=, MARKERCOLORGROUP=, and MARKERSYMBOLGROUP= options. When doing a spaghetti plot for pharmacokinetic (PK) curves, we usually have to make use of those options because GROUP=SUBJID will use a different GRAPHDATA style element for each subject. But, we want the line color to be based on treatment as shown in Figure 5 below. Suppose we have one page per treatment, and we want GREEN to be associated with a particular treatment.

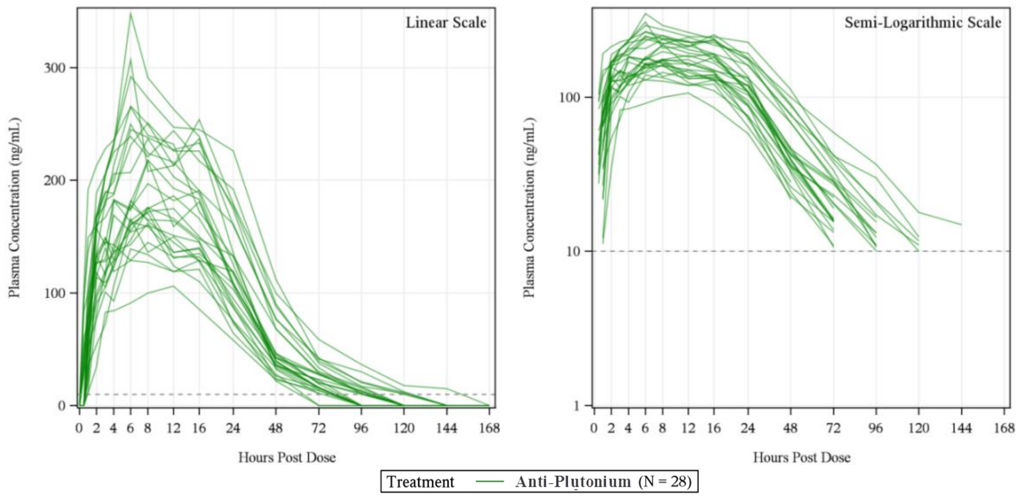

Figure 5. Spaghetti Plot with Color Based on Treatment

With the _BYVAL*n*_ family of special DYNAMIC variables, we can avoid using attribute maps and LINECOLORGROUP= altogether. First, we can assign the colors we want in a prior DATA step:

```
     if TRTAN=1 then color='BLUE';
else if TRTAN=2 then color='GREEN';
```

Below is the SERIESPLOT statement we might find within our TEMPLATE procedure along with the SGRENDER procedure. Note the use of _BYVAL3_ to assign color.

```
PROC TEMPLATE;
   ...
   SERIESPLOT  X=timepoint Y=aval
    / Group=subjid lineattrs=(thickness=1px color=_BYVAL3_ pattern=solid);
   ...
RUN;

PROC SGRENDER DATA=final TEMPLATE=pkconc;
    BY param trtan color;
RUN;
```

PAGINATION

Often, we display Page XX of YY pagination as subtitles in outputs. In an earlier DATA step, we could derive a variable, PG, that keeps track of the graph pages. For example, if we have one graph per lab parameter, we can increment PG when encountering the next parameter in a DATA step. Then, include it in the BY statement of the SGRENDER procedure:

```
PROC SGRENDER DATA=final TEMPLATE=__pluto;
  BY pg nicemin nicemax nice_tick_increment;
RUN;
```

This would allow us to use _BYVAL_ in the ENTRYTITLE statement of the TEMPLATE procedure:

```
ENTRYTITLE "Page " _BYVAL_ " of &maxpg."/halignCenter=Graph;
```

The macro variable MAXPG could be derived using CALL SYMPUTX routine at the end of a DATA step that is deriving the paging variable, PG:

```
CALL SYMPUTX('maxpg',pg);
```

AXIS LABELS

Notice in Figure 4, the y-axis has the value of PARAM for each figure. Passing PARAM as a BY variable in the SGRENDER procedure allows for data-driven axis labels:

```
PROC SGRENDER DATA=final TEMPLATE=__pluto;
  BY pg param nicemin nicemax nice_tick_increment;
RUN;
```

Below is the code snippet of the TEMPLATE procedure:

```
YAXISOPTS=(LABEL=_BYVAL2_ <other yaxisopts options>)
```

CONCLUSION

Hopefully, we've displayed the power of DYNAMIC variables, especially special DYNAMIC variables. These allow us to start creating data-driven solutions in our graph templates. The examples we covered were tick mark implementation, color assignments, pagination, and axis label assignment. In an environment that is becoming more standardized via CDISC and their SDTM and ADaM standards, along with the desire to streamline standard outputs, the techniques covered in this paper allow for a step in that direction. Another added benefit is providing a basis to QC or validate elements of the graph in an automated fashion. It will be interesting to see what other situations arise where we can apply the method of using special DYNAMIC variables to implement data-driven features. To the future!

REFERENCES

P. Heckbert. Nice numbers for graph labels. In A. Glassner, editor, Graphics Gems, pages 61–63. Academic Press, Boston, 1990.

Matange, Sanjay. 2013. *Getting Started with the Graph Template Language in SAS®: Examples, Tips, and Techniques for Creating Custom Graphs.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2016. *SAS® 9.4 Graph Template Language: User's Guide, Fifth Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2016. *SAS® 9.4 Graph Template Language: Reference, Fifth Edition*. Cary, NC: SAS Institute Inc.

AKNOWLEDGMENTS

There are many individuals to thank that made this paper possible; let's bring them out of the shadows.

A special thank you goes to Phillip Spark. First, thank you for raising the bar and showing there is another level to being a statistical programmer. Your knowledge and skillS are something I've strived for throughout my career. Secondly, thank you for making the connection to use special DYNAMIC variables to implement data-driven tick marks in GTL. You cracked the code!

A special thank you to Paul Heckbert for letting me contact him on LinkedIn and pick his brain about something he developed almost 30 years ago. I am humbled by your ingenuity.

Thank you to Scott Kosten for reviewing the paper, providing valuable feedback, and for your support and mentoring over the years. Great Scott!

Thank you to Mary Geurink for helping make this paper pro-grammatically correct.

Last but certainly not least, thank you to my employer, Cytel, and my manager, Geoff Welch, for always being supportive and finding opportunities for my growth and development as a statistical programmer.

CONTACT INFORMATION

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

Name: Chris Smith
Enterprise: Cytel Inc.
E-mail: chris.smith@cytel.com
Web: www.linkedin.com/in/smithchr