Paper DV07

Combining Trigonometry, Functions, and the Polygon Plot to Create Complex Figures Not Natively Available in SAS Including Sankey, Sunburst, and CIRCOS Charts

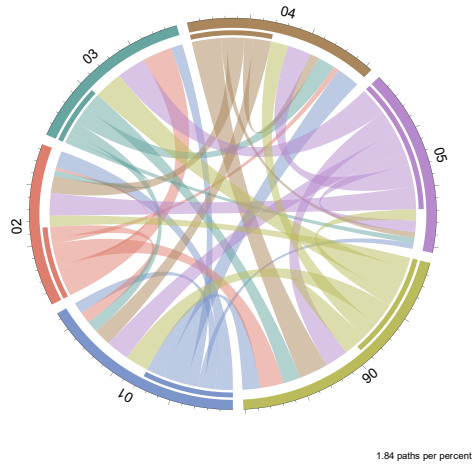
Jeffrey Meyers, Regeneron Pharmaceuticals, New Jersey, USA

ABSTRACT

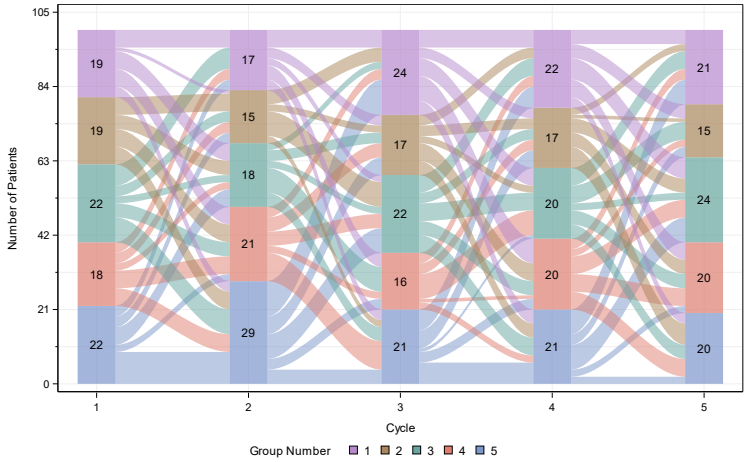
SAS currently has an expansive array of plot types available within the SG procedures and Graph Template Language, but there are still charts that are too complex to create using conventional means. Fortunately, SAS has a powerful trick up its sleeve to allow the ambitious programmer to design these charts: the polygon plot. A polygon plot allows the programmer to use geometric, trigonometric, and other equations to manually produce x and y coordinates that form the perimeter around a shape with the option to fill in the shape with color. This paper will walk through utilizing the polygon plot in combination with mathematical functions to create circular, twisting, and other complex graphs including the Sankey, sunburst and CIRCOS charts.

INTRODUCTION

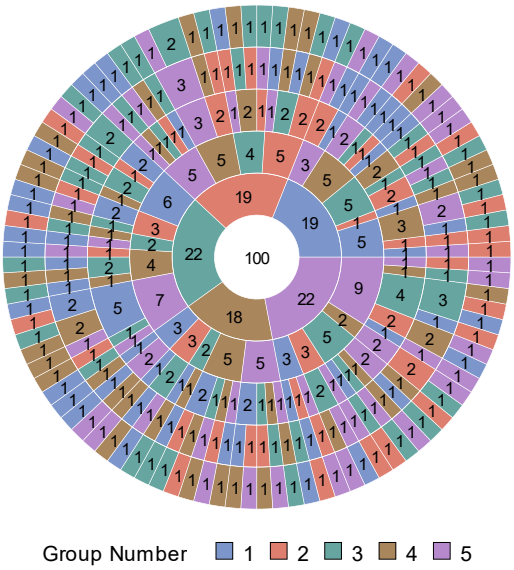
The SG procedures and Graph Template Language (GTL) are an ever-improving utility within the SAS software but is still missing several common graph types. The missing functionality typically revolves around circular plots and rectangles that change width at each end. A graphics challenge from R programmers to create a CIRCOS plot within SAS led to the development of a method to combine trigonometric functions with the POLYGON type plot within SAS graphics to imitate a polar axis using Cartesian coordinates. The first figure created with this method was the CIRCOS plot to answer the graphics challenge:



The CIRCOS plot is a very beautiful and fun way to display data of participants moving from one group to another but is not very quantitative and has not been useful in the author's experience in health science research. The methods however were able to be expanded to another prominent graph type in the pharmaceutical area: the Sankey diagram.



The Sankey diagram shows similar data to a CIRCOS diagram but across multiple timepoints and is better at quantifying the data being shown. Counts are shown within each group and can also be included on the individual paths moving from one group to another. One shortcoming to this figure is it's not typically possible to track a specific group of patients from the starting timepoint to the final timepoint. A third type of figure was presented for this purpose: the sunburst chart.



The sunburst chart starts with a group of patients or records at the center circle. Each successive ring moving from inside to outside represents another unit of time passing (e.g. one week, one month, one cycle). This presents the same information as the Sankey diagram with one key difference. The Same patients can be tracked from the center to the outside by following the rings. For example in the figure above 19 patients enter group number 2 in the first circle. These 19 patients are then split into 5, 5, 4, and 5 in the next ring. They cannot enter another wedge from the first circle. Each successive ring these initial 19 patients can be tracked to see their final path to the outermost circle. The method will be demonstrated within this paper on these three figure types, but can be expanded to other figures such as pie charts, CONSORT diagrams, donut charts and other circular figures.

THE POLYGON PLOT STRATEGY

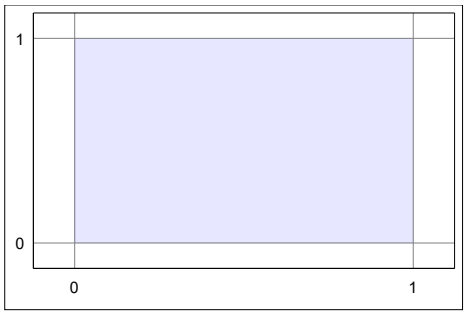
The strategy centers around combining the POLYGON plot and mathematics equations to plot shapes that SAS either does not have statements for currently or to have more control over the widths of a shape that SAS can currently create.

THE POLYGON PLOT

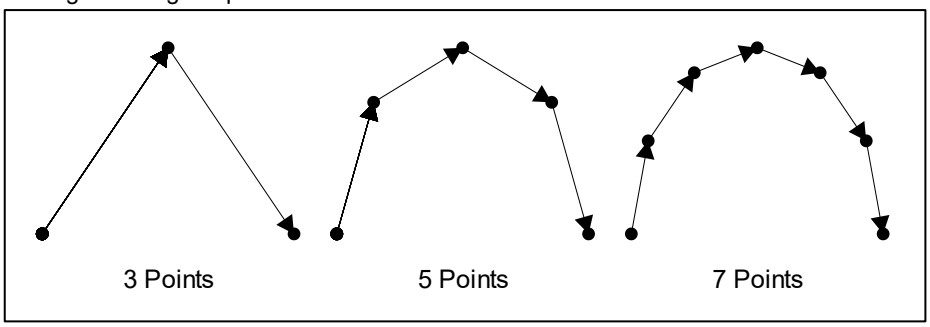
The POLYGON plot is a straightforward tool that only requires three variables: a x-coordinate, a y-coordinate, and an ID variable to distinguish between shapes. Lines will connect from each set of coordinates until the last point which will automatically connect to the first point. The following is an example data set to create a square:

X	Y	ID
0	0	1
1	0	1
1	1	1
0	1	1

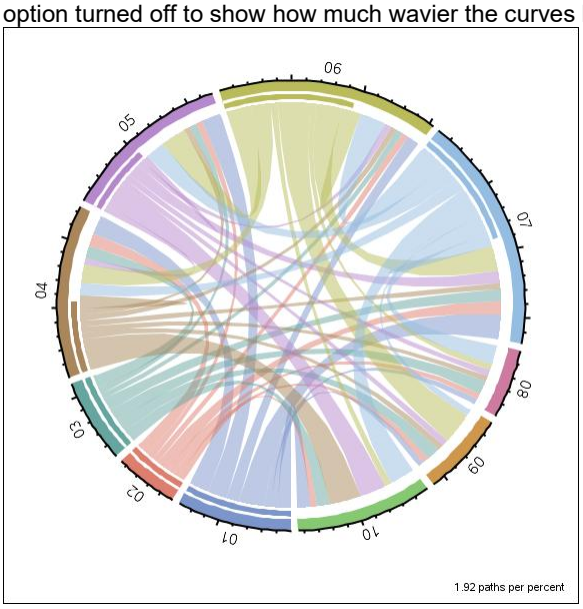
The X and Y variables contain the x and y coordinates, respectively. The ID variable can separate the coordinates into different shapes. The above set of coordinates and ID variables would create a square. The following is the graph made using the POLYGON plot on this dataset:



The POLYGON plot is best suited for making straight line shapes but is also capable of creating curves with enough points. The more points generated the smoother the curve becomes, but this also adds file size and processing time when generating the plot.



The more points that are used to draw the arc the smoother the curve becomes. The SUBPIXEL option was added to assist with this leading to needing less points to create a smooth curve by smoothening out lines and curves drawn with several plot statements including SERIES and POLYGON. The following is an example with the SUBPIXEL option turned off to show how much wavier the curves become:



APPLYING EQUATIONS

Creating the x and y-coordinates per an equation is done within a data step combined with DO loops. The starting point and ending point for the shape needs to be determined first as these will be the start and end of the do loop. The following example creates a pie slice around the origin (x=0 and y=0).

PIE SLICE EXAMPLE

A pie slice has two primary components: an arc and an origin that the arc revolves around. The slice is completed when the start and end of the arc connect back to the origin and the shape is filled in. SAS currently does not have a polar axis to easily make an arc, but trigonometric equations can be used to generate the points around an arc:

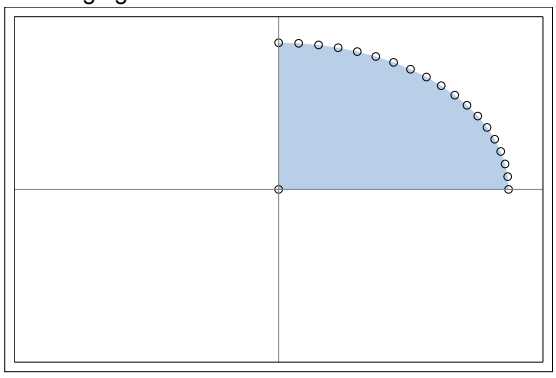
$$x = x_o + d \times cos\ (\theta)$$
$$y = y_o + d \times sin\ (\theta)$$

X_o and Y_o refer to the origin, d refers to the distance from the origin to the x and y-coordinate, and theta is the angle that the arc covers from start to finish. When theta is equal to zero a straight line is drawn, and when theta is equal to 360 degrees (two pi radians) a full circle is drawn. The following example will draw a pie slice that is 90 degrees about the origin (x=0 and y=0) with a distance of one:

```
Data arc;
  ID=1;
  D=1; /**Distance of 1**/
  X_O=0; /**X-coordinate of origin**/
  Y_O=0; /**Y-coordinate of origin**/
  /**Output x/y-coordinates at origin**/
  X=0;Y=0;OUTPUT;
  /**Run DO loop from start to end of arc**/
  DO theta = 0 to 90 by 5;
    X = X_O + D * COS((theta/360)*2*CONSTANT("PI"));
    Y = Y_O + D * SIN((theta/360)*2*CONSTANT("PI"));
    OUTPUT;
  end;
run;


PROC SGPLOT data=arc noautolegend;
    POLYGON X=X Y=Y ID=ID / FILL;
    SCATTER X=X Y=Y;
    REFLINE 0 / AXIS=X;
    REFLINE 0 / AXIS=Y;
    XAXIS DISPLAY=NONE MAX=1.1 MIN=-1.1;
    YAXIS DISPLAY=NONE MAX=1.1 MIN=-1.1;
RUN;
```

The distance of the arc, origin and number of points can all be adjusted in the previous example. The following is the resulting figure:



The arc is drawn with 18 coordinates and the origin. The origin is not added twice to the data set as the last point will automatically connect to the first. Notice in this example that setting up the graph window is also key to be able to properly see the shape.

IN SUMMARY

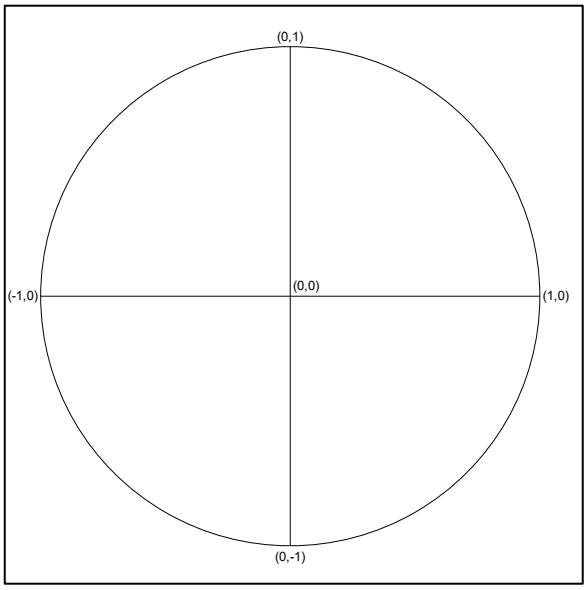
The strategy of finding the right equation to make the necessary shapes, mapping out the coordinates within a data step, generating the shape with POLYGON plot, and properly aligning the graph window can create nearly any graph. The following are specific examples of these methods creating graph types not normally available to SAS.

CREATING A CIRCOS DIAGRAM

A CIRCOS plot is a condensed circular visualization of the relationship between objects, positions and other time-point related data. They are built out of two primary components: the outer circle and the inner connecting curves. There are many optional add-ons depending on the software being used such as having multiple layers of outer circles and adding other distribution graphs such as histograms within the circle. The program highlighted with this paper creates CIRCOS plots with three primary pieces: the outer circle, an inner circle, and the connecting curves.

THE UNIT CIRCLE

The unit circle is a concept from trigonometry that bases a circle of radius 1 around the 0, 0 coordinates of Cartesian (linear) axes. Note that the notation (a, b) in the following figure stands for the position of the points:



Tracing around the circle can be easily accomplished using the basic trigonometry equations sine and cosine with a specified angle. The following equation displays how to calculate the x and y coordinates for each provided angle:

$$x = d \times cos(\theta)$$
$$y = d \times sin(\theta)$$

The coordinates are simply the distance (radius of circle) multiplied by the function of the angle (Θ) where the angle is measured counterclockwise from the x-axis. The unit circle simplifies these equations by making d equal to 1. Arcs can be easily plotted around this circle by creating x and y coordinates from across an angle. A quarter circle arc can be created by plotting points from Θ equal to 0 degrees up to Θ equal to 90 degrees. Note that the SIN and COS functions in SAS use angles measured in radians which can be converted from degrees with the following equation.

$$360^\circ = 2\pi$$

One complete circle is 360 degrees which is equivalent to two pi radians.

OUTER AND INNER CIRCLE

The outer circle uses curved rectangles to visually convey the proportion of the total number of paths entering and leaving each group. The axis is printed on the outer border of the rectangles with each tick mark representing one percentage point of the total sample size. Each group is labeled with the label rotated to be perpendicular to the circle, and each group is colored differently. The inner circle visually represents what proportion of the group is leaving to another group (called path hereafter) versus what proportion of the group is arriving from another group. The more complete the inner circle bar is the more paths that are leaving to other groups. A blank bar means all paths are going to that group. A percentage of the total circle is used to provide a gap between the groups. The length of each group's outer rectangle is equal to that group's proportion of the total number of paths:

$$L_i = \frac{Paths\_out_i + Paths\_in_i}{\sum_1^n (Paths\_out_i + Paths\_in_i)}$$

This proportion is then directly applied to the circumference of the outer circle. If group A has twenty percent of the total paths, then group A will encompass twenty percent of the outer circle. The width of the outer circle rectangles is determined by setting the distance from the center of the unit circle for the outer edge and inner edge of the rectangle. A bar width of five percent would mean the inside arc of the rectangle would be drawn at radius 0.95 and the outer arc of the rectangle would be drawn at radius 1.00.

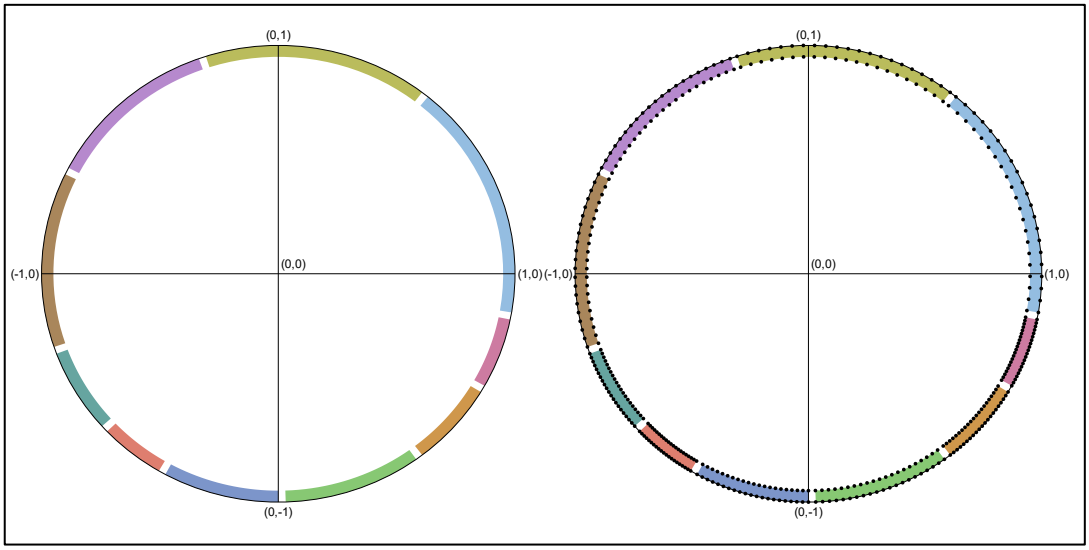
The length of each group's inner rectangle is equal to that group's proportion of paths leaving to total paths:

$$L_i = \frac{Paths\_out_i}{Paths\_out_i + Paths\_in_i}$$

Very similar logic is then used to create the coordinates for the inner rectangles. The only difference is changing the distance that the arcs are drawn at. Additional distance can be easily added between the inner and outer rectangles by setting the radius of the outer edge of the inner rectangle versus the inner edge of the outer rectangle.
The code to compute the coordinates for the rectangles is demonstrated below:

```
 /*Draws the outer part of the rectangle*/
do i=_cum*2 to (_cum+percent)*2 by percent/&points;
   x=cos(constant("pi")*i);
   y=sin(constant("pi")*i);
   output;
end;
/*Draws the inner part of the rectangle.
  Drawn at (1-&barwidth) of the distance from the center*/
do i=(_cum+percent)*2 to _cum *2 by -percent/&points;
   x=(1-&barwidth)*cos(constant("pi")*i);
   y=(1-&barwidth)*sin(constant("pi")*i);
   output;
end;
```

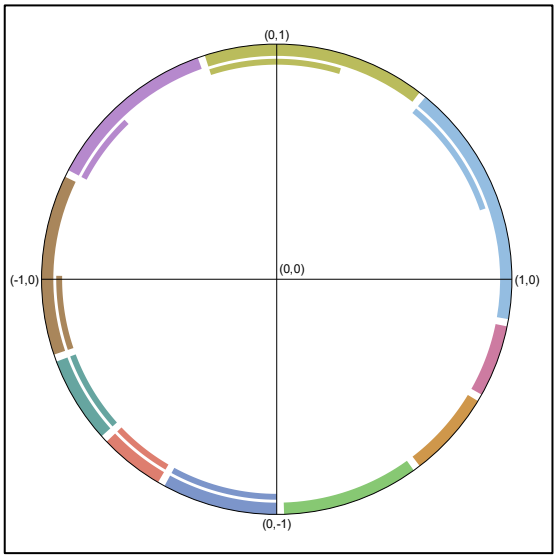
The macro variable BARWIDTH can be modified to set the width of the rectangles. The macro variable POINTS determines how many points are drawn to make the rectangle. The first DO loop draws the outer arc of the rectangle, and the second DO loop draws the same arc in the reverse direction closer to the center of the circle. The loop is multiplied by two to account for a full circle being 2*pi radians. The sides of the rectangles being separated and the change in direction that the coordinates are drawn is due to the nature of the POLYGON plot that is used to make the graph. The direction of the inner part of the rectangle must be opposite the outer or else the rectangle will be folded. The following figure shows the outer circle once as it is displayed and again with all of the plotted points that are entered into the polygon plot statement:



Similar code can be executed for the inner circle with a slight modification for the radius. The following code assumes that the outer circle is drawn at one on the unit circle and that the inner circle is drawn further inside the unit circle. The macro variable INNERGAP is added to add a space between the inner and outer circles:

```
/*Draws the outer part of the rectangle*/
if pct_start>0 then do i=_cum*2 to (_cum+pct_start)*2 by pct_start/&points;
   x=(1-&barwidth-&innergap/2)*cos(constant("pi")*i);
   y=(1-&barwidth-&innergap/2)*sin(constant("pi")*i);
   output;
end;
/*Draws the inner part of the rectangle.
  Drawn at (1-&barwidth) of the distance from the center*/
if pct_start>0 then do
 i=(_cum+pct_start)*2 to _cum*2 by -pct_start/&points;
   x=(1-&barwidth-&innergap/2-&barwidth/2)*cos(constant("pi")*i);
   y=(1-&barwidth-&innergap/2-&barwidth/2)*sin(constant("pi")*i);
   output;
end;
```

This adds the inner lines to the CIRCOS diagram:



CONNECTING CURVES

The curves showing the connections from one group to another are known as Bézier curves. These are curves that revolve around three points: start, end, and a midpoint. The curve is pulled in the direction of the midpoint as it bends from the start to the end. These curves' widths are different at the start and end with both being proportional to the number of paths leaving or entering the group with that curve. The curves that connect one group to another within the circle are known as quadratic Bézier curves which connect three points and are created with the following equation:

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, 0 \leq t \leq 1$$

The definitions of the components are as follows:

- P_0: starting point
- P_1: mid-point
- P_2: end-point
- t: represents the point between the start and end of the curve as a proportion

A benefit of using the unit circle to make this macro is that the mid-point is always the center of the circle which has coordinates of (0, 0). The result of this simplifies the equation to the following:

$$B(t) = (1-t)^2 P_0 + t^2 P_2, 0 \leq t \leq 1$$

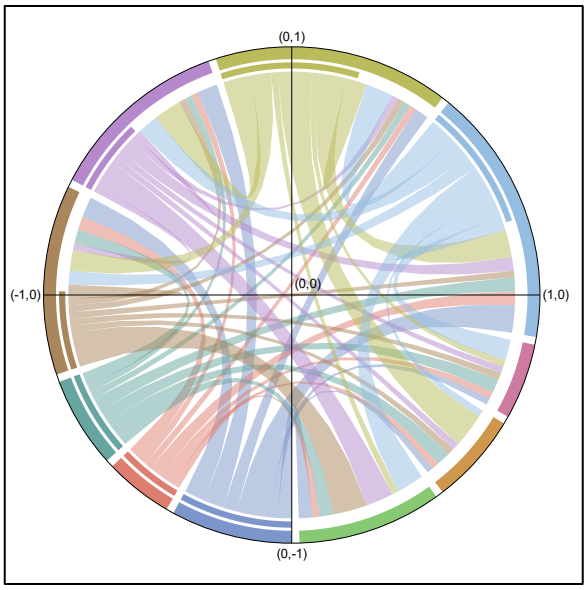
One Bézier curve is calculated for each side of the shape, and these curves are then connected with arcs using the same previous methods. This method gives the connecting curves the look of being wider at the start and ending points than it is in the middle and will pull the curves more towards the center of the circle the further away the starting point is from the ending point.

The DATA step code that calculates the coordinates for the Bézier curves is as follows:

```
/**Draw one side of connecting Bézier curve**/
do i = 0 to 1 by 1/&points;
    x=(1-i)**2*bx1b + 2*(1-i)*i*0 + i**2*bx2a;
    y=(1-i)**2*by1b + 2*(1-i)*i*0 + i**2*by2a;
    output;
end;
```

The variables bx1b and by1b are the x and y coordinates for the start of the curve (P_0), and the variables bx2a and by2a are the x and y coordinates for the end of the curve (P_2). The mid-point (P1) in this case is 0 which is why the second sum is multiplied by 0.

Suppose there are two groups A and B. Group A has 20 patients leaving and group B has 30 patients entering. Specifically, there are 10 patients going from group A to group B. This Bézier curve will be 50% of the width of group A's departing paths and 33.3% of the incoming paths to group B potentially giving different widths at the start and end of the curve. The following figure adds the connecting curves to the previous examples. The different widths of each curve at the start and end is demonstrated:



Calculating the coordinates for all of the Bézier curves requires a number of pre-calculations and organization. Each curve has the following components:

1. Two points at the start of the curve to form the arc around the circle within the beginning group
2. Two points at the end of the curve to form the arc around the circle within the ending group
3. The proportion of paths leaving the beginning group each curve contains
4. The proportion of paths entering the ending group each curve contains
5. The order in which the curves are plotted to avoid overlapping curves from the same group

CIRCULAR AXES

The circular axes are drawn in two pieces: the lines and the tick-marks. The lines are drawn with the same methods as the outside of the OUTER CIRCLE rectangles. The tick-marks are created by plotting one point at the outer edge of the OUTER CIRCLE and another further out on a line perpendicular to the circle. Every fifth tick-mark is twice the size of the others. The DATA step code to create the coordinates is the following:

```
/*Moves along OUTER CIRCLE border*/
do i=(_cum*2+0.02) to (_cum+percent)*2 by 0.02;
   /*Each tick has its own identifier for plotting*/
   tick+1;
   /*x,y coordinates for the base of the tick-mark*/
   xtick=cos(constant("pi")*i);
   ytick=sin(constant("pi")*i);
   output;
   /*Every fifth tick-mark is longer*/
   if mod(tick,5)=0 then do;
      xtick=(1.025)*cos(constant("pi")*i);
      ytick=(1.025)*sin(constant("pi")*i);
   end;
   /*Other tick-marks are shorter*/
   else do;
      xtick=(1.0125)*cos(constant("pi")*i);
      ytick=(1.0125)*sin(constant("pi")*i);
    end;
    output;
end;
```

The axes are created using two SERIESPLOT statements. The first series plot draws the axis line around the OUTER CIRCLE using the same coordinates from the outer circle rectangles. A flag variable, OUTLINE, is added in the data step when the coordinates for the rectangle are generated. A second SERIESPLOT statement draws the tick-marks using the coordinates from the previous code:

```
/**Uses the same coordinates that draw the outer border of the OUTER CIRCLE rectangles
to plot the axis line**/
/*The EVAL function subsets the coordinates using a variable previously calculated in
the data*/
seriesplot x=eval(ifn(outline=1,x,.)) y=y /
  lineattrs=(color=black pattern=1 thickness=2) group=bfr_id;
/**Uses the coordinates for the tick-marks calculated earlier.  Each tick-mark is
given a unique identifier to separate them using the GROUP option**/
seriesplot x=xtick y=ytick /
   lineattrs=(color=black pattern=1 thickness=2) group=tickmark;
```

LABELS

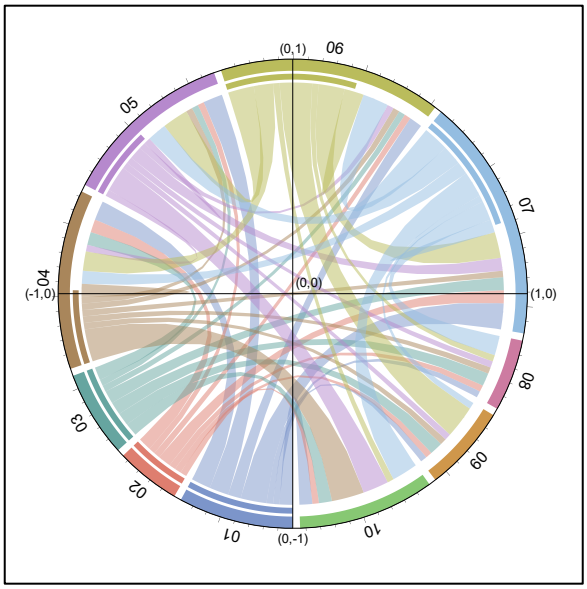
Three variables are required to be calculated to plot the labels: the x-coordinate, the y-coordinate and the rotation required to face the label towards the center of the circle (optional). The x and y coordinates are calculated at the center of each of the OUTER CIRCLE groups, and then the following hierarchical equation is used to calculate the rotation (R):

$$R = \begin{cases} 270, x = 1 \\ 90, x = -1 \\ 0, y = 1 \\ 180, y = -1 \\ -arccos(y) * \frac{360}{2\pi}, x > 0 \\ 90 - arcsin(y) * \frac{360}{2\pi}, x < 0 \end{cases}$$

The labels are created using the TEXTPLOT statement. This statement requires an x-coordinate, a y-coordinate, and a variable containing the label. A previously calculated rotate variable is used to rotate the text to face the center of the circle. The text is centered and justified on the bottom of the text box. The GTL code is shown below:

```
/*Plots the labels around the circle*/
textplot x=x_text y=y_text text=text / rotate=rotate
    textattrs=(color=black size=12pt weight=bold)
    position=top vcenter=bbox;
```

The final CIRCOS example is shown below with the labels and axes added:



The CIRCOS diagram demonstrates the absolute control that a programmer has when using the POLYGON plot with generated coordinates. The width of the shapes can vary, spaces between shapes are set in absolutes, and curved shapes can be easily generated by incorporating trigonometric equations.

CREATING A SANKEY DIAGRAM

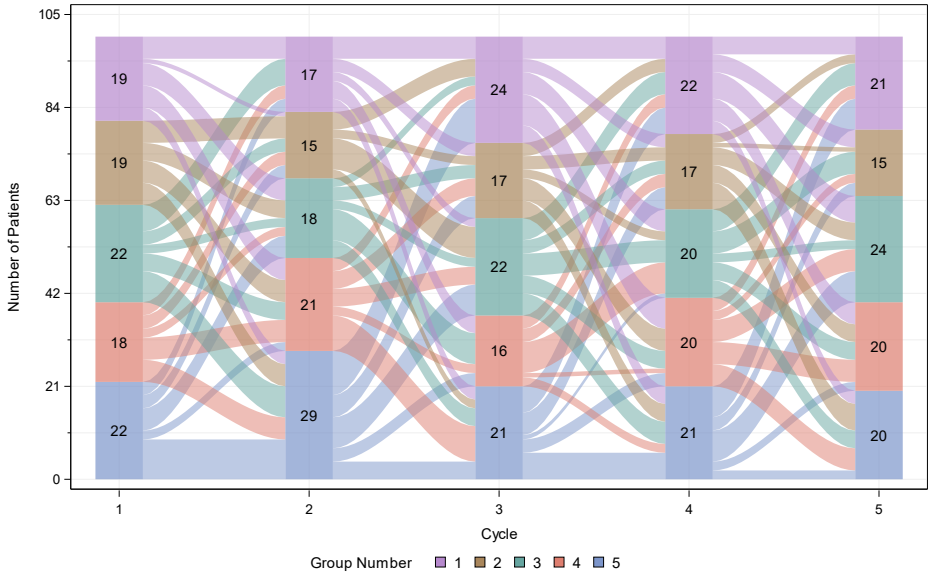
The Sankey diagram is a tool that specializes in showing the flow of patients from one group into another through different time-points, decisions, nodes, or other key points. Examples where Sankey plots are useful are longitudinal categorical data such adverse event data and quality of life (QOL) questionnaire data. The time-points are essentially stacked bar charts proportionally sized to the number of patients in each group. Between each time-point are curves showing the number of patients moving from one group to another. What makes these curves tricky to properly make is that the width of the curves is proportional to both the starting group and the ending group meaning the width of the curve is not constant from start to end. This prevents the use of simple plot statements such as SERIES from properly creating the curves.

RANDOMLY GENERATED DATA SET FOR EXAMPLES

The data set used to generate the diagram in figure five is randomly generated data using the following data step code:

```
data random;
    call streaminit(123);
    do id = 1 to 100;
        do cycle=1 to 5;
            u = rand("Uniform");
            group = 1+floor(5*u);
            output;
        end;
    end;
    drop u:;
    label cycle='Cycle' group='Group Number';
run;
```

This data set represents patients moving between levels of a group variable over several cycles. The group levels mimic variables such as tumor response values or quality of life questionnaire responses. The following shows an example Sankey diagram with randomly generated data.



The bar chart segments represent the population distribution at each cycle and the width of the connecting curves represents the proportion of patients moving from one group to the next.

COMPONENTS OF A SANKEY DIAGRAM

There are two main components of a Sankey Diagram: the stacked bar charts and the connecting curves. Other components can be added on to annotate the diagram such as counts within the segments of the bar charts or at the end of the curves.

STACKED BAR CHARTS

The stacked bar chart component of the diagram appears simple at first as SAS has a straightforward graphing

statement in VBAR. However, there are two primary reasons that POLYGON plots are more useful when generating the Sankey diagram:

1. When drawing the connecting lines it is necessary to know the exact x and y-coordinates of each segment of the bar chart to avoid overlapping the bar chart and connecting curve. This is more important when transparency is used with the shapes. Sankey diagrams also can have a buffer space between the bar chart and the connecting curve where knowing the exact coordinates makes this much easier.

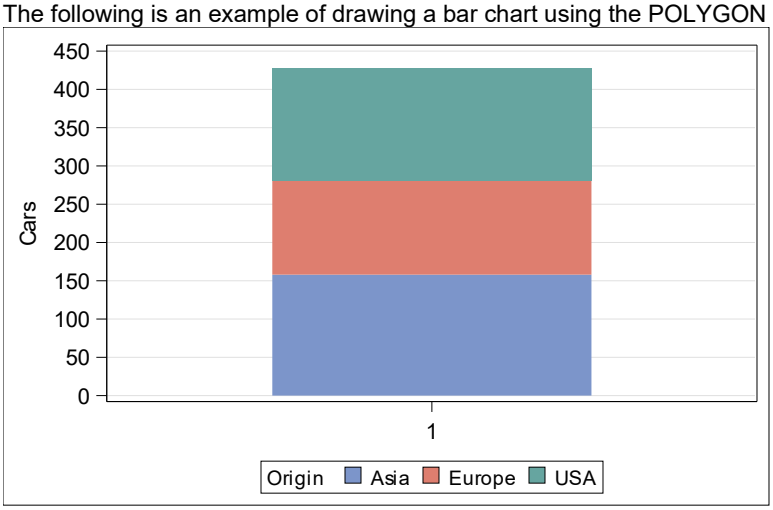
2. Some Sankey diagrams add space between the categories of the bar chart, and this is much easier to do when individually drawing each segment on its own.

The shape of each bar chart segment is a rectangle which only requires four points to be plotted in the POLYGON plot. These should be centered over the x-axis time-point for each node according to the preferred box width. The box height is determined by the number of patients within each box (one on the y-axis is equal to one patient), and these numbers can be calculated by either a simple FREQ procedure or SQL query. The y-axis coordinates are then calculated by knowing the maximum y-axis value of the previous box to be the base and adding the current group's patients to get the top of the rectangle:

```
proc freq data=sashelp.cars noprint;
    table origin / out=freq;
run;
data barchart;
    set freq;
    if _n_=1 then start=0;
    id=_n_;
    x=0.5;y=start;output;
    x=1.5;y=start;output;
    x=1.5;y=start+count;output;
    x=0.5;y=start+count;output;
    start=start+count;
    retain start;
run;

proc sgplot data=barchart ;
    polygon x=x y=y id=id / fill group=origin;
    xaxis display=(nolabel) min=0 max=2 values=(1) valueshint;
    yaxis label='Cars' min=0 max=450 values=(0 to 450 by 50) grid valueshint;
run;
```

The following is an example of drawing a bar chart using the POLYGON plot using the previous code:



Each rectangle requires the four corners to be plotted as x/y coordinate and separate ID values to distinguish different shapes. The GROUP option is used to apply color and legend values. This example was drawn with a width of exactly one and the coordinates of the edges of the rectangles is easily known for the next steps.

CONNECTING CURVES

The connecting curves show the flow of patients from one group of the stacked bar charts to another. Creating the curves properly is more difficult than anticipated for three main reasons:

1. The width is proportionate to the number of patients moving compared to both the starting group and the ending

group. This means that the two ends of the curve can have different widths meaning traditional plot statements such as SERIES will not work for this purpose

2. The width of each curve must be precise as well as the placement at each bar chart down to specific x/y coordinates. Traditional plot statements have x/y coordinates, but the width of each line cannot be precisely controlled. For example, a SERIES plot line thickness cannot be set to be one unit of the y-axis.

3. The easiest path to connect the groups is a straight line, but Sankey diagrams typically use curved lines for aesthetic reasons.

POLYGON plots solve the first two points by allowing the programmer to set precise x/y coordinates for where the connecting curves start and stop as well as the exact width. The third point requires a mathematical equation to create the appropriate curve. Research led to finding Bézier curves which have start/end points as well as one or more anchors to pull the curve in different directions. The curves in the Sankey diagram have up to two inflection points which led to using a cubic Bézier curve with the following equation:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + \mathrm{t}^3 \mathrm{P}_3, 0 \leq \mathrm{t} \leq 1$$

This equation requires four anchor points P0-P3. P0 and P3 are the start and ending points, and P1 and P2 are points between the start and end that pull the curve in their direction. The equation is run on values of t from zero (the start of the curve) to one (the end of the curve). The more steps taken between zero and one the smoother the curve will become. The Sankey diagram requires both x and y coordinates, so the equation is run once for X(t) and again for Y(t) setting the four anchor points to be:

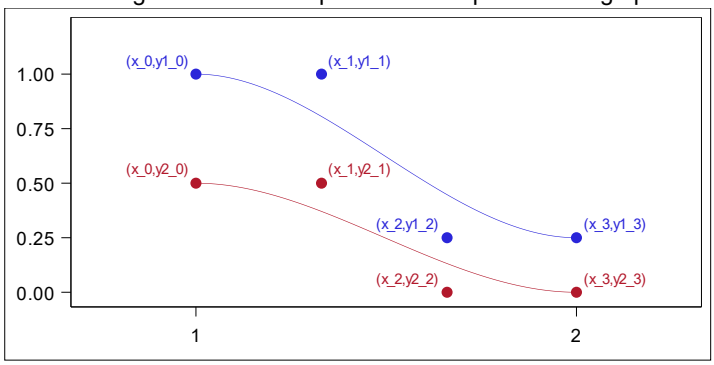
1. P_0 is the starting x/y coordinates for the curve

2. P_3 is the ending x/y coordinates for the curve

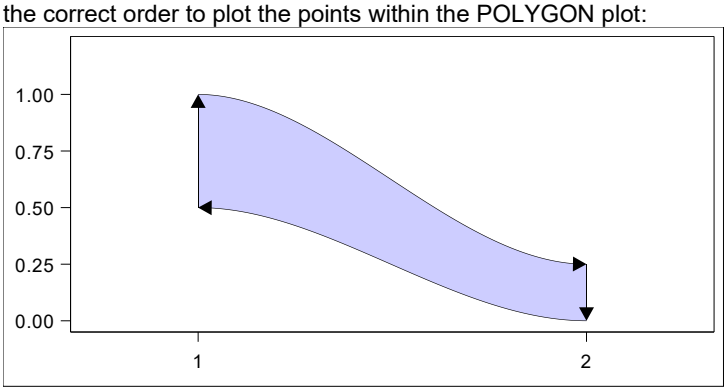
3. P_1 is one-third of the distance between the start and end for the x-coordinate and is equal to the starting y-coordinate

4. P_2 is two-thirds of the distance between the start and end for the x-coordinate and is equal to the ending y-coordinate

The following shows an example of the four points setting up cubic Bézier curves:



The middle two anchor-points can be adjusted but for the purpose of this demonstration the one-third and two-thirds distance was found to produce an aesthetically pleasing shape. This curve must be repeated twice at two different heights to produce a curved rectangle shape. The starting and ending x-coordinates will be the same in both curves and the y-coordinates will have two different start/end points depending on which side of the rectangle is being drawn. In order to take the two Bézier curves and correctly convert them into a rectangle the points must be drawn in the correct order. If drawn in the incorrect order the polygon will appear to be folded or bent. The following shows the correct order to plot the points within the POLYGON plot:



The perimeter of the polygon must be followed with the x-y coordinates. The top curve should follow the Bézier curve equation from zero to 1 for t, but the bottom curve must be plotted in the opposite direction. The following code shows the data step to output the coordinates for the polygon:

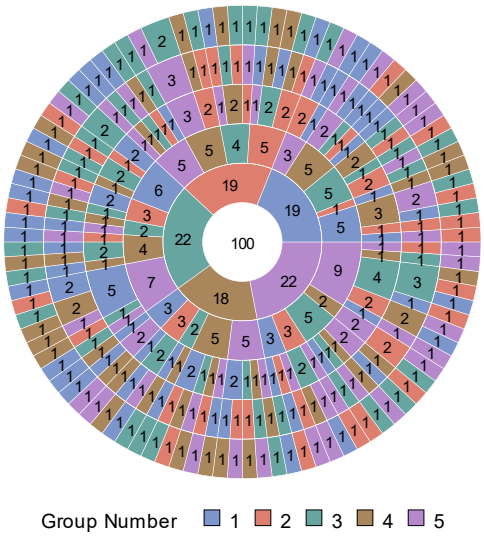
```
data bezier;
    starting_x=1;
    ending_x=2;
    start_y1=1;
    start_y2=0.5;
    end_y1=0.25;
    end_y2=0;
    id=1;
    /**Bezier Curve 1: From Left group to Right Group **/
    do t = 0 to 1 by 1/20;
        x=((1-t)**3)*starting_x+
            3*((1-t)**2)*t*(starting_x+(ending_x-starting_x)/3)+
            3*(1-t)*(t**2)*(starting_x+2*(ending_x-starting_x)/3)+
            (t**3)*ending_x;
        y=((1-t)**3)*start_y1+3*((1-t)**2)*t*start_y1+
            3*(1-t)*(t**2)*end_y1+(t**3)*end_y1;
        output;
    end;
    /**Bezier Curve 2: From Right Group back to Left Group**/
    do t = 0 to 1 by 1/20;
        x=((1-t)**3)*ending_x+
            3*((1-t)**2)*t*(starting_x+2*(ending_x-starting_x)/3)+
            3*(1-t)*(t**2)*(starting_x+(ending_x-starting_x)/3)+
            (t**3)*starting_x;
        y=((1-t)**3)*end_y2+3*((1-t)**2)*t*end_y2+
          3*(1-t)*(t**2)*start_y2+(t**3)*start_y2;
        output;
    end;
run;
```

The above code draws the top and bottom sides of the polygon in 20 steps each using DO loops. The key to putting this method into practice is to calculate the start and stop heights for each individual curve and then output the coordinates with separated ID values.

The Sankey diagram can be customized with annotation such as counts per box or counts per curve, and the TEXT plot is the best tool to draw these. The curves within each group will need to be sorted such that there is minimal overlap.

CREATING A SUNBURST CHART

The sunburst chart is a graph meant to display the same type of data as a Sankey diagram but in a format that is more conducive to following specific patient paths. Each ring represents a different time or node with the innermost ring being the first time-point and the outermost being the last time-point. Each ring is essentially a donut plot where categories are proportionally drawn as a curved rectangle. Each sequential ring can be traced backwards to see the full path over time for a group of patients. The following an example sunburst chart with the same example dataset as the Sankey diagram:



The sunburst chart displays the same data as the Sankey diagram but in a manner that allows the tracking of specific patient paths from one group to the next.

CREATING A DONUT RING

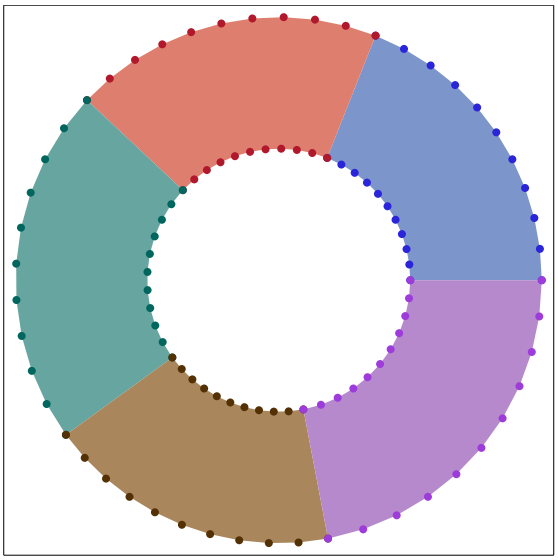
Each ring of a sunburst plot is essentially a donut plot or pie chart with the center blocked by a circle. Creating a donut plot requires drawing two arcs at different distances from the origin. The distance from one arc to the next becomes the width of the final curved rectangle segment. The following is a simple program to create the first donut ring of the sunburst plot using the randomly generated data set shown earlier:

```
proc freq data=random noprint;
    table group / out=freq;
    where cycle=1;
run;

data donut;
    set freq;
    if _n_=1 then start=0;
    id=_n_;
    end=start+360*percent/100;
    /**Draw outside perimeter**/
    do theta=start to end by (end-start)/10;
        x=1*cos(theta*constant('PI')/180);
        y=1*sin(theta*constant('PI')/180);
        output;
    end;
    /**Draw inside perimeter**/
    do theta=end to start by -(end-start)/10;
        x=0.75*cos(theta*constant('PI')/180);
        y=0.75*sin(theta*constant('PI')/180);
        output;
    end;
    start=start+percent*360/100;
    retain start;
run;
ods graphics / reset height=4in width=4in;
proc sgplot data=donut;
    polygon x=x y=y id=id / fill group=group;
    scatter x=x y=y / markerattrs=(symbol=circlefilled) group=group;
    xaxis display=none max=1.1 min=-1.1;
    yaxis display=none max=1.1 min=-1.1;
run;
```

The program first finds the frequency and percentages of the GROUP variable for CYCLE one and outputs the values

to a data set FREQ. These percentages are converted to degrees for a circle, and the trigonometric equations shown earlier to create arcs are used to draw the perimeter of the curved rectangles. The program changes direction to draw the inner arc to avoid twisting the rectangle. The resulting graph also includes a scatter plot to show the points used to draw the polygon plot:



The scatter plot overlayed on the donut plot show the points being used to draw the polygon plot. Each arc is drawn using 10 points, and the distance from the arcs to the center (the origin) is used to set the ring's width.

CREATING THE SUBSEQUENT RINGS

The majority of creating the subsequent rings is the data preparation. The percentages for each ring segment must be calculated including the previous paths. For example, the second ring could have 20 patients in group two, but they could be from three different groups in the prior ring and cannot be counted together. Once the frequencies for each ring are calculated the same graphing process is performed to create the rings, but the groups are sorted so that the correct paths through the rings align. The actual code from an internal macro is the following:

```
%do i = 1 %to &&nrings&b;
    data _ring&i;
        %if &i=1 %then %do;
            /**If first ring then setup alone**/
            set _temp&b._t3 (where=(ring=1));
        %end;
        %else %do;
            /**Merge current ring with previous ring dataset**/
            merge _temp&b._t3 (where=(ring=&i) in=a)
                  _ring%eval(&i-1) (keep=ring_id start
                                    rename=(ring_id=prior_ring
                                            start=prior_start) in=b);
        %end;
        by prior_ring;
        retain _end;
        if first.prior_ring then do;
            %if &i=1 %then %do;
                start=0;
            %end;
            %else %do;
                start=prior_start;/**Comes from prior ring data set**/
            %end;
            end=start+percent;
            _end=end;
        end;
        else do;
            start=_end;
            end=start+percent;
```

```
            _end=end;
        end;
        drop _end;
    run;
%end;
/**Put all rings together**/
data _rings;
    set %do i = 1 %to &&nrings&b; _ring&i %end;;
run;
```

The dataset _temp&b._t3 has pre-calculated the percentages for each group of each ring. Once the starting point from the prior ring's group is collected it is simple addition to find the end points of each group.

ANNOTATING EACH RECTANGLE WITH COUNTS

Knowing the starting points, ending points and the width of each rectangle allows the calculation of the center for adding in the counts via a text plot. The text can either be displayed rotated or unrotated depending on preference. The rotation for each number to face the center of the circle is given by the following equation:

$$R = \tan^{-1}\left(\frac{Y}{X}\right) * \frac{360}{2\pi}$$

The arctangent function in SAS returns the angle in radians and must be converted to degrees. The resulting value is then added to the TEXT plot in the ROTATE option.

The color of the outline is very important when differentiating between the groups and giving a buffer. When giving space between groups the size of the outline is easier to manipulate than calculating out space to add to the start/stop of groups. When creating POLYGON plots the OUTLINE portion of the plot tends to be drawn more jagged than using a SERIES plot instead. The numbers are added with a TEXT plot centered in each group.

CONCLUSION

The SAS graphics team is constantly adding in new functionality as new versions are released, but the graph types that do not exist in the software currently can still be created with math equations and the POLYGON plot. The method was originally created to create a CIRCOS diagram in a friendly competition. Further developing the techniques led to the creation of programs for the Sankey and sunburst charts. These methods are not the most efficient and require more storage space compared to regular plot statements but using the methods to create new graphs can also inspire SAS to continue updating their catalogue.

RECOMMENDED READING

Meyers, Jeffrey. April 2019. "Welcome to the Three Ring %CIRCOS: An Example of Creating a Circular Graph without a Polar Axis." *Proceedings of the SAS Global 2019 Conference*, Dallas, MN: SAS. at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3069-2019.pdf.

Meyers, Jeffrey. May 2024. "Combining Functions and the POLYGON Plot to Create Unavailable Graphs Including Sankey and Sunburst Charts." *Proceedings of the SAS Global 2024 Conference*, Baltimore, MN: SAS. at https://pharmasug.org/proceedings/2024/DV/PharmaSUG-2024-DV-155.pdf.

CONTACT INFORMATION

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

Jeffrey Meyers
Regeneron Pharmaceuticals
Jeffrey.meyers@regeneron.com