%PlotIt Macro

The %PlotIt autocall macro makes graphical scatter plots of labeled points. It is particularly designed to display raw data and results from analyses such as regression, correspondence analysis, MDPREF, PREFMAP, and MDS. However, it can make many other types of graphical displays as well. It can plot points, labeled points, vectors, circles and density. See pages 27–40 and 1231–1274, for example, plots and more about these methods. The %PlotIt macro is not needed nearly as much now as it was in previous SAS releases. Now, ODS Graphics automatically does much of what the %PlotIt macro was originally designed for, and usually, ODS Graphics does it better and more conveniently. Comparisons between the %PlotIt macro and ODS Graphics are as follows. The %PlotIt macro has a much more sophisticated algorithm for placing labels and avoiding label collisions. The %PlotIt macro has a few other capabilities that are not in ODS Graphics (e.g. more sophisticated color ramps or "painting" for some applications). ODS Graphics is superior to the %PlotIt macro in virtually every other way, and ODS Graphics has *many* capabilities that the %PlotIt macro does not have. The algorithm that ODS Graphics has for label placement in 9.2, while clearly nonoptimal, is good enough for many analyses.

The macro, by default, uses the last data set created. The macro creates an output Annotate data set that cannot be used as input to the macro, so you must specify data= if you run %PlotIt a second time.

By default (at least by default when a device like WIN is in effect on a PC), the %PlotIt macro creates a graphical scatter plot on your screen. If no graphics device has been previously specified (either directly or indirectly), you will be prompted for a device as follows:

```
No device name has been given--please enter device name:
```

Enter your graphics device. This name is remembered for the duration of your SAS session or until you change the device. You can modify the gopprint= and gopplot= options to set default devices so that you will not be prompted. Note that all graphics options specified in a goptions statement (except device=) are ignored by default. Use the macro options gopprint=, gopplot=, gopts2=, and gopts= to set goptions.

To display a plot on your screen using the default goptions in a PC environment, specify, for example, the following:

```
goptions device=win;
%plotit(data=sashelp.class(drop=age sex))
```

To create a PDF file named myplot.pdf, suitable for printing, specify the following:

```
ods listing style=statistical;
%plotit(data=sashelp.class(drop=age sex),
        gopts=device=pdf, method=print, post=myplot.pdf)
```

The ODS LISTING statement is only necessary in this example if you want to set the ODS style.

To create HTML output, specify the following:

```
goptions device=png;
ods listing close;
ods html body='b.html';
%plotit(data=sashelp.class(drop=age sex))
ods html close;
```

To create RTF output, specify the following:

```
goptions device=png;
ods listing close;
ods rtf file='myplot.rtf';
%plotit(data=sashelp.class(drop=age sex))
ods rtf close;
```

To create a PNG file, perhaps just to insert into a document, specify the following:

```
goptions device=png;
ods listing style=analysis;
%plotit(data=sashelp.class(drop=age sex), method=print, post=myplot.png)
ods listing close;
```

Use `gout=` to write the plot to a catalog.

The default color scheme and a few other options have changed with this release in ways consistent with SAS/GRAPH procedures. Like SAS/GRAPH procedures in 9.2, the appearance of the output is in part controlled by the `gstyle` system option. The following statements enable and disable this option:

```
options gstyle;
options nogstyle;
```

With `gstyle` in effect (which will typically be the default), the output will look much different (and probably better) than it looked in previous releases. In part, this is due to the defaults for several options changing. For example, with `gstyle`, the macro will use hardware fonts by default with the style instead of Swiss software fonts. Mostly, however, the differences are due to the output being at least in part controlled by the ODS style now. Specify the `nogstyle` system option to get the old appearance. Note that by default, the listing ODS destination is open with `style=listing`. That is the style that is used if you do not open any other destination or specify any other style. You can open other destinations, e.g. HTML (by default `style=default`), printer (by default `style=printer`), RTF (by default `style=rtf`), and so on by specifying them in an ODS destination statement. Other styles of particular interest include `statistical`, `analysis`, and `journal`. There are many other styles as well. Each destination has a default style, and each permits a style specification. The following statement enables the HTML destination with the `statistical` style:

```
ods html body='b.html' style=statistical;
```

If listing (with `style=listing`) and one other destination are open, the macro will run using the style from the other destination. If other destinations are open as well, the macro will only run if there is one style other than `listing`. Hence, submitting the following statements before running the macro will cause the macro to issue an error and quit:

```
ods html;
ods rtf;
ods printer;
```

The `default`, `rtf` and `printer` styles are in effect, and the macro has no basis to choose between them. However, the macro will run fine with the following destinations since there is only one style other than the `listing` style in effect, and that is `style=default`:

```
ods html body='b.html';
ods rtf style=default;
ods printer style=default;
```

When you specify a destination other than the listing destination, you can usually make everything run faster by closing the listing destination, if you are not interested in results from that destination, for example, as follows:

```
ods listing close;
```

Note that a macro such as the `%PlotIt` macro only has limited and varying access to style information. For most styles, it gets all of the information it needs, however, for some styles, that information is just not there, so you might not always get all of the right colors such as background colors.

If you do not like the default background color, specify `gopplot=cback=`*some-color*, (substituting your favorite color for some-color). This controls the area outside the graph. The area inside the graph is controlled by the `cframe=` option. You can permanently change the default by modifying the macro. Similarly, you can change `color=` and `colors=` as you see fit. The following step creates a plot with a white background inside the axes and a light gray background around the axes:

```
%plotit(data=sashelp.class(drop=age sex), cframe=white, gopts=cback=ligr)
```

If you are specifying colors, you will probably want to use the `nogstyle` option so that the style does not override the colors that you specify.

Sample Usage

For many plots, you only need to specify the `data=` and `datatype=` options. The following steps perform a simple correspondence analysis, however, PROC CORRESP and ODS Graphics can automatically make the same plot:

```
*------Simple Correspondence Analysis------;
proc corresp all data=cars outc=coor;
   tables marital, origin;
   title 'Simple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=corresp)
```

The following steps perform a multiple correspondence analysis, however, PROC CORRESP and ODS Graphics can automatically make the same plot:

```
*------Multiple Correspondence Analysis------;
proc corresp mca observed data=cars outc=coor;
   tables origin size type income home marital sex;
   title 'Multiple Correspondence Analysis';
   run;

%plotit(data=coor, datatype=mca)
```

The following steps perform a multidimensional preference analysis, however, PROC PRINQUAL and ODS Graphics can automatically make the same plot:

```
*------MDPREF------;
proc prinqual data=carpref out=results n=2 replace mdpref;
   id model mpg reliable ride;
   transform ide(judge1-judge25);
   title 'Multidimensional Preference (MDPREF) Analysis';
   run;

%plotit(data=results, datatype=mdpref 2.5)
```

The vector lengths are increased by a factor of 2.5 to make a better graphical display.

The following steps perform a preference mapping vector model, however, PROC TRANSREG and ODS Graphics can automatically make the same plot:

```
*------PREFMAP, Vector Model------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model ide(mpg reliable ride)=identity(prin1 prin2);
   output tstandard=center coefficients replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Vector';
   run;

%plotit(data=tresult1, datatype=vector 2.5)
```

Again, the vector lengths are increased by a factor of 2.5 to make a better graphical display.

The following steps perform a preference mapping ideal point model, however, PROC TRANSREG and ODS Graphics can automatically make the same plot:

```
*------PREFMAP, Ideal Point------;
proc transreg data=results(where=(_type_ = 'SCORE'));
   model identity(mpg reliable ride)=point(prin1 prin2);
   output tstandard=center coordinates replace out=tresult1;
   id model;
   title 'Preference Mapping (PREFMAP) Analysis - Ideal';
   run;

%plotit(data=tresult1, datatype=ideal, antiidea=1)
```

The `antiidea=1` option is specified to handle anti-ideal points when large data values are positive or ideal.

The following steps perform multidimensional preference analysis, however, PROC PRINQUAL and ODS Graphics can automatically make the same plot:

```
*------MDPREF, labeled vector end points------;
proc prinqual data=recreate out=rec mdpref rep;
   transform identity(sub:);
   id activity active relaxing spectato;
   title 'MDPREF of Recreational Activities';
   run;

%plotit(data=rec, datatype=mdpref2 3,
        symlen=2, vechead=, adjust1=%str(
        if _type_ = 'CORR' then do;
           __symbol = substr(activity,4);
           __ssize  = 0.7;
           activity = ' ';
           end;))
```

The `mdpref2` specification means MDPREF and label the vectors *too*. The vector lengths are increased by a factor of 3 to make a better graphical display. The `symlen=2` option specifies two-character symbols. The specification `vechead=`, (a null value) means no vector heads since there are labels. The `adjust1=` option is used to add full SAS DATA step statements to the preprocessed data set. This example processes `_type_ = 'CORR'` observations (those that contain vector the coordinates) the original variable names (`sub1`, `sub2`, `sub3`, ..., from the `activity` variable) and creates symbol values (1, 2, 3, ...) of size 0.7. The result is a plot with each vector labeled with a subject number.

The following steps create a contour plot, displaying density with color:

```
*------Bivariate Normal Density Function------;
proc iml;
   title 'Bivariate Normal Density Function';
   s = inv({1 0.25 , 0.25 1});
   m = -2.5; n = 2.5; inc = 0.05; k = 0;
   x = j((1 + (n - m) / inc) ** 2, 3, 0);
   c = sqrt(det(s)) / (2 * 3.1415);
   do i = m to n by inc;
      do j = m to n by inc;
         v = i || j; z = c * exp(-0.5 * v * s * v`);
         k = k + 1; x[k,] = v || z;
         end;
      end;
   create x from x[colname={'x' 'y' 'z'}]; append from x;
   quit;

%plotit(datatype=contour, data=x, extend=close,
        paint=z white blue magenta red)
```

The `paint=z white blue magenta red` option specifies that color interpolation is based on the variable `z`, going from white (zero density) through blue, magenta, and to red (maximum density). This color list is designed for a background of white. The option `extend=close` is used with contour plots so that the plot boundaries appear exactly at the edge of the contour data. By default, `%PlotIt` usually adds a bit of extra white space between the data and the plot boundaries which provides extra room for labels, which are not used in this example. Similar plots can be made with ODS Graphics.

The goal of this next example is to create a plot of the Fisher iris data set with each observation identified by its species. Similar and nicer plots can be made with ODS Graphics. Species name is centered at each point's location, and each species name is plotted in a different color. This scatter plot is overlaid on the densities used by PROC DISCRIM to classify the observations. There are three densities, one for each species. Density is portrayed by a color contour plot with white (the assumed background color) indicating a density of essentially zero. Yellow, orange, and red indicate successively increasing density.

The `data=` option names the input SAS data set. The `plotvars=` option names the y-axis and x-axis variables. The `labelvar=_blank_` option specifies that all labels are blank. This example does not use any of PROC PLOT's label collision avoidance code. It simply uses PROC PLOT to figure out how big to make the plot, and then the macro puts everything inside the plot independently of PROC PLOT, so the printer plot is blank. The `symlen=4` option specifies that the maximum length of a symbol value is 4 characters. This is because we want the first four characters of the species names as symbols. The `exttypes=symbol contour` option explicitly specifies that PROC PLOT will know nothing about the symbols or the contours. They are external types that are added to the graphical plot by the macro after PROC PLOT has finished. The `ls=100` option specifies a constant line size. Since no label avoidance is done, there can be no collisions, and the macro will not iteratively determine the plot size. The default line size of 65 is too small for this example, whereas `ls=100` makes a better display. The `paint=` option specifies that based on values the variable `density`, colors should be interpolated ranging from white (minimum `density`) to yellow to orange to red (maximum `density`). The `rgbtypes=contour` option specifies that the `paint=` option should apply to contour type observations.

The grid (created with the loops: `do sepallen = 30 to 90 by 0.6;` and `do petallen = 0 to 80 by 0.6;`) is not square, so for optimal results the macro must be told the number of horizontal and vertical positions. The PLOTDATA DATA step creates these values and stores them in macro variables `&hnobs` and `&vnobs`, so the specification `hnobs=&hnobs, vnobs=&vnobs`, specifies the grid size. Of course these values could have been specified directly instead of through symbolic variables. The `excolors=CXFFFFFF` option is included for efficiency. The input data consist of a large grid for the contour plot. Most of the densities are essentially zero, so many of the colors are `CXFFFFFF`, which is white, computed by `paint=`, which is the same color as the background. (See the `paint=` option, page 1204 for information about CX*rrggbb* color specifications.) Excluding them from processing makes the macro run faster and creates smaller datasets.

This example shows how to manually do the kinds of things that the `datatype=` option does for you with standard types of data sets. The macro expects the data set to contain observations of one or more types. Each type is designated by a different value in a variable, usually named `_type_`. In this example, there are four types of observations, designated by the `_type_` variable's four values, 1, 2, 3, 4, which are specified in the `types=` option. The `symtype=` option specifies the symbol types for these four observation types. The first three types of observations are `symbol` and the last type, `_type_` = 4, designates the contour observations. The first three symbols are the species names (`symbols=` values) displayed in `symfont=none` (hardware font). The last symbol is null because contours do not use symbols. The first three symbols, since they are words as opposed to a single character, are given a small size (`symsize=0.6`). A value of 1 is specified for the symbol size for contour type observations. The macro determines the optimal size for each color rectangle of the contour plot. The following steps process the data, perform the analysis, and create the plot:

```
*------Discriminant Analysis------;
data plotdata;  * Create a grid over which DISCRIM outputs densities.;
   do SepalLength = 30 to 90 by 0.6;
      h + 1; * Number of horizontal cells;
      do PetalLength = 0 to 80 by 0.6;
         n + 1; * Total number of cells;
         output;
         end;
      end;
   call symput('hnobs', compress(put(h    , best12.))); * H grid size;
   call symput('vnobs', compress(put(n / h, best12.))); * V grid size;
   drop n h;
   run;

proc discrim data=iris testdata=plotdata testoutd=plotd
             method=normal pool=no short noclassify;
   class species;
   var PetalLength SepalLength;
   title 'Discriminant Analysis of Fisher (1936) Iris Data';
   title2 'Using Normal Density Estimates with POOL=NO';
   run;

data all;
   * Set the density observations first so the scatter plot points
     are on top of the contour plot.  Otherwise the contour plot
     points will hide the scatter plot points.;
   set plotd iris(in=iris);
   if iris then do;
      _type_ = species; * unformatted species number 1, 2, 3;
      output;
      end;
   else do;
      _type_ = 4; * density observations;
      density = max(setosa,versicolor,virginica);
      output;
      end;
   run;

%plotit(data=all, plotvars=PetalLength SepalLength, labelvar=_blank_,
        symlen=4, exttypes=symbol contour, ls=100, gopplot=cback=white,
        paint=density white yellow orange red, rgbtypes=contour,
        hnobs=&hnobs, vnobs=&vnobs, excolors=CXFFFFFF,
        types  =1      2          3          4,
        symtype=symbol symbol     symbol     contour,
        symbols=Set    Vers       Virg       '',
        symsize=0.6    0.6        0.6        1,
        symfont=none   none       none       solid
        )

title;
```

How %PlotIt Works

You create a data set either with a DATA step or with a procedure. Then you run the macro to create a graphical scatter plot. This macro is not a SAS/GRAPH procedure (although it uses PROC GANNO), and in many ways, it does not behave like a typical SAS/GRAPH procedure. The `%PlotIt` macro performs the following steps.

1. The `%PlotIt` macro reads an input data set and preprocesses it. The preprocessed data set contains information such as the axis variables, the point-symbol and point-label variables, and symbol and label types, sizes, fonts, and colors. The nature of the preprocessing depends on the type of data analysis that generated the input data set. For example, if the option `datatype=mdpref` had been specified with an input data set created by PROC PRINQUAL for a multidimensional preference analysis, then the `%PlotIt` macro creates blue points for `_type_ = 'SCORE'` observations and red vectors for `_type_ = 'CORR'` observations.
2. A DATA step, using the DATA Step Graphics Interface, determines how big to make the graphical plot.
3. PROC PLOT determines where to position the point labels. By default, if some of the point label characters are hidden, the `%PlotIt` macro recreates the printer plot with a larger line and page size, and hence creates more cells and more room for the labels. Note that when there are no point labels, the printer plot might be empty. All of the information that is in the graphical scatter plot can be stored in the `extraobs=` data set. All results from PROC PLOT are written to data sets with ODS. The macro will clear existing `ods select` and `ods exclude` statements.
4. The printer plot is read and information from it, the preprocessed data set, and the extra observations data set are combined to create an Annotate data set. The label position information is read from the PROC PLOT output, and all of the symbol, size, font, and color information is extracted from the preprocessed (or extra observations) data set. The Annotate data set contains all of the instructions for drawing the axes, ticks, tick marks, titles, point symbols, point labels, axis labels, and so on. Circles can be drawn around certain locations, and vectors can be drawn from the origin to other locations.
5. The Annotate data set is displayed with the GANNO procedure. The `%PlotIt` macro does not use PROC GPLOT.

Debugging

When you have problems, try `debug=vars` to see what the macro thinks you specified. It is also helpful to specify: `debug=mprint notes`. You can also display the final Annotate data set and the preprocessing data set, for example, as follows:

```
options ls=180;
proc print data=anno uniform;
   format text $20. comment $40.;
   run;

proc print data=preproc uniform;
   run;
```

Advanced Processing

You can post-process the Annotate DATA step to change colors, fonts, undesirable placements, and so on. Sometimes, this can be done with the `adjust4=` option. Alternatively, when you specify `method=none`, you create an Annotate data set without displaying it. The data set name is by default WORK.ANNO. You can then manipulate it further with a DATA step or PROC FSEDIT to change colors, fonts, or sizes for some labels; move some labels; and so on. If the final result is a new data set called ANNO2, you can display it by running the following:

```
proc ganno annotate=anno2;
   run;
```

Notes

With `method=print`, the macro creates a file. See the `filepref=` and `post=` options and make sure that the file name does not conflict with existing names.

This macro creates variable names that begin with two underscores and assumes that these names will not conflict with any input data set variable names.

It is not feasible with a macro to provide the full range of error checking that is provided with a procedure. Extensive error checking is provided, but not all errors are diagnosed.

Not all options will work with all other options. Some combinations of options might produce macro errors or Annotate errors.

This macro might not be fully portable. When you switch operating systems or graphics devices, some changes might be necessary to get the macro to run successfully again.

Graphics device differences might also be a problem. We do not know of any portability problems, but the macro has not been tested on all supported devices.

Some styles are structured in such a way that the `%PlotIt` macro cannot extract information from them. For those styles, default colors are used instead.

This macro tries to create a plot with equated axes, where a centimeter on one axis represents the same data range as a centimeter on the other axis. The only way to accomplish this is by explicitly and jointly controlling the `hsize=`, `vsize=`, `hpos=`, and `vpos= goptions`. By default, the macro tries to ensure that all of the values work for the specific device. See `makefit=`, `xmax=`, and `ymax=`. By default the macro uses GASK to determine `xmax` and `ymax`. If you change any of these options, your axes might not be equated. Axes are equated when `vsize` × `hpos` / `hsize` × `hpos` = `vtoh`.

When you are plotting variables that have very different scales, you might need to specify appropriate tick increments for both axes to get a reasonable plot, like this for example: `plotopts=haxis=by 20 vaxis=by 5000`. Alternatively, just specifying the smaller increment is often sufficient: `plotopts= haxis= by 20`. Alternatively, specify `vtoh=`, (null value) to get a plot like PROC GPLOT's, with the window filled.

By default, the macro iteratively creates and recreates the plot, increasing the line size and the flexibility in the `placement=` list until there are no penalties.

The SAS system option `ovp` (overprint) is not supported by this macro.

%PlotIt Macro Options

The following options can be used with the `%PlotIt` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`adjust1=`*SAS-statements*	adjust the preprocessing data set
`adjust2=`*SAS-statements*	includes statements with PROC PLOT
`adjust3=`*SAS-statements*	extra statements for the final DATA step
`adjust4=`*SAS-statements*	extra statements for the final DATA step
`adjust5=`*SAS-statements*	extra statements for the final DATA step
`antiidea=`*n*	eliminates PREFMAP anti-ideal points
`blue=`*expression*	blue part of RGB colors
`bright=`*n*	generates random label colors
`britypes=`*type*	types to which `bright=` applies
`cframe=`*color*	color of background within the frame
`cirsegs=`*n*	circle smoothness parameter
`color=`*color*	default color
`colors=`*colors-list*	default label and symbol color list
`cursegs=`*n*	number of segments in a curve
`curvecol=`*color*	color of curve
`data=`*SAS-data-set*	input data set
`datatype=`*data-type*	data analysis that generated data set
`debug=`*values*	debugging output
`excolors=`*color-list*	excludes from the Annotate data set
`extend=`*axis-extensions*	extend the x and y axes
`extraobs=`*SAS-data-set*	extra observations data set
`exttypes=`*type*	types for `extraobs=` data set
`filepref=`*prefix*	file name prefix
`font=`*font*	default font
`framecol=`*color*	color of frame
`gdesc=`*description*	catalog description
`gname=`*name*	catalog entry
`gopplot=`*goptions*	`goptions` for plotting to screen
`gopprint=`*goptions*	`goptions` for printing
`gopts2=`*goptions*	`goptions` that are always used
`gopts=`*goptions*	additional `goptions`
`gout=`*catalog*	`proc ganno gout=` catalog
`green=`*expression*	green part of RGB colors
`hminor=`*n* | *do-list*	horizontal axis minor tick marks
`hnobs=`*n*	horizontal observations for contour plots
`hpos=`*n*	horizontal positions in graphics area
`href=`*do-list*	horizontal reference lines
`hsize=`*n*	horizontal graphics area size
`inc=`*n*	`haxis=by inc`, `vaxis=by inc`
`interpol=`*method*	axis interpolation method
`labcol=`*label-colors*	colors for the point labels
`label=`*label-statement*	`label` statement
`labelcol=`*color*	color of variable labels

Option	Description
`labelvar=`*label-variable*	point label variable
`labfont=`*label-fonts*	fonts for the point labels
`labsize=`*label-sizes*	sizes for the point labels
`ls=`*n*	how line sizes are generated
`lsinc=`*n*	increment to line size
`lsizes=`*number-list*	line sizes (thicknesses)
`makefit=`*n*	proportion of graphics window to use
`maxiter=`*n*	maximum number of iterations
`maxokpen=`*n*	maximum acceptable penalty sum
`method=`*value*	where to send the plot
`monochro=`*color*	overrides all other colors
`nknots=`*n*	number of knots option
`offset=`*n*	move symbols for coincident points
`options=border`	draws a border box
`options=close`	close up the border box and the axes
`options=diag`	draws a diagonal reference line
`options=expand`	expands the plot to fill the window
`options=noback`	do not set the frame color
`options=nocenter`	do not center
`options=noclip`	do not clip
`options=nocode`	suppress the PROC PLOT and `goptions` statements
`options=nodelete`	do not delete intermediate data sets
`options=nohistory`	suppress the iteration history table
`options=nolegend`	suppress the display of the legends
`options=noprint`	`nolegend`, `nocode`, and `nohistory`
`options=square`	the same ticks for both axes and a square plot
`options=textline`	lines overwrite text
`out=`*SAS-data-set*	output Annotate data set
`outward=none` | *char*	PLOT statement `outward=`
`paint=`*interpolation*	color interpolation
`place=`*placement*	generates a `placement=` option
`plotopts=`*options*	PLOT statement options
`plotvars=`*variable-list*	y-axis and x-axis variables
`post=`*filename*	graphics stream file name
`preproc=`*SAS-data-set*	preprocessed `data=` data set
`procopts=`*options*	PROC PLOT statement options
`ps=`*n*	page size
`radii=`*do-list*	radii of circles
`red=`*expression*	red part of RGB colors
`regdat=`*SAS-data-set*	intermediate regression results data set
`regopts=`*options*	regression curve fitting options
`regprint=`*regression-option*	regression options
`rgbround=`*RGB-rounding*	`paint=` rounding factors
`rgbtypes=`*type*	types to which `paint=` and RGB options apply
`style=`A | B	obsolete option not supported
`symbols=`*symbol-list*	plotting symbols
`symcol=`*symbol-colors*	colors of the symbols
`symfont=`*symbol fonts*	symbol fonts

Option	Description
`symlen=`*n*	length of the symbols
`symsize=`*symbol-sizes*	sizes of symbols
`symtype=`*symbol-types*	types of symbols
`symvar=`*symbol-variable*	plotting symbol variable
`tempdat1=`*SAS-data-set*	intermediate results data set
`tempdat2=`*SAS-data-set*	intermediate results data set
`tempdat3=`*SAS-data-set*	intermediate results data set
`tempdat4=`*SAS-data-set*	intermediate results data set
`tempdat5=`*SAS-data-set*	intermediate results data set
`tempdat6=`*SAS-data-set*	intermediate results data set
`tickaxes=`*axis-string*	axes to draw tick marks
`tickcol=`*color*	color of ticks
`tickfor=`*format*	tick format used by `interpol=tick`
`ticklen=`*n*	length of tick mark in horizontal cells
`titlecol=`*color*	color of title
`tsize=`*n*	default text size
`types=`*observation-types*	observations types
`typevar=`*variable*	observation types variable
`unit=in` | `cm`	`hsize=` and `vsize=` unit
`vechead=`*vector-head-size*	how to draw vector heads
`vminor=`*n* | *do-list*	vertical axis minor tick marks
`vnobs=`*n*	vertical observations for contour plots
`vpos=`*n*	vertical positions in graphics area
`vref=`*do-list*	vertical reference lines
`vsize=`*n*	vertical graphics area size
`vtoh=`*n*	PROC PLOT `vtoh=` option
`xmax=`*n*	maximum horizontal graphics area size
`ymax=`*n*	maximum vertical graphics area size

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%plotit(help)
%plotit(?)
```

Note that for many analyses, the only options you need to specify are `data=`, `datatype=`, and sometimes `method=`. To specify variables to plot, specify `plotvars=`, `labelvar=`, and `symvar=`.

Overriding Options

This macro looks for a special global macro variable named `plotitop`. If it exists, its values are used to override the macro options. If you have a series of calls to the plotting macro and say, for example, that you want to route each graph to a PDF file, you can specify the following statement once:

```
%let plotitop = gopts=dev=pdf, method=print;
```

Then you can run the macro repeatedly without change. The value of the `plotitop` macro variable must consist of a name, followed by an equal sign, followed by a value. Optionally, it can continue with a comma, followed by another `name=`*value*, and so on, just like the way options are specified with the macro. Option values must not contain commas.

Destination and GOPTIONS

The options in this section specify the plot destination and SAS `goptions`. Note that with the `%PlotIt` macro, you do not specify a `goptions` statement. If you do, it will be overridden. All `goptions` (except `device=`) are specified with macro options. If you would prefer to specify your own `goptions` statement and have the macro use it, just specify or change the default for these four options to null: `gopplot=`, `gopprint=`, `gopts2=`, `gopts=`. If you use a locally installed copy of the macro, you can modify the `gopprint=` and `gopplot=` options defaults to include the devices that you typically use. Otherwise, the macro checks the `goptions` to get a device.

gopplot= *goptions*
specifies the `goptions` for directly plotting on the screen. There are no default goptions for `gopplot=`.

gopprint= *goptions*
specifies the `goptions` for printing (creating a graphics stream file). The default is `gopprint=gsfmode=replace gaccess=gsasfile gsfname=gsasfile`.

The following options show how you might modify the defaults for `gopprint=` and `gopplot=` option defaults to set default devices:

```
gopprint=gsfmode=replace gaccess=gsasfile gsfname=gsasfile device=png,
gopplot=cback=black device=win,
```

gopts= *goptions*
provides a way to specify additional `goptions` that are always used. There are no default goptions for `gopts=`. For example, to rotate to a landscape orientation with a black background color, specify `gopts=rotate cback=black`.

gopts2= *goptions*
specifies the `goptions` that are always used, no matter which `method=` is specified. The default is `gopts2=reset=goptions erase`.

method= `gplot` | `plot` | `print` | `none`
specifies where to send the plot. The default is `method=gplot`.

`gplot` – displays a graphical scatter plot on your screen using the `goptions` from `gopplot=`. The `gopplot=` option should contain the `goptions` that only apply to plots displayed on the screen.

`plot` – creates a printer plot only.

`print` – routes the plot to a graphics stream file, such as a postscript file, using the `goptions` from `gopprint=`. The `gopprint=` option should contain the `goptions` that only apply to hard-copy plots. Specify the file name with `post=`.

`none` – just creates the Annotate data set and sets up `goptions` using `gopplot=`.

Data Set and Catalog Options

These options specify the input SAS data set, output Annotate data set, and options for writing plots to files and catalogs.

data= *SAS-data-set*

specifies the input data set. The default input data set is the last data set created. You should always specify the `data=` option since the macro creates data sets that are not suitable for use as input.

filepref= *file-name-prefix*

specifies the file name prefix. The default is `filepref=sasplot`.

gdesc= *description*

specifies the name of a catalog description. This option can optionally be used with `proc ganno gout=` to provide the `description=`.

gname= *name*

specifies the name of a catalog entry. This option can optionally be used with `proc ganno gout=` to provide the `name=`.

gout= *catalog*

specifies the `proc ganno gout=` catalog. With `gout=gc.slides`, first specify: `libname gc '.'`; Then to replay, run: `proc greplay igout=gc.slides; run;` Note that replayed plots will not in general have the correct aspect ratio.

out= *SAS-data-set*

specifies the output Annotate data set. This data set contains all of the instructions for drawing the graph. The default is `out=anno`.

post= *filename*

specifies the graphics stream file name. The default name is constructed from the `filepref=` value and '`ps`' in a host-specific way.

Typical Options

These are some of the most frequently used options.

datatype= *data-type*

specifies the type of data analysis that generated the data set. This option is used to set defaults for other options and to do some preprocessing of the data.

When the data type is `corresp`, `mds`, `mca`, `row`, `column`, `mdpref`, `mdpref2`, `vector`, or `ideal`, the `label=typical` option is implied when `label=` is not otherwise specified. The default point label variable is the last character variable in the data set.

Some data types (`mdpref`, `vector`, `ideal`, `corresp`, `row`, `mca`, `column`, `mds`) expect certain observation types and set the `types=` list accordingly. For example, `mdpref` expects `_type_ = 'SCORE'` and `_type_ = 'CORR'` observations. The remaining data types do not expect any specific value of the `typevar=` variable. So if you do not get the right data types associated with the right observation types, specify `types=`, and specify the `types=` values in an order that corresponds to the order of the symbol types in the `Types Legend` table. Unlike `symtype=`, the order in which you specify `datatype=` values is irrelevant.

A null value (`datatype=`, the default) specifies no special processing, and the default plotting variables are the first two numeric variables in the data set. Specifying `corresp`, `mds`, `mca`, `row`, or `column` will set the default `plotvars` to `dim2` and `dim1`. Otherwise, when a nonnull value is specified, the default `plotvars` are `prin2` and `prin1`.

The various data types are as follows:

`datatype=column`
specifies a `proc corresp profile=column` analysis. Row points are plotted as vectors.

`datatype=contour`
draws solid color contour plots. When the number of row points is not the same as the number of column points in the grid, use `hnobs=` and `vnobs=` to specify the number of points. This method creates an `hnobs=` by `vnobs=` grid of colored rectangles. Each of the rectangles should touch all adjacent rectangles. This method works well with a regular grid of points. The `method=square` option is a good alternative when the data do not fall in a regular grid.

`datatype=corresp`
specifies an ordinary correspondence analysis.

`datatype=curve`
fits a curve through the scatter plot.

`datatype=curve2`
fits a curve through the scatter plot and tries to make the labels avoid overprinting the curve.

`datatype=function`
draws functions. Typically, no labels or symbols are drawn. This option has a similar effect to the PROC GPLOT `symbol` statement options `i=join v=none`.

`datatype=ideal`
specifies a PREFMAP ideal point model. See the `antiidea=`, `radii=`, and `cirsegs=` options.

`datatype=mca`
specifies a multiple correspondence analysis.

`datatype=mdpref`
specifies multidimensional preference analysis with vectors with blank labels. Note that `datatype=mdpref` can also be used for ordinary principal component analysis.

`datatype=mdpref2`
specifies MDPREF with vector labels (MDPREF and labels *too*).

`datatype=mds`
specifies multidimensional scaling.

`datatype=mds ideal`
specifies PREFMAP ideal point after the MDS.

`datatype=mds vector`
specifies PREFMAP after MDS.

`datatype=row`
specifies a `proc corresp profile=row` analysis. Column points are plotted as vectors.

`datatype=square`
plots each point as a solid square. The `datatype=square` option is useful as a form of contour plotting when the data do not form a regular grid. The `datatype=square` option, unlike `datatype=contour`, does not try to scale the size of the square so that each square will touch another square.

`datatype=symbol`
specifies an ordinary scatter plot.

`datatype=vector`
specifies a PREFMAP vector model.

`datatype=vector ideal`
specifies both PREFMAP vectors and ideal points.

For some `datatype=` values, a number can be specified after the name. This is primarily useful for biplot data sets produced by PROC PRINQUAL and PREFMAP data sets produced by PROC TRANSREG. This number specifies that the lengths of vectors should be changed by this amount. The number must be specified last. Examples: `datatype=mdpref 2`, `datatype=mds vector 1.5`.

The primary purpose of the `datatype=` option is to provide an easy mechanism for specifying defaults for the options in the next section (`typevar=` through `outward=`).

labelvar= *label-variable* | `_blank_`

specifies the variable that contains the point labels. The default is the last character variable in the data set. If `labelvar=_blank_` is specified, the macro will create a blank label variable.

options= *options-list*

specifies binary options. Specify zero, one, or more in any order. For example: `options=nocenter nolegend`.

`border`
`noborder`

draws a border box around the outside of the graphics area, like the border `goption`. The default depends on the style, and with typical styles, there is a border.

`close`
if a border is being drawn, perform the same adjustments on the border that are performed on the axes. This option is most useful with contour plots.

`diag`
draws a diagonal reference line.

`expand`
specifies Annotate data set post processing, typically for use with `extend=close` and contour plots. This option makes the plot bigger to fill up more of the window.

`noback`
specifies that `%PlotIt` should not set the frame color (the background color within the plot boundary).

`nocenter`
do not center. By default, when `nocenter` is not specified, `vsize=` and `hsize=` are set to their maximum values, and the `vpos=` and `hpos=` values are increased accordingly. The x and y coordinates are increased to positioning the plot in the center of the full graphics area.

`noclip`
do not clip. By default, when `noclip` is not specified, labels that extend past the edges of the plot are clipped. This option will not absolutely prevent labels from extending beyond the plot, particularly when sizes are greater than 1.

`nocode`
suppresses the display of the PROC PLOT and `goptions` statements.

`nodelete`
do not delete intermediate data sets.

`nohistory`
suppresses the display of the iteration history table.

`nolegend`
suppresses the display of the legends.

`noprint`
equivalent to `nolegend`, `nocode`, and `nohistory`.

`square`
uses the same ticks for both axes and tries to make the plot square by tinkering with the `extend=` option. Otherwise, ticks might be different.

`textline`
put text in the data set, followed by lines, so lines overwrite text. Otherwise text overwrites lines.

plotvars= *two-variable-names*

specifies the *y*-axis variable then the *x*-axis variable. To plot `dim2` and `dim3`, specify `plotvars=dim2 dim3`. The `datatype=` option controls the default variable list.

symlen= *n*

specifies the length of the symbols. By default, symbols are single characters, but the macro can center longer strings at the symbol location.

symvar= *symbol-variable* | `_symbol_`

specifies the variable that contains the plotting symbol for input to PROC PLOT. When `_symbol_` is specified, which is the default, the symbol variable is created, typically from the `symbols=` list, which might be constructed inside the macro. (Note that the variable `__symbol` is created to contain the symbol for the graphical scatter plot. The variables `_symbol_` and `__symbol` might or might not contain the same values.) Variables can be specified, and the first `symlen=` characters are used for the symbol. When a null value (`symvar=`) or a constant value is specified, the symbol from the printer plot is used (which is always length one, no matter what is specified for `symlen=`). To get PROC PLOT pointer symbols, specify `symvar='00'x`, (hex null constant). To center labels with no symbols, specify: `symbols='', place=0`.

Observation-Type List Options

Data sets for plotting can have different types of observations that are plotted differently. These options let you specify the types of observations, the variable that contains the observation types, and the different ways the different types should be plotted. For many types of analyses, these can all be handled easily with the `datatype=` option, which sets analysis-specific defaults for the list options. When you can, you should use `datatype=` instead of the list options. If you do use the list options, specify a variable, in `typevar=`, whose values distinguish the observation types. Specify the list of valid types in `types=`. Then specify fonts, sizes, and so on for the various observation types. Alternatively, you can use these options with `datatype=`. Specify lists for just those label or symbol characteristics you want to change, for example, colors, fonts or sizes.

The lists do not all have to have the same number of elements. The number of elements in `types=` determines how many of the other list elements are used. When an observation type does not match one of the `type=` values, the results are the same as if the first type is matched. If one of the other lists is shorter than the `types=` list, the shorter list is extended by adding copies of the last element to the end. Individual list values can be quoted, but quotes are not required. *Embedded blanks are not permitted. If you embed blanks, you will not get the right results.* Values of the `typevar=` variable are compressed before they are used, so for example, an `_type_` value of `'M COEFFI'` must be specified as `'MCOEFFI'`.

britypes= *type*

specifies the types to which `bright=` applies. The default is `britypes=symbol`.

colors= *colors-list*

specifies the default color list for the `symcol=` and `labcol=` options. The default depends on the style. With `gstyle`, this list is ignored and the list comes from the style. Otherwise, the default is `colors=blue red green cyan magenta orange gold lilac olive purple brown gray rose violet salmon yellow`. This is the old legacy default. Colors such as these from the default ODS style `colors=CX2A25D9 CXB2182B CX01665E CX9D3CDB CX543005 CX2597FA CX7F8E1F CXB26084 CXD17800 CX47A82A CXB38EF3 CXF9DA04` tend to look much better than the default colors. (See page 1204 for information about CX*rrggbb* color specifications.)

exttypes= *type*

specifies the types to always put in the `extraobs=` data set when they have blank labels. The default is `exttypes=vector`.

labcol= *label-colors*

specifies the colors for the point labels. The default either comes from the style or from `colors=` with `nogstyle`. This option is ignored with `gstyle`. Examples (none means hardware):

```
labcol='red'
labcol='red' 'white' 'blue'
```

labfont= *label-fonts*

specifies the fonts for the point labels. Examples (none means hardware):

```
labfont=none
labfont='swiss'
labfont='swiss' 'swissi'
```

labsize= *label-sizes*

specifies the sizes for the point labels. Examples:

```
labsize=1
labsize=1 1.5
labsize=1 0
```

rgbtypes= *type*

specifies the types to which `paint=`, `red=`, `green=`, and `blue=` apply. The default is `rgbtypes=symbol`.

symbols= *symbol-list*

specifies the plotting symbols. Symbols can be more than a single character. You must specify `symlen=n` for longer lengths. Blank symbols must be specified as `''` with no embedded blanks. Examples:

```
symbols='*'
symbols='**'
symbols='*' '+' '*' ''
symbols='NC' 'OH' 'NJ' 'NY'
```

symcol= *symbol-colors*

specifies the colors of the symbols. The default list is constructed from the `colors=` option. Examples:

```
symcol='red'
symcol='red' 'white' 'blue'
```

symtype= *symbol-types*

specifies the types of symbols. Valid types include: `symbol`, `vector`, `circle`, `contour`, and `square`. Examples:

```
symtype='symbol'
symtype='symbol' 'vector'
symtype='symbol' 'circle'
```

symfont= *symbol fonts*

specifies the symbol fonts. The font is ignored for vectors with no symbols. Examples (none means hardware):

```
symfont=none
symfont='swiss'
symfont='swiss' 'swissi'
```

symsize= *symbol-sizes*

specifies the sizes of symbols. Examples:

```
symsize=1
symsize=1 1.5
```

types= *observation-types*

specifies the observations types. Observation types are usually values of a variable like `_type_`. Embedded blanks are not permitted. Examples:

```
types='SCORE'
types='OBS' 'SUPOBS' 'VAR' 'SUPVAR'
types='SCORE'
types='SCORE' 'MCOEFFI'
```

The order in which values are specified for the other options depends on the order of the types. The default types for various `datatype=` values are given next:

```
corresp:  'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
row:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mca:      'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
column:   'VAR' 'OBS' 'SUPVAR' 'SUPOBS'
mdpref:   'SCORE' 'CORR'
vector:   'SCORE' 'MCOEFFI'
ideal:    'SCORE' 'MPOINT'
mds:      'SCORE' 'CONFIG'
```

For combinations of options, these lists are combined in order, but without repeating `'SCORE'`, for example, with `datatype=mdpref vector ideal`, the default `types=` list is: `'SCORE' 'CORR' 'MCOEFFI' 'MPOINT'`.

typevar= *variable*

specifies a variable that is looked at for the observation types. By default, this is `_type_` if it is in the input data set.

Internal Data Set Options

The macro creates one or more of these data sets internally to store intermediate results.

extraobs= *SAS-data-set*

specifies a data set used to contain the extra observations that do not go through PROC PLOT. The default is `extraobs=extraobs`.

preproc= *SAS-data-set*

specifies a data set used to contain the preprocessed `data=` data set. The default is `preproc=preproc`.

regdat= *SAS-data-set*

specifies a data set used to contain intermediate regression results for curve fitting. The default is `regdat=regdat`.

tempdat1= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat1=tempdat1`.

tempdat2= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat2=tempdat2`.

tempdat3= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat3=tempdat3`.

tempdat4= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat4=tempdat4`.

tempdat5= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat5=tempdat5`.

tempdat6= *SAS-data-set*

specifies a data set used to hold intermediate results. The default is `tempdat6=tempdat6`.

Miscellaneous Options

The following options are sometimes needed for certain situations to control the details of the plots:

antiidea= *n*

eliminates PREFMAP anti-ideal points. The TRANSREG ideal-point model assumes that small attribute ratings mean that the object is similar to the attribute and large ratings imply dissimilarity to the attribute. For example, if the objects are food and the attribute is “sweetness,” then the analysis assumes that 1 means sweet and 9 is much less sweet. The resulting coordinates are usually ideal points, representing an ideal amount of the attribute, but sometimes they are anti-ideal points and need to be converted to ideal points. This option is used to specify the nature of the data (small ratings mean similar or dissimilar) and to request automatic conversion of anti-ideal points.

null value – (`antiidea=`, the default) – do nothing.

1 – reverses in observations whose `_TYPE_` contains 'POINT' when `_issq_` > 0. Specify `antiidea=1` with `datatype=ideal` for the unusual case when large data values are positive or ideal.

–1 – reverses in observations whose `_type_` contains 'POINT' when `_issq_` < 0. Specify `antiidea=-1` with `datatype=ideal` for the typical case when small data values are positive or ideal.

extend= *axis-extensions*

is used to extend the x and y axes beyond their default lengths. Specify four values, for the left, right, top, and bottom axes. If the word `close` is specified somewhere in the string, then macro moves the axes in close to the extreme data values, and the computed values are added to the specified values (if any). Sample specifications: `extend=2 2`, or `extend=3 3 -0.5 0.5`. Specifying a positive value n extends the axis n positions in the indicated direction. Specifying a negative value shrinks the axis. The defaults are in the range –2 to 2, and are chosen in an attempt to add a little extra horizontal space and make equal the extra space next to each of the four extreme ticks. When there is enough space, the horizontal axis is slightly extended by default to decrease the chance of a label slightly extending outside the plot. PROC PLOT usually puts one or two more lines on the top of the plot than in the bottom. The macro tries to eliminate this discrepancy. This option does not add any tick marks; it just extends or shrinks the ends of the axis lines. So typically, only small values should be specified. Be careful with this option and a positive `makefit=` value.

font= *font*

specifies the default font. With `nogstyle`, the default is `font=swiss`. Otherwise the font comes from the style. The annotate data set will often contain fonts of `none`, which means that a hardware font is used. This will usually look much better than a software font such as the Swiss fonts that were heavily used before the 9.2 release.

hminor= *n* | *do-list*

specifies the number of horizontal axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `haxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `hminor=0.25 to 5 by 0.25`, with data ranging up to 10^5.

href= *do-list*

specifies horizontal-axis reference lines (which are drawn vertically). Specify a DATA step `do` list. By default, there are no reference lines.

inc= *n*

specifies `haxis=by inc` and `vaxis=by inc` values. The specified increments apply to both axes. To individually control the increments, you must specify the PLOT statement `haxis=` and `vaxis=` options on the `plotopts=` option. When you are plotting variables that have very different scales, you might need to independently specify appropriate tick increments for both axes to get a reasonable plot, like this for example: `plotopts=haxis=by 20 vaxis=by 5000`.

interpol= `ls` | `tick` | `no` | `hlog` | `vlog` | `yes`

specifies the axis interpolation method.

> `ls` – uses the least-squares method only. This method computes the mapping between data and positions using ordinary least-squares linear regression. Usually, you should not specify `interpol=ls` because slight inaccuracies can result, producing aesthetically unappealing plots.
>
> `hlog` – specifies that the x-axis is on a log scale.
>
> `no` – does not interpolate.
>
> `tick` – uses the tick mark method. This method computes the slope and intercept using tick marks and their values. Tick marks are read using the `tickfor=` format.
>
> `vlog` – specifies that the y-axis is on a log scale.
>
> `yes` – the default, interpolates symbol locations, starting with least squares but replacing them with tick-based estimates when they are available.

This option makes the symbols, vectors, and circles map to the location they would in a true graphical scatter plot, not the cell locations from PROC PLOT. This option has no effect on labels, the frame, reference lines, titles, or ticks. With `interpol=no`, plots tend to look nicer whereas `interpol=yes` plots are slightly more accurate. Note that the strategy used to interpolate can be defeated in certain cases. If the horizontal axis tick values are displayed vertically, specify `interpol=ls`. The `hlog` and `vlog` values are specified in addition to the method. For example, `interpol=yes vlog hlog`.

label= *label-statement*

specifies a `label` statement. Note that specifying the keyword `label` to begin the statement is optional. You can specify `label=typical` to request a label statement constructed with `'Dimension'` and the numeric suffix of the variable name, for example, `label dim1 = 'Dimension 1' dim2 = 'Dimension 2';` when `plotvars=dim2 dim1`. The `label=typical` option can only be used with variable names that consist of a prefix and a numeric suffix.

ls= *n* | *iterative-specification*

specifies how line sizes are generated. The default is `ls=compute search`. When the second word is `search`, the macro searches for an optimal line size. See the `place=` option for more information about searches. When the first word is `compute`, the line size is computed from the iteration number so that the line sizes are: 65 80 100 125 150 175 200. Otherwise the first word is the first linesize and with each iteration the linesize is incremented by the `lsinc=` amount. Example: `ls=65 search`.

lsinc= *n*

specifies the increment to line size in iterations when line size is not computed. The default is `lsinc=15`.

lsizes= *number-list*

specifies the line sizes (thicknesses) for frame, ticks, vectors, circles, curves, respectively. The default is `lsizes=1 1 1 1 1`.

maxiter= *n*

specifies the maximum number of iterations. The default is `maxiter=15`.

maxokpen= *n*

specifies the maximum acceptable penalty sum. The default is `maxokpen=0`. Penalties accrue when label characters collide, labels get moved too far from their symbols, or words get split.

offset= *n*

move symbols for coincident points `offset=` spaces up/down and left/right. This helps to better display coincident symbols. Specify a null value (`offset=,`) to turn off offsetting. The default is `offset=0.25`.

place= *placement-specification*

generates a `placement=` option for the plot request. The default is `place=2 search`. Specify a non-negative integer. Values greater than 13 are set to 13. As the value gets larger, the procedure is given more freedom to move the labels farther from the symbols. The generated placement list is displayed in the log. You can still specify `placement=` directly in the `plotopts=` option. This option just gives you a shorthand notation. The following list shows the correspondence between these two options:

```
place=0 --placement=((s=center))


place=1 --placement=((h=2 -2 : s=right left)
                     (v=1 * h=0 -1 to -2 by alt))


place=2 --placement=((h=2 -2 : s=right left)
                     (v=1 -1 * h=0 -1 to -5 by alt))


place=3 --placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt))


place=4 --placement=((h=2 -2 : s=right left)
                     (v=1 to 2 by alt * h=0 -1 to -10 by alt)
                     (s=center right left * v=0 1 to 2 by alt *
                      h=0 -1 to -6 by alt * l= 1 to 2))
```

and so on.

The `place=` option, along with the `ls=` option can be used to search for an optimal placement list and an optimal line size. By default, the macro will create and recreate the plot until it avoids all collisions.

The search is turned off when a `placement=` option is detected in the plot request or plot options.

If search is not specified with `place=` or `ls=`, the specified value is fixed. If search is specified with the other option, only that option's value is incremented in the search.

plotopts= *options*

specifies PLOT statement options. The `box` option is specified, even if you do not specify it. Reference lines should not be specified using the PROC PLOT `href=` and `vref=` options. Instead, they should be specified directly using the `href=` and `vref=` macro options. By default, no PLOT statement options are specified except `box`.

procopts= *options*

specifies PROC PLOT statement options. The default is `procopts=nolegend`.

tickaxes= *axis-string*

specifies the axes to draw tick marks. The default with `nogstyle` is `tickaxes=LRTBFlb`, and with a style, the default is `tickaxes=LBF`, which provides a cleaner look than the old default, more like ODS Graphics. The specification, `tickaxes=LRTBFlb`, means major ticks on left (L), right (R), top (T), and bottom (B), and the full frame (F) is to be drawn, and potentially minor tick marks on the left (l) and bottom (b). Minor ticks on the right (r) and top (t) can also be requested. To just have major tick marks on the left and bottom axes, and no full frame, specify `tickaxes=LB`. Order and spacing do not matter. `hminor=` and `vminor=` must also be specified to get minor ticks.

tickfor= *format*

specifies the tick format used by `interpol=tick`. You should change this if the tick values in the PROC PLOT output cannot be read with the default `tickfor=32.` format. For example, specify `tickfor=date7.` with dates.

ticklen= *n*

specifies the length of tick marks in horizontal cells. A negative value can be specified to indicate that only half ticks should be used, which means the ticks run to but not across the axes. The default is `ticklen=1.5`.

tsize= *n*

specifies the default text size. The default is `tsize=1`.

vminor= *n* | *do-list*

specifies the number of vertical axis minor tick marks between major tick marks. A typical value is 9. The number cannot be specified when `vaxis=` is specified with `plotopts=`. Alternatively, specify a DATA step `do` list. Note that with log scaling, specify $\log_{10}$'s of the data values. For example, specify `vminor=0.25 to 5 by 0.25`, with data ranging up to 10^5.

vref= *do-list*
specifies vertical reference lines (which are drawn horizontally). Specify a DATA step `do` list. By default, there are no reference lines.

Color Options

With `gstyle`, the symbol and point label colors are set by the style. Otherwise, the symbol and point label colors are set by the `labcol=` and `symcol=` options. The other color options are as follows:

bright= *n*
generates random label colors for `britypes=` values. In congested plots, it might be easier to see which labels and symbols go together if each label/symbol pair has a different random color. Colors are computed so that the mean RGB (red, green, blue) components equal the specified `bright=` value. The valid range is $5 \leq bright \leq 250$. 128 is a good value. Small values will produce essentially black labels and large values will produce essentially white labels, and so should be avoided. The default is a null value, `bright=`, and there are no random label colors. If you get a color table full error message, you need to specify larger values for the `rgbround=` option.

cframe= *color*
specifies the color of the background within the frame. This is analogous to the `cframe=` SAS/GRAPH option. With `gstyle`, the style color is used. This option is ignored with `gstyle`.

color= *color*
specifies the default color that is used when no other color is set. The default color is black. With `gstyle`, this option will typically have little or no effect.

curvecol= *color*
specifies the color of curves in a regression plot. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

excolors= *color-list*
excludes observations from the Annotate data set with colors in this list. For example, with a white background, to exclude all observations that have a color set to white as well as those with a computed white color, for example, from `bright=` or `paint=`, specify `excolors=white CXFFFFFF`. This is done for efficiency, to make the Annotate data set smaller. (See the `paint=` option, page 1204 for information about CX*rrggbb* color specifications.)

framecol= *color*
specifies the color of the frame. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

labelcol= *color*
specifies the color of the variable labels. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

monochro= *color*
overrides all other specified colors. By default when `monochro=` is null, this option has no effect. Typical usage: `monochro=black`. Typically, you would rarely if ever use this option, and you would only use it with `nogstyle`.

style= this option is obsolete and is no longer supported. Use OPTIONS `gstyle/nogstyle` and a `style=` specification in your ODS destination to control the style. The `nogstyle` option will make the macro run the way it did in older releases. The `gstyle` option will make the appearance sensitive to the ODS style. The new results are usually superior. However, if you want explicit color control, you will typically need to specify `nogstyle`.

tickcol= *color*
specifies the color of tick labels. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

titlecol= *color*
specifies the color of the title. The default either comes from the style or from `color=` with `nogstyle`. This option is ignored with `gstyle`.

Color Interpolation and Painting

These next options are used to create label and symbol colors using some function of the input data set variables. For example, you can plot the first two principal components on the x and y axes and show the third principal component in the same plot by using it to control the label colors. The `paint=` option gives you a simple and fairly general way to interpolate colors. The `red=`, `green=`, and `blue=` options are used together for many other types of interpolations, but these options are much harder to use. These options apply to `rgbtypes=` observations. If `red=`, `green=`, and `blue=` are not flexible enough, for example, if you need full statements, specify `red=128` (so later code will know you are computing colors) then insert the full statements you need to generate the colors using `adjust1=`.

paint= *color-interpolation-specification*
is used to interpolate between colors based on the values of a variable. The simplest specification is `paint=variable`. More generally, specify the following option:

```
paint=variable optional-color-list optional-data-value-list
```

The following color names are recognized: red, green, blue, yellow, magenta, cyan, black, white, orange, brown, gray, olive, pink, purple, violet. For other colors, specify the RGB color name. The `paint=` option applies to the observation types mentioned in the `rgbtypes=` option. Valid types include: `symbol`, `vector`, `circle`, `contour`, and `square`.

Colors can be represented as `CX`*rrggbb* where *rr* is the red value, *gg* is the green, and *bb* is blue, all three specified in hex. The base ten numbers 0 to 255 map to 00 to FF in hex. For example, white is `CXFFFFFF` (all colors at their maximum), black is `CX000000` (all colors at their minimum), red is `CXFF0000` (maximum red, minimum green and blue), blue is `CX0000FF` (maximum blue, minimum red and green), and magenta is `CXFF00FF` (maximum red and blue, minimum green). When a variable named `z` is specified with no other arguments, the default is `paint=z blue magenta red`. The option `paint=z red green 1 10` interpolates between red and green, based on the values of the variable `z`, where values of 1 or less map to red, values of 10 or more map to green, and values in between map to colors in between. The specification `paint=z red yellow green 1 5 10`, interpolates between red at `z=1`, yellow at `Z=5`, and green at `Z=10`. If the data value list is omitted, it is computed from the data.

red= *expression*
green= *expression*
blue= *expression*
specify for arithmetic expressions that produce integers in the range 0 to 255. Colors are created as follows:

```
__color = 'CX' ||
          put(%if &red   ne %then round(&red,  __roured); %else 128; ,hex2.) ||
          put(%if &green ne %then round(&green,__rougre); %else 128; ,hex2.) ||
          put(%if &blue  ne %then round(&blue, __roublu); %else 128; ,hex2.);}
```

The `__rou` variables are extracted from the second through fourth values of the `rgbround=` option. Example: `red = min(100 + (z - 10) * 3, 255)`, `blue=50`, `green=50`. Then all labels are various shades of red, depending on the value of `z`. Be aware that light colors (small red-green-blue values) do not show up well on white backgrounds and dark colors do not show up well on dark backgrounds. Typically, you will not want to use the full range of possible red-green-blue values. Computed values greater than 255 are set to 255.

rgbround= *RGB-rounding-specification*
specifies rounding factors used for the `paint=` variable and RGB values. The default is `rgbround=-240 1 1 1`. The first value is used to round the `paint=` variable. Specify a positive value to have the variable rounded to multiples of that value. Specify a negative value $-n$ to have a maximum of n colors. For the other three values, specify positive values. The last three are rounding factors used to round the values for the red, green, and blue component of the color (see `red=`). If more than 256 colors are generated, you will get the error that a color was not added because the color table is full. By default, when a value is missing, there is no rounding. Rounding the `paint=` variable is useful with contour plots.

Contour Options

Use these options with contour plots. For example, the grid for a contour plot might be generated as follows:

```
do x = -4 to 4 by 0.1;
   do y = -2 to 2 by 0.1;
      ... statements ...
      end;
   end;
```

Horizontally, there are $1+(4--4)/0.1 = 81$ horizontal points and $1+(2--2)/0.1 = 41$ vertical points, so you would specify `hnobs=81`, `vnobs=41`. By default, the square root of the number of contour type observations is used for both `hnobs=` and `vnobs=` (which assumes a square grid).

hnobs= *n*
specifies the number of horizontal observations in the grid for contour plots.

vnobs= *n*
specifies the number of vertical observations in the grid for contour plots.

Advanced Plot Control Options

You can use the these next options to add full SAS DATA step statements to strategic places in the macro, such as the PROC PLOT step, the end of the preprocessing, and last full DATA steps. These options do minor adjustments before the final plot is produced. These options allow very powerful customization of your results to an extent not typically found in procedures. However, they might require a fair amount of work and some trial and error to understand and get right.

adjust1= *SAS-statements*
specifies preprocessing SAS statements. The following variables are created in the preprocessing data set:

`__lsize` – label size
`__lfont` – label font
`__lcolor` – label color
`__ssize` – symbol size
`__sfont` – symbol font
`__scolor` – symbol color
`__stype` – symbol type
`__symbol` – symbol value
`__otype` – observation type

Use `adjust1=` to adjust these variables in the preprocessing data set. You must specify complete statements with semicolons, for example, as follows:

```
adjust1=%str(__lsize = 1.2; __lcolor = green;)

adjust1=%str(if z > 20 then do;
__scolor = 'green'; __lcolor = 'green'; end;)
```

adjust2= *SAS-statements*
specifies statements with PROC PLOT such as `format` statements. Just specify the full statement.

adjust3= *SAS-statements*
adjust4= *SAS-statements*
specify options to adjust the final Annotate data set. For example, in Swiss fonts, asterisks are not vertically centered when displayed, so `adjust3=` converts to use the SYMBOL function, so by default, `adjust3=%str(if text = '*' and function = 'LABEL' then do; style = ' '; text = 'star'; function = 'SYMBOL'; end;)`. The default for `adjust4=` is null, so you can use it to add new statements. If you add new variables to the data set, you must also include a `keep` statement.

The following illustrates using `adjust4=` to vertically display the y-axis label, like it would in PROC PLOT:

```
adjust4=%str(if angle = 90 then do; angle = 270; rotate = 90; keep rotate; end;)
```

The following option changes the size of title lines:

```
adjust4=%str(if index(comment, 'title') then size = 2;)
```

adjust5= *SAS-statements*

adds extra statements to the final DATA step that is used only for `datatype=function`. For example, to periodically mark the function with pluses, specify the following:

```
adjust5=%str(if mod(_n_,30) = 0 then do;
                 size=0.25; function = 'LABEL'; text = '+'; output; end;)
```

Other Options

The remaining options for the `%PlotIt` macro are as follows:

cirsegs= *n*

specifies a circle smoothness parameter used in determining the number of line segments in each circle. Smaller values create smoother circles. The `cirsegs=` value is approximately related to the length of the line segments that compose the circle. The default is `cirsegs=.1`.

cursegs= *n*

specifies the number of segments in a regression function curve. The default is `cursegs=200`.

debug= vars | dprint | notes | time | mprint

specifies values that control debugging output.

`dprint` – print intermediate data sets.

`mprint` – run with `options mprint`.

`notes` – do not specify `options nonotes` during most of the macro.

`time` – displays total macro run time, ignored with `options nostimer;`

`vars` – displays macro options and macro variables for debugging.

You should provide a list of names for more than one type of debugging. Example: `debug=vars dprint notes time mprint`. The default is `debug=time`.

hpos= *n*

specifies the number of horizontal positions in the graphics area.

hsize= *n*

specifies the horizontal graphics area size in `unit=` units. The default is the maximum size for the device. By default, when `options=nocenter` is not specified, `hsize=` affects the size of the plot but not the `hsize= goption`. When `options=nocenter` is specified, `hsize=` affects both the plot size and the `hsize= goption`. If you specify just the `hsize=` but not `vsize=`, the vertical size is scaled accordingly.

makefit= *n*

specifies the proportion of the graphics window to use. When the `makefit=` value is negative, the absolute value is used, and the final value might be changed if the macro thinks that part of the plot might extend over the edge. When a positive value is specified, it will not be changed by the macro. When nonnull, the macro uses GASK to determine the minimum and maximum graphics window sizes and makes sure the plot can fit in them. The macro uses `gopprint=` or `gopplot=` to determine the device. The default is `makefit=-0.95`.

nknots= *n*

specifies the PROC TRANSREG number of knots option for regression functions.

outward= none | *char*

specifies a quoted string for the PLOT statement `outward=` option. Normally, this option's value is constructed from the symbol that holds the place for vectors. Specify `outward=none` if you want to not have `outward=` specified for vectors. The `outward=` option is used to greatly increase the likelihood that labels from vectors are displayed outward—away from the origin.

ps= *n*

specifies the page size.

radii= *do-list*

specifies the radii of circles (in a DATA step do list). The unit corresponds to the horizontal axis variable. The `radii=` option can also specify a variable in the input data set when radii vary for each point. By default, no circles are drawn.

regopts= *options*

specifies the PROC TRANSREG options for curve fitting. Example: `regopts=nknots=10 evenly`.

regfun= *regression-function*

specifies the function for curve fitting. Possible values include:

`linear` – line

`spline` – nonlinear spline function, perhaps with knots

`mspline` – nonlinear but monotone spline function, perhaps with knots

`monotone` – nonlinear, monotone step function

See PROC TRANSREG documentation for more information

regprint= *regression-options*
specifies the PROC TRANSREG PROC statement options, typically display options such as:

`noprint` – no regression printed output

`short` – suppress iteration histories

`ss2` – regression results

To see the regression table, specify: `regprint=ss2 short`. The default is `regprint=noprint`.

unit= `in` | `cm`
specifies the `hsize=` and `vsize=` unit in inches or centimeters (in or cm). The default is `unit=in`.

vechead= *vector-head-size* specifies how to draw vector heads. For example, the default specification `vechead=0.1 0.025`, specifies a head consisting of two hypotenuses from triangles with sides 0.1 units long along the vector and 0.025 units on the side perpendicular to the vector. This is smaller than the default in previous releases.

vpos= *n*
specifies the number of vertical positions in the graphics area.

vsize= *n*
specifies the vertical graphics area size in `unit=` units. The default is the maximum size for the device. By default when `options=nocenter` is not specified, `vsize=` affects the size of the plot but not the `vsize= goption`. When `options=nocenter` is specified, `vsize=` affects both the plot size and the `vsize= goption`. If you specify just the `vsize=` but not `hsize=`, the horizontal size is scaled accordingly.

vtoh= *n*
specifies the PROC PLOT `vtoh=` option. The `vtoh=` option specifies the ratio of the vertical height of a typical character to the horizontal width. The default is `vtoh=2`. Do not specify values much different than 2, especially by default when you are using proportional fonts. There is no one-to-one correspondence between characters and cells and character widths vary, but characters tend to be approximately twice as high as they are wide. When you specify `vtoh=` values larger than 2, near-by labels might overlap, even when they do not collide in the printer plot. The macro uses this option to equate the axes so that a centimeter on one axis represents the same data range as a centimeter on the other axis. A null value can be specified, `vtoh=`, when you want the macro to just fill the window, like a typical GPLOT.

Smaller values give you more lines and smaller labels. The specification `vtoh=1.75` is a good alternative to `vtoh=2` when you need more lines to avoid collisions. The specification `vtoh=1.75` means 7 columns for each 4 rows between ticks (7 / 4 = 1.75). The `vtoh=2` specification means the plot will have 8 columns for each 4 rows between ticks. Note that PROC PLOT sometimes takes this value as a hint, not as a rigid specification so the actual value might be slightly different, particularly when a value other than 2.0 is specified. This is generally not a problem; the macro adjusts accordingly.

xmax= *n*
specifies the maximum horizontal size of the graphics area.

ymax= *n*
specifies the maximum vertical size of the graphics area.