PharmaSUG 2024 - Paper DV-155

Combining Functions and the POLYGON Plot to Create Unavailable Graphs Including Sankey and Sunburst Charts

Jeffrey Meyers, Regeneron Pharmaceuticals, Basking Ridge NJ

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 to create circular, twisting, and other complex graphs including as the Sankey diagram and sunburst chart.

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 macro[1] that I previously presented at PharmaSUG generates CIRCOS plots by combining the POLYGONPLOT statement in GTL with trigonometric equations. This paper will use the same techniques to demonstrate how other unavailable graph types can also be created with the primary focus being on Sankey diagrams and sunburst charts

Figure 1 is an example CIRCOS plot

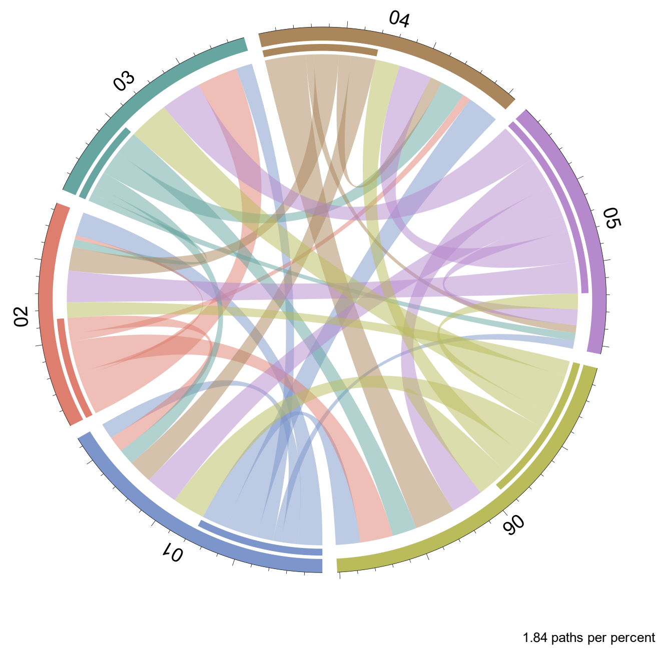

Figure 1. The CIRCOS plot shows how many patients are moving to and from different groups at two different time-points. The paper[1] describes the methods created to produce shapes not available by default with SAS graphics.

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:

Table 1 is a sample data set to input into a POLYGONPLOT statement.

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

Table 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.

Figure 2 is the graph created by table 1.

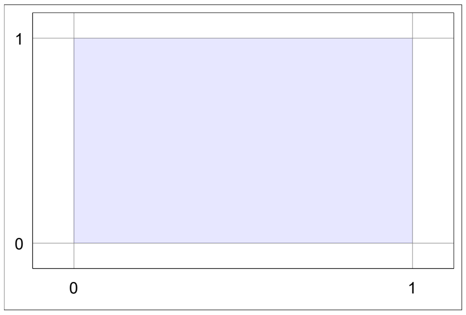

Figure 2. The square is drawn at the coordinates in the data set described by table 1.

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.

Figure 3 is an example of drawing a curve by adding more points.[1]

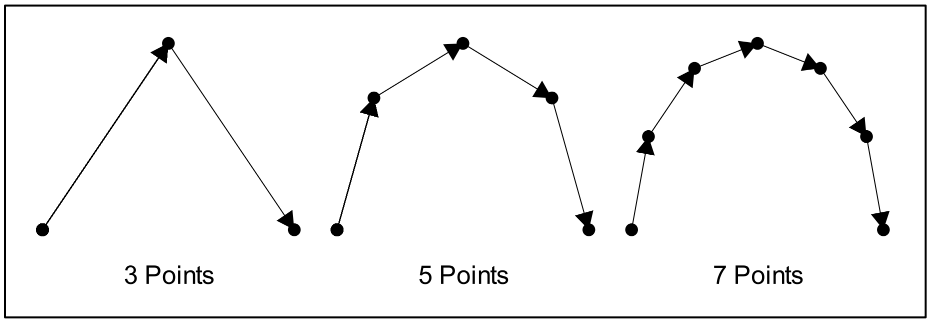

FIGURE 3. The more points that are used to draw the arc the smoother the curve becomes.

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.

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:

Figure 4 is a 90 degree arc drawn around the origin

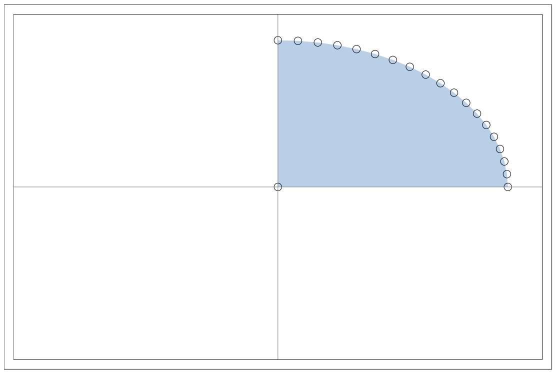

FIGURE 4. The arc is drawn with 18 coordinates and the origin. The origin is not added twice to the data set as the last point will automatically connect to the first.

Notice in the prior 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 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.

Figure 5 shows an example Sankey diagram with randomly generated data.

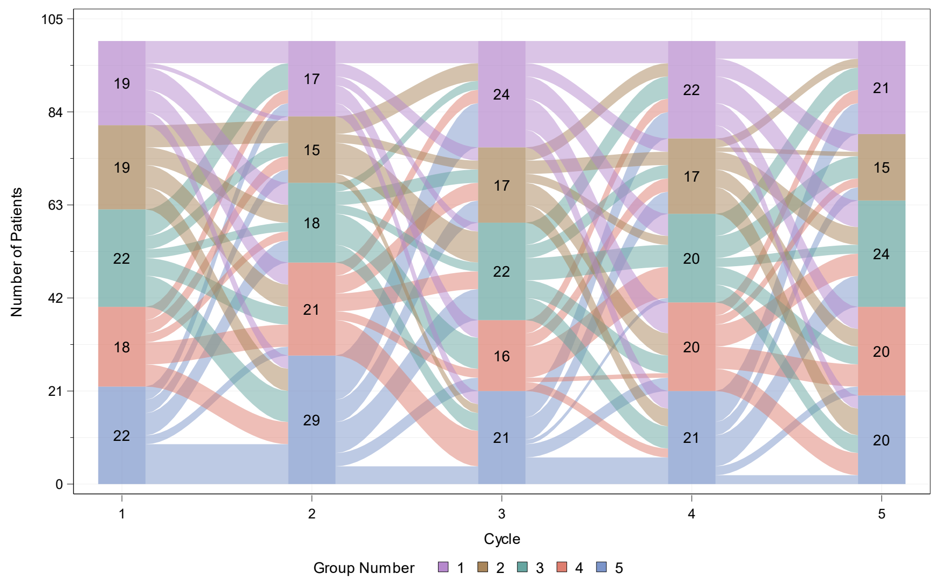

FIGURE 5. The bar chart segments represent the population distribution at each cycle and the width of the connecting curves represents the proportion of patients moving from one group to the next.

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;
```

Figure 6 is an example of drawing a bar chart using the POLYGON plot

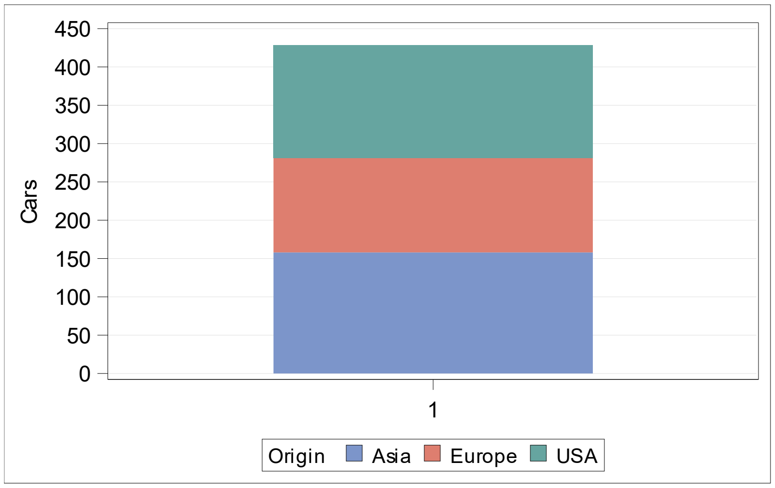

FIGURE 6. Each rectangle requires the four corners to be plotted as x/y coordinate and separate ID values to distinguish different shapes. The GROUP option is used to apply color and legend values.

Figure six is 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:

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

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

Figure 7 is an example of drawing two cubic Bézier curves with four anchor points

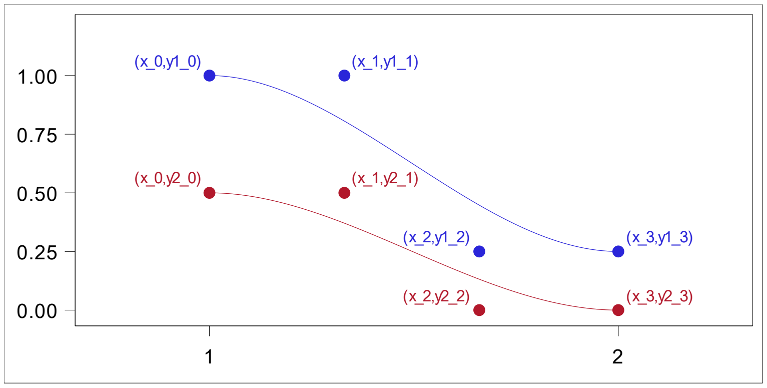

FIGURE 7. The blue curve uses the y1_0-y1_3 points and the red curve uses the y2_0-y2_3 points. Both curves use the x_0-x_3 points.

The middle two anchor-points can be adjusted but for the purpose of this macro 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 in figure 7 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.

Figure 8 shows the correct order to plot the points within the POLYGON plot.

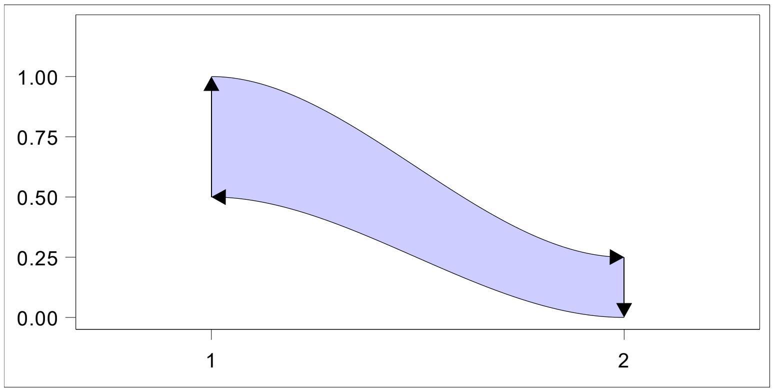

FIGURE 8. The perimeter of the polygon must be followed with the x-y coordinates. The top curve should follow the Bézier curve equation from zero to 1 for t, but the bottom curve must be plotted in the opposite direction.

The following code shows the data step to output the coordinates for the polygon in figure 7:

```
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.

FINAL REMARKS

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.

Figure 9 shows an example sunburst chart with randomly generated data.

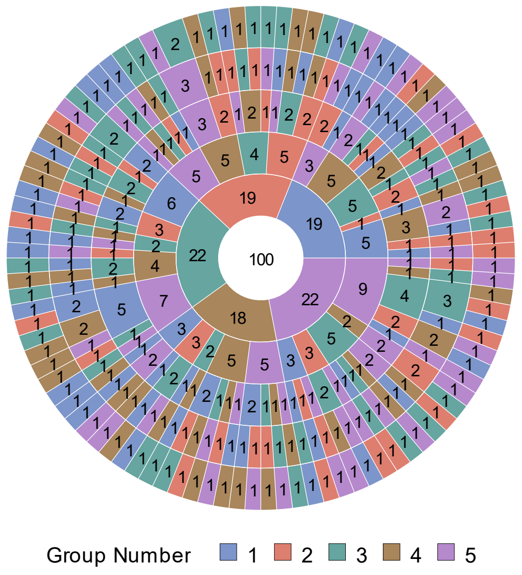

FIGURE 9. The sunburst chart displays the same data as the Sankey diagram but in a manner that allows the tracking of specific patient paths from one group to the next.

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.

Figure 8 shows the innermost ring of figure 10

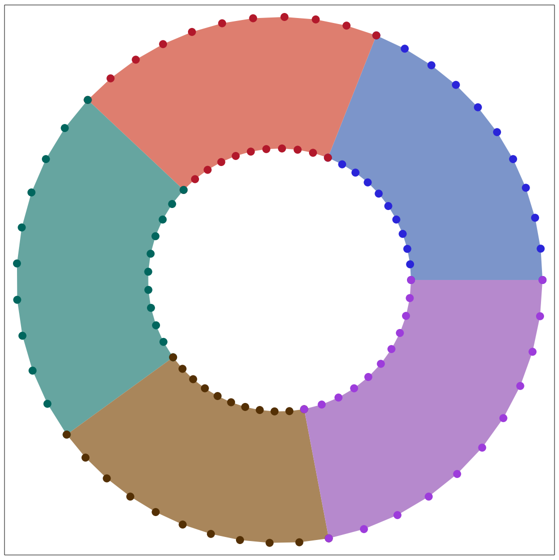

FIGURE 10. The scatter plot overlayed on the donut plot show the points being used to draw the polygon plot. Each arc is drawn using 10 points, and the distance from the arcs to the center (the origin) is used to set the ring's width.

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 the 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.

FINAL REMARKS

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.

ADDITIONAL PLOT EXAMPLES

EXPLODED PIE CHART

The pie chart is a common and well-known graph that SAS already has several methods of creating. There is a variation of this graph unavailable in SAS called the "exploded" pie chart that is used when several of the categories are too small to distinguish in the pie. The exploded portion is shown as a second pie within the graph including all groups under a certain threshold and these groups are combined into an "Other" category in the first pie.

Figure 11 shows an example of a regular pie chart vs an exploded pie chart

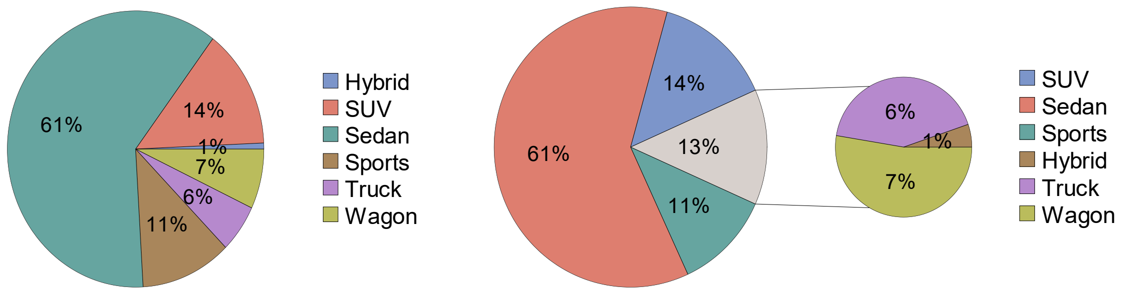

FIGURE 11. The pie charts are generated from the SASHELP.CARS data set and are showing the distribution of the types of vehicles. The Hybrid category is very small and hard to see in the left figure. The right figure "explodes" the smaller categories making them easier to see.

There are more practical applications of the exploded pie chart when there are more small categories in figure 11 but the methods are the same.

Creating a Pie Chart

Creating a pie chart using the POLYGON plot statement is simply creating proportional pie slices such as the example in figure three and is a simpler version of the donut chart from figure eight. The percentages for each group represent to the proportion of 360 degrees each pie slice encompasses. The easiest location to draw a pie chart in linear axes is around the origin (x=0, y=0) using trigonometric functions where the distance of the arc from the origin becomes the radius of the circle. Nearly the same code to make the donut plot can be used to make a pie chart:

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

```
data pie;
    set freq;
    if _n_=1 then start=0;
    id=_n_;
    end=start+360*percent/100;
    /**Mark Origin**/
    x=0;y=0;output;
    /**Draw arc**/
    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;
    start=start+percent*360/100;
    retain start;
run;
ods graphics / reset height=4in width=4in;
proc sgplot data=pie;
    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;
```

Figure 12 shows the resulting pie chart

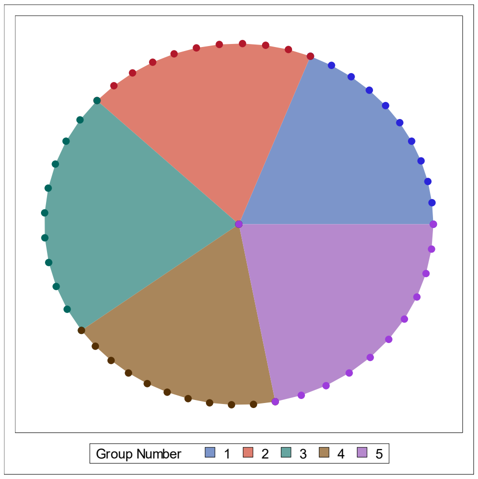

FIGURE 12. The pie chart is the same as the donut plot without cutting out the hole in the center.

Creating Multiple Pie Charts in The Same Graph

The limitation of SAS's built-in pie chart functionality is there are no axes and other plot types cannot be overlayed into the graph. When building a pie chart with the POLYGON plot on a linear access an unlimited number of pie charts and other graph types can be overlayed together. The same equation for the arcs is used but the origin points are moved to avoid the pie charts overlapping. The equations referenced the origin point as X_0 and Y_0 which for the example that generated figure 12 are both set to 0 to simplify the equation. The following example creates two bar charts where the second one is translated 2.1 units to the right:

```
proc freq data=random noprint;
    table cycle*group /
        out=freq (drop=percent rename=(pct_row=percent)) outpct;
    where cycle in(1 2);
run;
data pie;
    set freq;
    by cycle;
    if first.cycle then do;
        start=0;
        x_text=(cycle-1)*2.1;y_text=1.1;text=catx(' ','Cycle',cycle);
    end;
    id=_n_;end=start+360*percent/100;
    /**Mark Origin**/
    x=(cycle-1)*2.1;y=0;output;call missing(x_text,y_text,text);
    /**Draw arc**/
    do theta=start to end by (end-start)/10;
        x=(cycle-1)*2.1+(1/cycle)*cos(theta*constant('PI')/180);
        y=(1/cycle)*sin(theta*constant('PI')/180);output;
    end;
    start=start+percent*360/100;
    retain start;
run;
ods graphics / reset height=4in width=8in;
proc sgplot data=pie;
    polygon x=x y=y id=id / fill group=group;
    scatter x=x y=y / markerattrs=(symbol=circlefilled) group=group;
    text x=x_text y=y_text text=text / position=center textattrs=(size=10pt);
    xaxis display=none max=3.1 min=-1.1;
    yaxis display=none max=1.1 min=-1.1;
run;
```

Figure 13 shows the two side-by-side pie charts

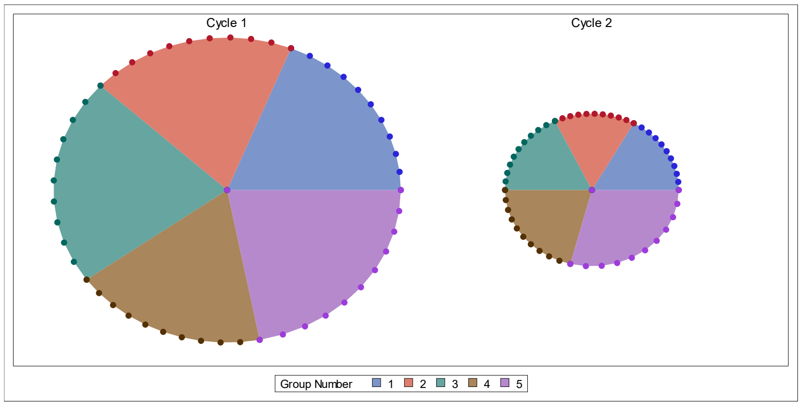

FIGURE 13. The left pie chart is centered on the origin (x=0, y=0) and the right pie chart is centered on x=2.1, y=0. Changing the distance of the arcs changes the radius of the circle.

Closing Remarks

The remaining steps for the exploded pie chart are annotation. The numbers can be added with TEXT plots after calculating the center point for each pie slice. The lines connecting the two pie charts together can run from the min/max of the combined pie slice to the top/bottom of the second pie chart. Different textures can be added to the pie charts with the DATASKIN option. A simple macro that created figure 11 will be included with the paper.

EXPLODED CIRCOS DIAGRAM

The CIRCOS plot shown at the beginning of this paper is complicated to read with paths entering and exiting each section. One way to simplify the figure is to “explode” it and only feature paths going one direction.

Figure 14 shows patients moving from one group to another

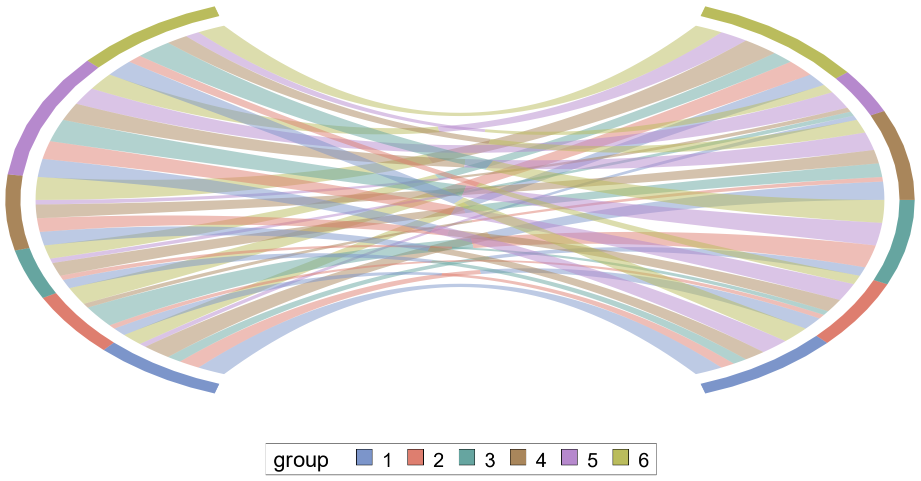

FIGURE 14. The left side is the original group and the right side is the new group. The paths change colors halfway through so that the groups patients are leaving to on the left and the groups patients are coming from on the right can be identified without having to trace the line.

The Anatomy of the CIRCOS Diagram

The details of creating a CIRCOS diagram are demonstrated in a prior paper[1] and is displayed in figure one of this paper. The two primary components of the CIRCOS diagram are the proportionally sized outside rectangles representing populations and the curves that connect the populations showing the flow from one group to another. The connecting curves are sized proportionally to both the starting group and the ending group and change widths from start to finish. The color of the connecting curve can reflect the starting group or the ending group.

Creating the CIRCOS Diagram Components

Breaking the CIRCOS diagram down into simple components leads to need to build the outside rectangles and the connecting curves. The outside rectangles are created with the same methods used to make the doughnut rings in the prior sunburst example of this paper. The connecting curves use the same methods as the Sankey example within this paper, but instead of using cubic Bezier curves the quadratic version is used:

$$B(t) = (1-t)P_0 + 2(1-t)tP_1 + \mathrm{t}^2\mathrm{P}_2, 0 \leq \mathrm{t} \leq 1$$

Where the variables represent the following:

1. P_0 is the starting x/y coordinates for the curve
2. P_2 is the ending x/y coordinates for the curve
3. P_1 is the midpoint x/y coordinate between the start and the end of the curve

The code creating the CIRCOS graph strategically picks the midpoint to be the coordinate X=0 and Y=0 to remove the P_1 point from the equation. Normally a CIRCOS plot is created around a singular circle around this midpoint, but an “exploded” CIRCOS plot variant is created using two circles pulled apart:

Figure 15 shows the graph space for creating the exploded CIRCOS diagram

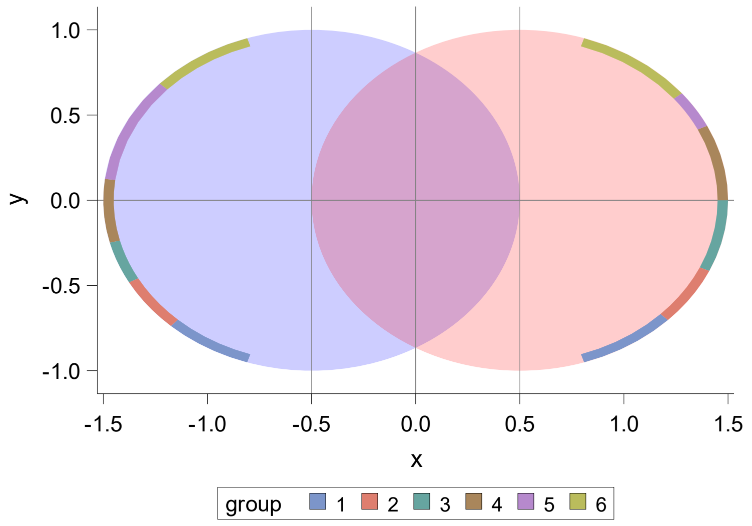

FIGURE 15. Two circles pulled apart from the midpoint by 0.5 in both directions are used for the starting and ending points

Figure 15 demonstrates the use of two circles pulled apart to create the starting and ending points (P_0 and P_2 respectively). The midpoint for the Bezier curves is still set to be the coordinate X=0 and Y=0 to simplify the equation. The data step code for the Bezier curves contains the following:

```
    id=catx('-','CR',before,after);

    group=after;

    /**Draw left edge of Bezier Curve**/
```

```
do t=before_end to before_start by (before_start-before_end)/5;
    x=0.9*cos(t)-0.5;y=0.9*sin(t);output;
end;
/**Draw bottom edge of Bezier Curve**/
do t = 0 to 0.5 by 1/50;
    x=(1-t)**2*bx1 + t**2*ax1;
    y=(1-t)**2*by1 + t**2*ay1;
    output;
end;
/**Draw top edge of Bezier Curve**/
do t = 0.5 to 0 by -1/50;
    x=(1-t)**2*bx2 + t**2*ax2;
    y=(1-t)**2*by2 + t**2*ay2;
    output;
end;
```

The above code outputs the first half of the Bezier curve by plotting t from zero to 0.5 in the quadratic equation. There is a section for the endcap of the curve and sections for the bottom and top edges of the curve. These must be done in the appropriate order or else the Bezier curve will fold on itself. The starting corner coordinates are represented by bx1, by1, bx2, and by2. The ending corner coordinates are represented by ax1, ay1, ax2, and ay2. The second half of the Bezier curve is drawn by plotting t from 0.5 to 1:

```
id=catx('-','CR2',before,after);
group=before;
/**Draw bottom edge of Bezier Curve**/
do t = 0.5 to 1 by 1/50;
    x=(1-t)**2*bx1 + t**2*ax1;
    y=(1-t)**2*by1 + t**2*ay1;
    output;
end;
/**Draw right edge of Bezier Curve**/
do t=after_start(after) to after_end by (after_end-after_start(after))/5;
    x=0.9*cos(t)+0.5;y=0.9*sin(t);output;
end;
/**Draw top edge of Bezier Curve**/
do t = 1 to 0.5 by -1/50;
    x=(1-t)**2*bx2 + t**2*ax2;
    y=(1-t)**2*by2 + t**2*ay2;
    output;
end;
```

The ID variable determines the color of each section of the Bezier curve. By assigning different values (the before and after group values) the sections of the Bezier curve can match the rectangles on the outside of the figure to indicate where the curve is going and where it is coming from.

Figure 16 shows the Bezier curves overlayed over the two circles

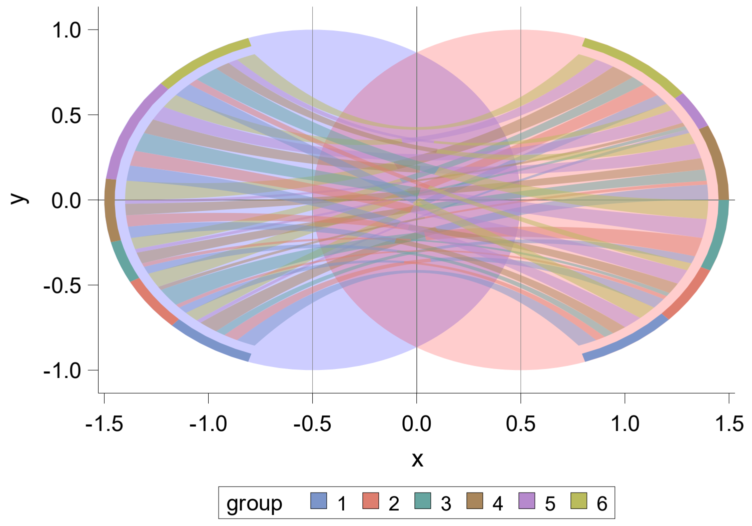

FIGURE 16. The connecting curves are pulled towards the X=0 and Y=0 coordinate.

This example also shows the connecting curves changing color halfway through the length. This is done by assigning the after group to the first half of the Bezier curve and the before group to the second half of the Bezier curve for color mapping.

Closing Remarks

The exploded CIRCOS diagram is another example of creating an unavailable graph using the right plot space setup, the right function, and the polygon plot. This example also demonstrated some flexibility in the polygon plot by separating the curve made by the polygon plat in half to show two different colors.

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 explained within a prior paper[1] describing how to create a CIRCOS diagram. Multiple examples utilizing these methods were displayed within this paper. 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.

REFERENCES

[1]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.

RECOMMENDED READING

- *Welcome to the Three Ring %CIRCOS: An Example of Creating a Circular*

CONTACT INFORMATION

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

Jeffrey Meyers
Regeneron Pharmaceuticals
Jeffrey.meyers@regeneron.com