URL: http://cxc.harvard.edu/ciao3.4/sherpa−plot−hooks.html
Last modified: December 2006

AHELP for CIAO 3.4

sherpa−plot−hooks

Context: sherpa

Jump to: Description Bugs

Synopsis

Customizing Sherpa plots using the prefunc and postfunc fields of Sherpa State Objects

Description

The prefunc and postfunc Functions

The state objects sherpa.plot, sherpa.dataplot, sherpa.fitplot, sherpa.resplot, and sherpa.multiplot all have fields called prefunc and postfunc. These act as "hooks" for user functions that can be used with the LPLOT command. The point is to permit some additional manipulation of data before it is plotted to a drawing area (prefunc), and to allow additional ChIPS commands to be executed after each curve is plotted to a drawing area (postfunc). By default, these two fields are set to NULL; this means that no such functions are present unless the user adds them.

The function that prefunc refers to takes four arguments, in the following order:

- a variable of type Curve_Data
- a variable of type Curve_Label
- an Integer_Type variable specifying the dataset being plotted
- a String_Type variable specifying the type of plot (e.g. for sherpa.resplot.prefunc, this would be "residuals", "delchi", or "ratio")

These variables are passed to the user script by the Sherpa plotting script that calls the prefunc function. These types are defined in the Sherpa module, and are described below:

```
typedef struct
{
  x, y, xerrlo, xerrhi, yerrlo, yerrhi
}
Curve_Data;
```

The variables x and y are arrays of Float_Type; these arrays represent the data set to be plotted, and must be two arrays of equal length. The variables xerrlo, xerrhi, yerrlo, yerrhi represent the error bars; these also are arrays of Float_Type, of the same length as x and y, but need not necessarily be present.

```
typedef struct
{
  xlabel, ylabel
  }
  Curve_Label;
```

The xlabel field is a string containing the label for the x−axis, and the ylabel field is a string containing the label for the y−axis. If labels are not present, these fields are set to NULL.

(The user need not define these types, as they are already defined in the Sherpa module. All the user need do is to write functions like the examples shown below, and load them into Sherpa via "evalfile".)

A (rather dumb) example of a such a user script is shown here:

```
  define process_data(data, labels)
  {
    if (_NARGS != 2)
      return;
    if (typeof(data) != Curve_Data)
      return;

    data.y = data.y * 10;
    if (data.yerrhi != NULL)
      data.yerrhi = data.yerrhi * 10;
    if (data.yerrlo != NULL)
      data.yerrlo = data.yerrlo * 10;

    if (labels != NULL)
      labels.ylabel = "10 * " + labels.ylabel;

    return;
  }
```

When writing such user scripts, the user should always allow for the possibility that labels could be set to NULL, as could the error arrays of the data variable.

To have this function executed each time an LPLOT command is issued, the user would then update the prefunc field in this manner:

```
  sherpa> sherpa.plot.prefunc = &process_data
  sherpa> sherpa.dataplot.prefunc = &process_data % To apply to "lplot data"
  sherpa> sherpa.fitplot.prefunc = &process_data % To apply to "lplot fit"
  sherpa> sherpa.resplot.prefunc = &process_data % To apply to "lplot resid"

  sherpa> lp data  % y−values multiplied by 10 in the curve
                   % plotted to drawing area 1

  sherpa> lp 2 fit residuals % y−values multiplied by 10 in the
                             % fit curve plotted in drawing area 1,
                             % and y−values multiplied by 10 in the
                             % residuals curve plotted in drawing area 2.

  sherpa> sherpa.fitplot.prefunc = NULL

  sherpa> lp 2 fit residuals % y−values NO LONGER multiplied by 10 in the
                             % fit curve plotted in drawing area 1,
                             % but y−values STILL multiplied by 10 in the
                             % residuals curve plotted in drawing area 2.
```

(Please note that if this field is changed only in sherpa.plot, then the function process_data() is NOT run when the command LPLOT DATA is issued; to have that happen, the prefunc field of sherpa.dataplot must be changed. Similarly, if the user wants this function to be run during LPLOT FIT, the prefunc field of sherpa.fitplot must be changed; and to affect residual and ratio plots, sherpa.resplot must be updated.)

Please do NOT make this assignment:

```
sherpa> sherpa.plot.prefunc = process_data
```

since S−Lang will try to run process_data and assign the result to sherpa.plot.prefunc immediately. This will result in an error.

The function that postfunc refers to takes no arguments at all −− the purpose here is to allow some further modifications to plots via ChIPS commands after a curve has been plotted to a drawing area. Here is an example function:

```
  define add_plot_stuff()
  {
    if (_NARGS > 0)
      return;

    () = chips_eval("grids on");
    () = chips_eval("grids maj");
    () = chips_eval("symbol blue");
    return;
  }
```

which, once made available to Sherpa (via a call to evalfile), can be set by:

```
  sherpa> sherpa.dataplot.postfunc = &add_plot_stuff
  sherpa> lp data
```

Then every plot created with LPLOT DATA will have a grid with grid lines along the major tick marks, and the symbols used in the curve will be blue.

The prefunc and postfunc Fields of sherpa.multiplot

The sherpa.multiplot state object is a little different from the other sherpa state objects that affect plotting. The other plot state objects affect individual plots; the sherpa.multiplot object has settings that apply to all plots. (E.g., sherpa.multiplot.newarea controls whether or not new plots occur in separate drawing areas; if sherpa.multiplot.newarea is true, then LPLOT 2 FIT RESIDUALS results in the first plot occupying drawing area 1, and the second plot occupying drawing area 2. If false, all plots occupy the same drawing area.)

Consequently, the prefunc and postfunc functions will also behave a little differently. Instead of accepting arguments of type Curve_Data, sherpa.multiplot one argument that is a string. This string is composed of the arguments to the LPLOT command. Thus, if the command issued is LPLOT 2 FIT RESIDUALS, the Sherpa parser first interprets the command as LPLOT 2 FIT 1 RESIDUALS 1 (i.e., the fit and residual plots associated with data set 1). Then, the argument FIT 1 RESIDUALS 1 is passed to sherpa.multiplot.prefunc if that field is not set to NULL. The point here is that sherpa.multiplot.prefunc can identify each "type" plot going into each drawing area, as well as the data set number associated with each plot, and use that information to, for example, issue some ChIPS commands before any of the individual plots are created.

The function that sherpa.multiplot.postfunc refers to is not passed any arguments. Its purpose is similar to the postfunc field of the other plot state objects. The user can issue some final ChIPS commands, after all the plots have been created.

Bugs

See the Sherpa bug pages online for an up−to−date listing of known bugs.

Last modified: December 2006