Paper 11362-2016

Generating Color Scales in SAS®: 256 Shades of RGB

Jeff GRANT, Bank of Montreal; Mahmoud MAMLOUK BMO Harris Bank

ABSTRACT

Color is an important aspect of data visualization and provides an analyst with another tool for identifying data trends. But is the default option the best for every case? Default color scales may be familiar to us; however, they can have inherent flaws that skew our perception of the data. The impact of data visualizations can be improved with a little thought toward choosing the correct colors.

A simple technique is presented, detailing how you can use SAS® to generate a number of color scales and apply them to your data. Other techniques can be used to apply pre-defined color maps on your data in order to emphasize singularities or volumes.

Using just a few graphics procedures, you can transform almost any complex data into an easily digestible set of visuals. The techniques used in this discussion were developed and tested using SAS® Enterprise Guide® 5.1 under SAS 9.3.

INTRODUCTION

Most SAS programmers are introduced to plot color customization when first creating simple scatter, bar or series plots. Usually, this information is an afterthought and is relegated to a single line: “to change the color of your line, use the COLOR= option”. Does this sound familiar?

What if you want to do something a little more interesting than line colors? This paper will describe a method to apply complex color schemes to your data.

COLORS IN SAS

There are many different color spaces available in SAS, which include the RGB (Red Green Blue) space, CMY (Cyan Magenta Yellow) space, gray-scale, SAS Registry Colors and the CNS Color Names. The RGB color space is the focus of this paper.

The most frequently used schemes are the SAS Registry and the CNS schemes. These are invoked in a plot command by specifying the actual color name or variant, such as ‘green’ or ‘very light green’. This color scheme can be quite flexible as other examples include ‘pale violet red’, ‘dark grayish green’, and ‘very light purplish blue’. However, these examples are distinct color values and this sort of definition can be quite cumbersome when attempting to create continuous color gradients.

Consider the RGB color space. It is an additive color model, meaning that any color can be defined by the combination of red, green, and blue colors. Continuous color gradients can be developed by varying the contributions of each of the three colors by a given amount. This makes it simple to apply gradients to our data. The first step is defining the color scale.

WHAT IS A COLOR SCALE?

A color scale is the mapping between a continuous gradient of color and a desired set of data. The result is a 3-by-*m* matrix specifying values for RGB components – the intensities of the red, green, and blue components. These RGB triplets contain information in three color channels with values between 0 and 255 that are then scaled to the data. A typical color scale will have 256 RGB values, but the length or step size between shades in the color scale can be varied with *m*.

An example of a color scale is the JET color scale, a rainbow color scale popular in data visualizations. Figure 1 plots the individual contributions of the red, green, and blue components (the higher the number, the higher the contribution) on a scale from 0 to 255. The resulting color gradient is also shown.

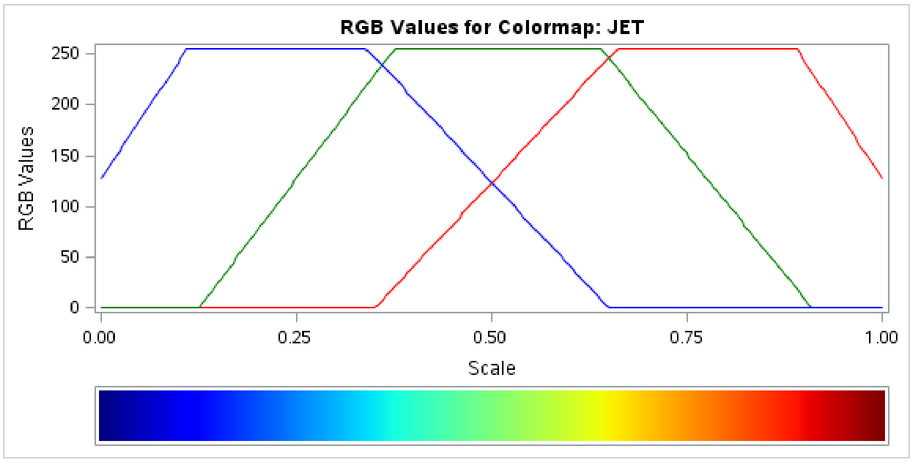

Figure 1: RGB components of a popular rainbow color map

If you printed this paper on a black and white printer, you may already see one of the problems with the rainbow color scale. The brightness of the colors goes from dark at one extreme to light in the middle back to dark at the other extreme, making it difficult to quickly identify the extreme values especially in gray scale. Another problem with the rainbow color map is that it uses all the colors of the rainbow, which can be troublesome for people with colorblindness. This, along with the non-linearity in brightness, makes it difficult to determine the relative values of the data.

It is important to consider these issues when developing your data visualizations. Some examples of color scales that are linear in brightness and mitigate some colorblindness problems are shown in Figure 2 and Figure 3.

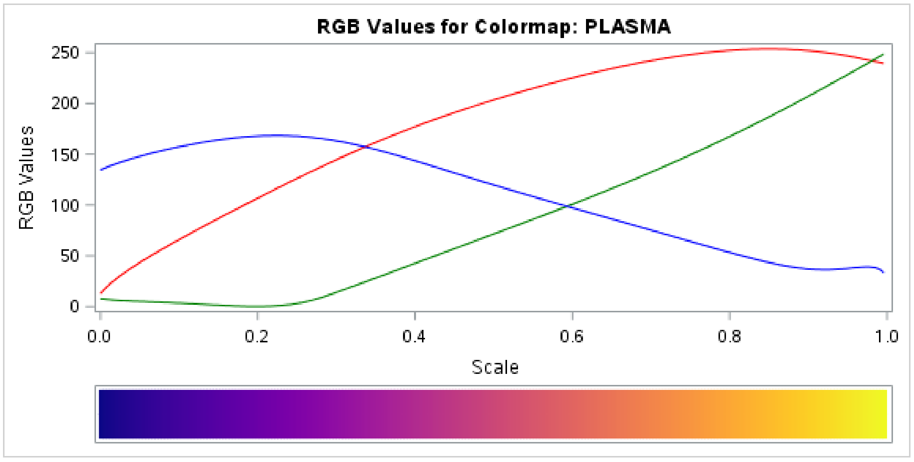

Figure 2: RGB values for PLASMA, an alternate color map with linear brightness and a simple color scheme.

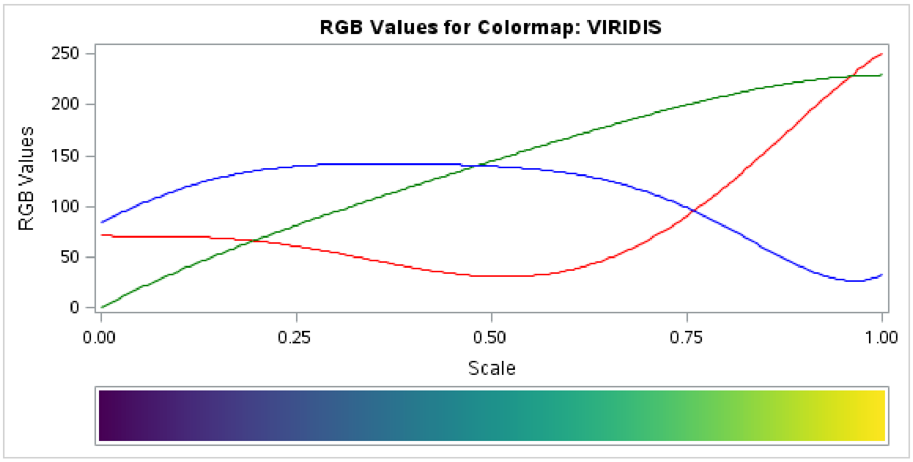

Figure 3: RGB values for VIRIDIS, an alternate color map with linear brightness and a simple color scheme.

DEFINING COLOR SCALES

There are many ways to define a color scale – all that is needed is a set of RGB triplets. The individual R, G, and B components can be defined by any combination of linear or non-linear functions. For example, the rainbow color scale in Figure 1 defines its components as combinations of flat and linear sections, while the scales in Figure 2 and Figure 3 have their components defined by polynomial functions. It is also possible to search out color scales and import them into SAS. This paper presents a method for generating a color scale for use in SAS using either an imported table or a set of functions.

The method is as follows:

1) Generate a table, either by import or calculation, containing 256 observations of R, G, and B values. The values should range between 0 and 255 for convenience.
2) Apply an index of 1 to 256, 1 being the first color and 256 being the last, to allow for easy scaling to your data.
3) Convert the RGB triplets to hexadecimal for use in SAS.
4) Scale your data to an index of 1 to 256, and join with the hexadecimal values created in Step 3.
5) Plot your data using the hexadecimal values as the color response.

Step 1: Generate the Color Scale Table

The import tables used in this paper had RGB values scaled between 0 and 1. For convenience when converting to hexadecimal, these values are multiplied by 255 to get the desired range. This is done in a simple DATA step, using an array to define the components:

```
data color_scale;
   array rgb[3];
   set import_scale;
   color_index = _n_;
   rgb[1] = R * 255;
   rgb[2] = G * 255;
   rgb[3] = B * 255;
run;
```

To define the components using functions, replace the values of the RGB array with the function definitions and loop over the desired color scale length:

```
data color_scale;
   array rgb[3];
   n_steps = 256;
```

```
      do i = 0 to n_steps-1;
         c = i / (n_steps-1);
         color_index = i + 1;
         rgb[1] = 255*(-11.5444*(c**5)+24.7079*(c**4)-
                   14.4479*(c**3)+2.122875*(c**2)-0.13446*c+0.279996078);
         rgb[2] = 255*(-1.21921*(c**4)+2.267498*(c**3)-
                   1.73326*(c**2)+1.582772*c+0.001702159);
         rgb[3] = 255*(26.2720929*(c**6)-65.3209*(c**5)+56.65624*(c**4)-
                   19.2918*(c**3)+0.091963*(c**2)+1.386773*c+0.332663775);
         output;
      end;
      drop i;
   run;
```

The functions in the example above were generated based on a scale of 0 to 1, so it is necessary to calculate the RGB values using that same scale as exhibited in the variable C. These functions generate the color scale VIRIDIS, shown in Figure 3.

Step 2: Assign a color index

In the examples in Step 1, the color index is assigned using the color index variable. This variable has values that range from 1 to 256 and is defined according to the color scale generation method.

Step 3: Convert the RGB triplets to hexadecimal

SAS accepts hexadecimal input for colors making it easy to assign colors while avoiding color allocations of 'light grayish reddish brown'. The conversion from RGB triplet to hexadecimal requires the concatenation of the converted RGB array into an 8-character string, prefixed with 'CX':

```
    c_rgb = 'CX------';
    digits='0123456789ABCDEF';
    do rgb_index = 0 to 2;
       substr(c_rgb,3+2*rgb_index,1) =
                         substr(digits,rgb[rgb_index+1]/16 +1,1);
       substr(c_rgb,4+2*rgb_index,1) =
                         substr(digits,mod(rgb[rgb_index+1],16)+1,1);
    end;
```

After the conversion, the color scale is defined by two columns: COLOR_INDEX and C_RGB.

Step 4: Scale your data and join the color scale

To apply the color scale to your data, it must be scaled and indexed from 1 to 256 to match the color index on the color scale. In this paper, we present method using PROC SQL and the DATA step:

```
   proc sql noprint;
      select max(value), min(value) into :_max, :_min from test_data;
   quit;
   data data_w_color_scale;
      if 0 then set color_scale(keep=color_index c_rgb);
      set test_data;
      color_index = 1+round(255*(value-&_min)/(&_max-&_min));
      if _n_ eq 1 then do;
            declare hash ci(dataset:"&colormap");
            ci.definekey('color_index');
            ci.definedata('c_rgb');
            ci.definedone();
      end;
      rc=ci.find();
      drop rc;
   run;
```

The PROC SQL defines the data extremes of the desired variable (in this case, VALUE) that are assigned values of 1 and 256. The COLOR_INDEX variable in the DATA step completes the scaling and is used as the join condition for the color scale table. The result is a dataset with an additional column specifying the color for each observation.

Step 5: Plot your data using the defined color scale

To generate surface plots, the TEMPLATE procedure is used because it provides a large amount of flexibility for defining plot characteristics. The HEATMAPPARM option is used for 2D surface plots:

```
  heatmapparm x=_x1 y=_x2 colorresponse=altcolorscale;
```

The X and Y variables correspond to the two dimensions in your data. Typically, the COLORRESPONSE will be defined by the dependent data (in this case the variable VALUE). However, the desired response is to use the colors defined by the color scale, C_RGB. To do so, a range attribute map between the scaled data (COLOR_INDEX) and the color scale is required in the PROC TEMPLATE definition:

```
   rangeattrmap name='altrgbscale';
      %generate_range_statements
   endrangeattrmap;
   rangeattrvar attrvar=altcolorscale var=color_index attrmap='altrgbscale';
```

The range attribute map defines the map between the range attribute var (the scaled response variable, COLOR_INDEX) and the specific characteristics for that range. In this case, we want to specify the color for each range. The macro %generate_range_statements defines the ranges and avoids writing 256 distinct range statements:

```
   %macro generate_range_statements;
      %local n start end;
      %do n = 1 %to &n_colors;
         %let start = %eval(&n);
         %let end = %eval(&n+1);
         range &start -< &end / rangecolor=&&rgbval&n;
      %end;
   %mend generate_range_statements;
```

where the &rgbval macro variable is defined as the hexadecimal color variable for that color index:

```
  data _null_;
     set color_scale end=eof;
     call symputx(cats('rgbval',_n_),c_rgb,'g');
     if eof then call symputx('n_colors',_n_,'g');
  run;
```

The result is a template for a 2D surface plot, where the colors are defined by the custom color scale generated in Steps 1 to 4. The full TEMPLATE procedure is as follows:

```
  proc template;
     define statgraph color_my_data;
     begingraph;
     rangeattrmap name='altrgbscale';
        %generate_range_statements
     endrangeattrmap;
     rangeattrvar attrvar=altcolormap var=color_index attrmap='altrgbscale';
     layout overlay;
        heatmapparm x=_x1 y=_x2 colorresponse=altcolormap;
     endlayout;
```

```
      endgraph;
      end;
   run;
```

The template COLOR_MY_DATA can be referenced in a SGRENDER procedure to produce the final visualization:

```
   proc sgrender data=data_w_color_scale template=color_my_data;
   run;
```

EXAMPLE USING PROC SGRENDER

An example dataset can be generated using the following function:

$$f(x_1, x_2) = \sqrt{|x_1|^5 + {x_2}^2} \qquad \forall \qquad -5 \leq x_1 \leq 5, -5 \leq x_2 \leq 5$$

Evaluating this function over the prescribed window produces what is referred to as the ‘Hat Function’, shown in Figure 4.

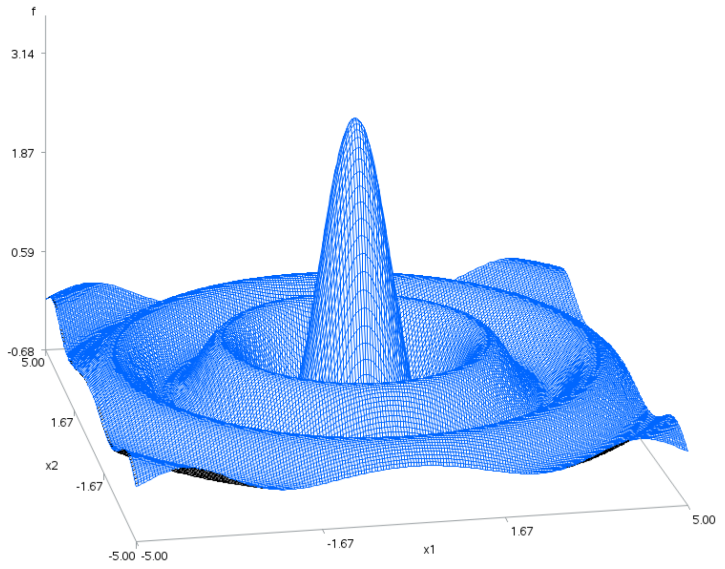

Figure 4: Example data set, referred to as the Hat Function

Using the PLASMA color scale from Figure 2, a 2D surface plot can be generated using the template COLOR_MY_DATA and is shown in Figure 5.

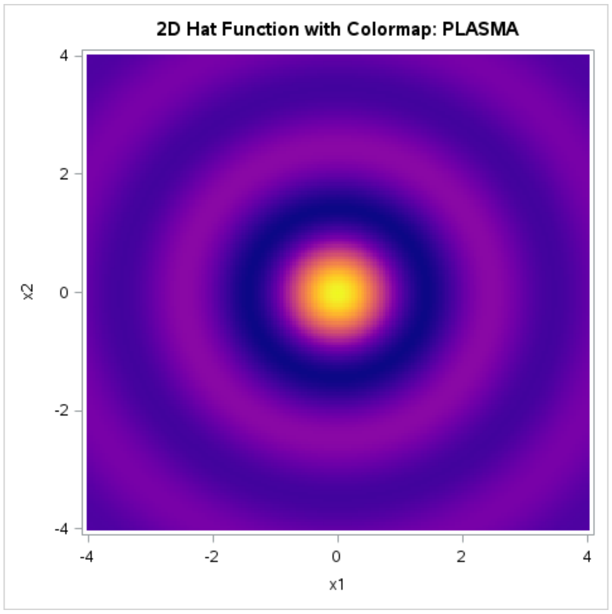

Figure 5: A 2D surface plot of the Hat Function, colored with PLASMA

Important to note here is the distinction between the two extremes – the light colors show the center point clearly with the alternating light dark sections show the relative depths of the surrounding waves. The gradient is continuous, showing the true trends in the data – the center spike is smooth with no sharp transitions.

Using the popular rainbow color map, Figure 6 is generated.

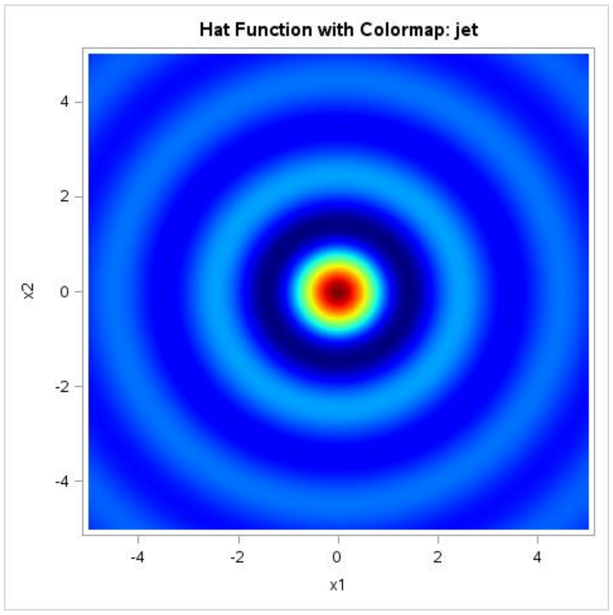

Figure 6: A 2D surface plot of the Hat Function, colored with a rainbow color scale

When viewed in color, the two extremes are easy to spot – but if you printed this in gray scale you might have trouble determining if the center is the low or high extreme. The center spike appears to have sharp transitions due to the distinct changes in color – this type of color scale can highlight false trends in the data.

SAS default color scales can be used, for example, Figure 7 uses the SAS style HTMLBLUE.

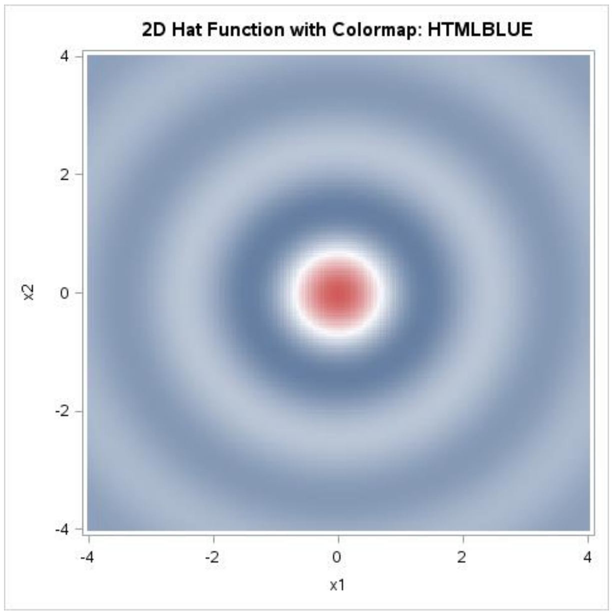

Figure 7: A 2D surface plot of the Hat Function, colored with SAS Style HTMLBLUE

This color scale is a diverging color scale, which is typically used when considering diverging data. A diverging dataset typically has a transition region in the middle that is special, for example, fluid temperature with the freezing point in the middle and positive and negative numbers on either side. The Hat function is not a diverging function, so this color scale is not necessarily appropriate for this data. The center circle of white indicates a region of significance, however, the actual data does not reflect this.

It is important to consider the type of data and the features you need to highlight when choosing a color scale. The correct color scale can significantly improve the visualization, while an inappropriate color scale can cloud understanding of the true trends in the data.

EXAMPLE USING PROC G3D

PROC G3D has a built-in parameter that can map each data point in a scatter plot to a specific color. The key is to scale the dataset to plot and directly assign a color between 1 and 256 according to the color scale created.

3D scatter plots can be used to inform the shape of closed volumes in SAS that contour and surface plots cannot. Layering on colors in a smart way on 3D scatter plots can give more information on how the data is positioned with respect to the view of the data. For example, it can highlight If some points are more in the background than others or if other points are in between other volumes.

In order to illustrate this idea, we generate a dataset that represents a Klein bottle. A Klein bottle is a 3D volume that represents two Mobius strips glued to each other. It’s supposed to look like one tube going around and intersecting with itself. In the 9.3 version of SAS, one of the ways to plot this shape is to use a scatter plot in PROC G3D.

The default colors in the SAS color scheme HTMLBLUE produces the output in Figure 8.

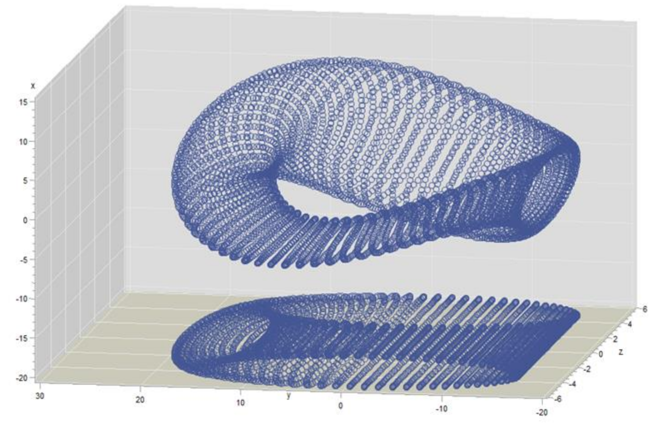
Figure 8: Klein Bottle - Default color scale

As can be seen in Figure 8, the default color scale, although it highlights the complexity of the volume, it doesn't allow us to distinguish what exactly is in the foreground or in the background.

A projection on the horizontal axis shows that there is an inner part of the volume that is intersecting with itself. This is where a color scale with some variations can come in handy.

As mentioned above, PROC G3D has the COLOR key parameter for color mapping. Example,

```
proc g3d data= data_w_color_scale;
scatter x*y=z / color=c_rgb;
run;
```

C_RGB is created in Step 4 of the first section where the scaling of the data is occurring. Applying the popular rainbow color map to the scaled Klein bottle data produces Figure 9.

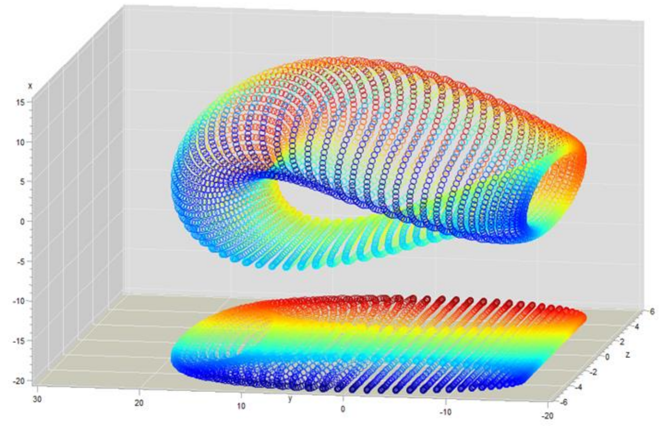
Figure 9: Klein Bottle - Jet color scale

This allows us to understand the dynamics of the volume. Notice that the background is now in Red, the middle part has shades of yellow and light blue and foreground is blue. We can now clearly distinguish that the middle part of the bottle is intersecting itself where the tube goes inside its opening to close the loop.

This can be important when representing data that has different layers or planes that do not necessarily transition in a clear manner from one to the other. We can highlight singularities, sharp transitions or distinguish the shape of the intersections between two planes.

Using the Plasma color scale (Figure 10) also highlights the same features and gives less emphasis on the sharp transitions and presents in a clear way the overall volume with two dominating colors (purple and orange),

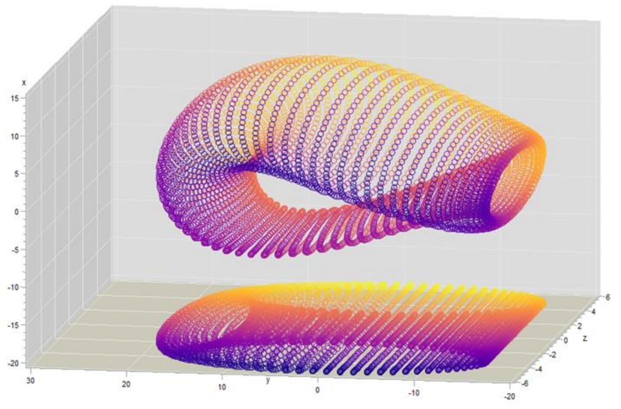

Figure 10: Klein Bottle - Plasma color scale

CONCLUSION

Colors are an important part of data visualization and can highlight the variability of the data in the space considered and that might not be detected with the default settings. The appropriate color scale can really help highlight the important features of your data whereas the wrong color scale can hinder the understanding.

Using PROC TEMPLATE can add significant customization to your plots, where ranges or discrete values can be mapped to specific colors. Using RANGEATTRMAP let you apply user-defined color scales to your data. These templates can then be called in PROC SGRENDER and ODS options provide options for final output and appearance.

On the other hand PROC G3D has a built -in parameter (color =) when used with Scatter plot that accommodates a different color for each data point without the need to define a template.

This paper does not necessarily specify how to obtain the color scale itself rather how to apply a given color scale to your data. Creating your own color scale depends on the data in hand and there is a wide range of color scale out there that can be directly used or customized to your needs.

REFERENCES

Smith, Nathaniel. 2015. “A Better Default Colormap.for Matplotlib” *SciPy 2015*, UCLA. Available at https://bids.github.io/colormap/.

Warren F. Kuhfeld, SAS Institute Inc, Cary NC “*The Graph Template Language and the Statistical Graphics Procedures: An Example-Driven Introduction”*, SAS Global Forum 2010, 334-2010.

Steve Eddins, MathWorks, “*Rainbow Color Map Critiques”* http://www.mathworks.com/tagteam/81137_92238v00_RainbowColorMap_57312.pdf

Paul Bourke, Klein Bottle parametrization (http://paulbourke.net/geometry/klein/)

RECOMMENDED READING

- Wow! You Did That Map with SAS/GRAPH®?

(http://support.sas.com/resources/papers/proceedings09/215-2009.pdf)

- *http://matplotlib.org/users/colormaps.html*

CONTACT INFORMATION

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

Jeff GRANT
Bank of Montreal
Jeff1.Grant@bmo.com

Mahmoud MAMLOUK
BMO Harris Bank
Mahmoud.Mamlouk@bmo.com

APPENDIX: RELEVANT CODE

```
proc fcmp outlib=work.funcs.hexconvert;
    function gethex(rgb1,rgb2,rgb3) $;
          array rgb[3] rgb1-rgb3;
          c_rgb = 'CX------';
          digits='0123456789ABCDEF';
          do rgb_index = 0 to 2;
                substr(c_rgb,3+2*rgb_index,1) =
substr(digits,rgb[rgb_index+1]/16 +1,1);
                substr(c_rgb,4+2*rgb_index,1) =
substr(digits,mod(rgb[rgb_index+1],16)+1,1);
          end;
          return (c_rgb);
    endsub;
quit;
options cmplib=work.funcs;



%macro generate_colormap(map_out=colormap /*define the name of the output
dataset*/
                                                     ,import_map= /*name of
the imported dataset, sorted*/
                                                     ,f_R=
                                                     ,f_G=
                                                     ,f_B=
                                                     ,msg=YES
                                                     );

    /*---Ensure macro runs in open code---*/

    %if &sysprocname ne %then %do;
          %put WARNING: [&sysmacroname] must run in open code.  Terminating
macro execution. ;
          %goto exit;
    %end;

    /*---Housekeeping: Macro Options---*/
    %let map_out = %upcase(&map_out);
    %let import_map= %upcase(&import_map);
    %let msg = %upcase(&msg);

    /*---Housekeeping: System Options---*/
    %local _opts _star _tid;
    %let _opts = %sysfunc(getoption(mprint)) %sysfunc(getoption(notes))
%sysfunc(getoption(symbolgen)) %sysfunc(getoption(mlogic));
    %if &msg eq NO %then %let _star = *;
    options mprint nonotes nosymbolgen nomlogic;

    /*---Begin execution---*/
    %&_star.put;
    %&_star.put [&sysmacroname] -> Start.;

    /*---Check for Parameter Errors---*/

    /*---Determine TYPE of defintion---*/
    %local cm_type;
    %if &import_map eq  and (%length(&f_R) eq 0 or %length(&f_G) eq 0 or
%length(&f_B) eq 0) %then %do;
          %put [&sysmacroname] -> No import or not all functions provided.
Default colormap [VIRIDIS] will be generated.;
          %let cm_type = DFLT;
    %end;
    %else %if &import_map ne %then %do;
          %put [&sysmacroname] -> Import map [&import_map] chosen to define
colormap.;
          %let cm_type = IMPT;
    %end;
    %else %if %length(&f_R) gt 0 and %length(&f_G) gt 0 and %length(&f_B)
gt 0 %then %do;
          %put [&sysmacroname] -> Functions have been provided that will
define the colormap.;
          %let cm_type = FUNC;
    %end;
    %&_star.put [&sysmacroname] -> Application type is [&cm_type].;

    /*---Begin Core Processing---*/
    data &map_out;
          array rgb[3];
          array tmp[3] _temporary_;
          length c_rgb $8.;

          %if &cm_type eq IMPT %then %do;
                set &import_map;
                c = (_n_ - 1)/255;
                color_index = _n_;
                rgb[1] = R * 255;
                rgb[2] = G * 255;
                rgb[3] = B * 255;
                c_rgb = gethex(rgb[1],rgb[2],rgb[3]);
          %end;

          %if &cm_type eq DFLT %then %do; /*Viridis colormap*/
                steps = 256;
                do gcm_ii = 0 to steps-1;
                      /*scale goes from 0 to 1*/
                      c = gcm_ii/(steps-1);
                      color_index = gcm_ii + 1;
                      rgb[1] = 255*(-11.5444*(c**5)+24.7079*(c**4)-
14.4479*(c**3)+2.122875*(c**2)-0.13446*c+0.279996078);
                      rgb[2] = 255*(-1.21921*(c**4)+2.267498*(c**3)-
1.73326*(c**2)+1.582772*c+0.001702159);
                      rgb[3] = 255*(26.2720929*(c**6)-
65.3209*(c**5)+56.65624*(c**4)-
19.2918*(c**3)+0.091963*(c**2)+1.386773*c+0.332663775);
                      c_rgb = gethex(rgb[1],rgb[2],rgb[3]);
                      output;
                end;
                drop gcm_ii;
          %end;

          %if &cm_type eq FUNC %then %do;
                steps = 256;
                do gcm_ii = 0 to steps-1;
                     c = gcm_ii/(steps-1);
                     color_index = gcm_ii + 1;
                     rgb[1] = &f_R;
                     rgb[2] = &f_G;
                     rgb[3] = &f_B;
                     c_rgb = gethex(rgb[1],rgb[2],rgb[3]);
                     output;
                end;
           %end;
     run;

     /*---Generate table for colorbar based on the colormap---*/
     proc sort data=&map_out;
           by color_index;
     run;
     data _null_;
           set &map_out end=eof;
           call symputx(cats('rgbval',_n_),c_rgb,'g');
           if eof then call symputx('n_colors',_n_,'g');
     run;
     data work.&map_out._bar;
           do until(eof);
                set &map_out end=eof;
                do x2 = 0 to 1;
                     output;
                end;
           end;
     run;

     /*---Clean up and revert original options---*/
     %finish:
           %&_star.put [&sysmacroname] -> Finish.;
           %&_star.put;
           options &_opts;

%exit:
%mend generate_colormap;

%macro generate_range_statements;
     %local n start end;
     %do n = 1 %to &n_colors;
           %let start = %eval(&n);
           %let end = %eval(&n+1);
           range &start -< &end / rangecolor=&&rgbval&n;
     %end;
%mend generate_range_statements;

%macro plot_colormap(colormap=);

     /*---Define Template for the RGB and Colorbar---*/
     proc template;
           define statgraph plot_colormap;
           begingraph;
                entrytitle "RGB Values for Colormap: &colormap";
                rangeattrmap name='rgbmap';
                     %generate_range_statements
                endrangeattrmap;
                rangeattrvar attrvar=colormap var=color_index
attrmap='rgbmap';
                layout lattice / rows=2 rowgutter=5 rowweights=(0.85 0.15)
columndatarange=union;
                      layout overlay / xaxisopts=(label='Scale')
yaxisopts=(label='RGB Values' linearopts=(viewmin=0 viewmax=255));
                            seriesplot x=c y=rgb1 / lineattrs=(color=red)
primary=true;
                            seriesplot x=c y=rgb2 /
lineattrs=(color=green);
                            seriesplot x=c y=rgb3 / lineattrs=(color=blue);
                      endlayout;
                      layout overlay / xaxisopts=(display=none)
yaxisopts=(display=none);
                            heatmapparm x=c y=x2 colorresponse=colormap;
                      endlayout;
                endlayout;
          endgraph;
          end;
    run;

    ods graphics / reset width=6in height=3in
imagename="&colormap._rgbvalues";
    proc sgrender data=&cm._bar template=plot_colormap;
    run;

%mend plot_colormap;

proc template;
    define statgraph color_my_data;
    begingraph;
          dynamic _x1 _x2 _scale _title;
          entrytitle _title;
          rangeattrmap name='altrgbmap';
                %generate_range_statements
          endrangeattrmap;
          rangeattrvar attrvar=altcolormap var=color_index
attrmap='altrgbmap';
/*        layout overlay / xaxisopts=(linearopts=(viewmin=-2 viewmax=2))
yaxisopts=(linearopts=(viewmin=-1 viewmax=1));*/
          layout overlay;
                heatmapparm x=_x1 y=_x2 colorresponse=altcolormap /
name='heatmap';
                continuouslegend 'heatmap';
          endlayout;
    endgraph;
    end;
run;



proc sgrender data=test_data_scaled template=color_my_data;
    dynamic _x1='x1' _x2='x2';
run;

/******For Proc G3D*********/
```

```
%macro scale_colormap_z(colormap=,data=, out= );
    proc sql noprint;
          select max(z), min(z) into :_max, :_min from &data;
    quit;
    data &out;
          if 0 then set &colormap(keep=color_index c_rgb);
          set &data;
          color_index = 1+round(255*(z-&_min)/(&_max-&_min));
          if _n_ eq 1 then do;
                declare hash ci(dataset:"&colormap");
                ci.definekey('color_index');
                ci.definedata('c_rgb');
                ci.definedone();
          end;
          rc=ci.find();
          drop rc;
    run;
%mend scale_colormap_z;

%let step = 70;

data work.klein_data_&step;
do  u = 0 to 6.2 by 6.2/&step;
    r = 4*(1-cos(u)/2);
    do v = 0 to 6.2 by 6.2/&step;
          if 0<=u<3.14 then do;
                x = -(6*cos(u) * (1+sin(u)) + r * cos(u) * cos(v));
                y = 16 * sin(u) + r * sin(u) * cos(v);
          end;
          else do;
                x = -(6*cos(u) * (1+sin(u)) + r * cos(v+3.14));
                y = 16 * sin(u);
          end;
    z = r * sin(v);
    output;
    end;
end;
run;

%let cm = jet_2;
%scale_colormap_z(colormap=&cm,data=klein_data_&step, out=
klein_data_scaled_&step)

proc g3d data=klein_data_scaled_&step;
scatter y*z=x / noneedle color=c_rgb rotate=85;
run;
```