Subject: plot object
Posted by Mark Fardal on Mon, 01 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Hey,

after my last silly question, thought I'd try to contribute something semi-useful for a change. So here's a generic plotting object that I've been using a bit lately.

The idea is to set a bunch of properties of the plot, including the data, and then tell it to draw itself. There are some advantages to this approach. You can fiddle with the properties without having to enter the entire plot command every time. The legend will draw itself semi-automatically. And if you want to select points based on where they lie on the plot, it's easy--there's one routine to select a polygon region and another to return the points within that region.

I find that this object is most useful in plots of intermediate formality. If I just want a simple, one-off plot for my one purposes it's easier just to enter the plot command directly, usually adding extra annotations with pencil. If I want the plot to go in a paper then I'd better save it in a routine. But if I want to show it to others in my research group, but not necessarily publish it, the plotting object seems to work the best for me. Maybe that's because I tend to tinker with plots.

I'm sure many people have written things like this. E.g., David F. wrote about one back in his long object-advocacy post. Anyone want to send theirs in? If we could get as many submissions as with the did with the "read tabular data" problem we could probably wind up with something pretty cool. Or if you have suggestions for my code please let me know.

Words of warning:
This is all recently written, and isn't production quality. I'm only submitting it to spur discussion.
The legend uses my slightly modified version of David Windt's legend.pro, which I call legend_w. Feel free to replace it with your favorite legend-constructor.
The routine to pick the polygon region is from the JHU library.
The routine PNPOLY to figure out which points are in the polygon is basically copied from the comp.graphics.algorithms FAQ. I can post it if you want.

Mark Fardal
UMass

```
 ********************************************************** ************

;+
; NAME
;   OBJPLOT__DEFINE
;
; PURPOSE: Defines an OBJPLOT object, which records the data and
;   plotting style for a plot.  A legend can be added easily.  Sets of
;   points can be selected with the mouse.  The plotting is done in
;   the direct graphics system.
;
; AUTHOR:
;   Mark Fardal
;   UMass
;
; CATEGORY:
;   Graphics
;
; CALLING SEQUENCE:
;   p = obj_new('objplot')
;
; INPUTS:
;   Input to the program is done through the object methods
;   SET, ADDDATA, CHANGEDATA.
;
; OBJECT METHODS:
;   set the plot properties
;     SET, title=title, xtitle=xtitle, ytitle=ytitle, tfont=tfont,
;        polycolor=polycolor
;   add a x-y pair, with optional properties
;     ADDDATA, x, y, name=name, psym=psym, symbol=symbol,
;            line=line, thick=thick, color=color
;   shorthand for ADDDATA, x(*,ix), y(*,ix)
;     ADDCOL, x, ix, iy
;   change data or plotting properties
;     CHANGEDATA, x, y, key=key, name=name, psym=psym,
;          symbol=symbol, line=line, thick=thick, color=color
;   remove one or all datasets
;     ZAPDATA, key=key
;   grab region on the screen
;     GRAB
;   return the points so selected, for dataset "key"
;     function SELECTED, key
;   draw the plot
;     DRAW, drawpoly=drawpoly, _extra=e
;   add a legend
;     ADDLEGEND, position=position, order=order, _extra=e
```

```
;
; OUTPUTS:
;   none
;
; SIDE EFFECTS:
;   headaches, nausea, vomiting
;
; RESTRICTIONS:
;   need the routines pnpoly, drawpoly, and legend_w to be in path
;   needs IDL 5.0 at least, only tested on 5.1
;
; EXAMPLE:
;   A random example:
;
;   IDL> x=findgen(100)/5.
;   IDL> y=sin(x)
;   IDL> p=obj_new('objplot')
;   IDL> p->adddata, x, y, psym=6
;   IDL> p->draw
;   IDL> p->changedata, color=!sky
;   IDL> p->adddata, x+5, y, color=!red
;   IDL> p->draw
;   IDL> p->grab
;   % Compiled module: DRAWPOLY.
;
;    Draw a polygon using the mouse.
;    Left button --- new side.
;    Middle button --- delete side.
;    Right button --- quit polygon.
;
;   IDL> print, p->selected(0)
;   % Compiled module: PNPOLY.
;           48          62
;   etc...
;
; MODIFICATION HISTORY:
;   original version (still evolving) 11/1/99
; BUGS:
;   inadequately tested
;   should store more plot properties
;   usage of key inconsistent, sometimes argument and sometimes keyword
;   methods to add:
;   get (get property)
;   shuffle (swap order of datasets)
;   zap (reset all properties)
; ACKNOWLEDGEMENTS:
;   J.D. Smith suggested the use of arrays of structures for the data
;
```

```
;utility routine for plotting specific colors sensibly
function objplot::checkcolor, color

result = color
fgset = where(color eq -1, count)
if (count gt 0) then result(fgset) = !p.color
return, result

end



;set one or more of the plot properties
pro objplot::set, $
  title=title, $
  xtitle=xtitle, $
  ytitle=ytitle, $
  tfont=tfont, $
  polycolor=polycolor

if n_elements(title) ne 0 then self.title = title
if n_elements(xtitle) ne 0 then self.xtitle = xtitle
if n_elements(ytitle) ne 0 then self.ytitle = ytitle
if n_elements(tfont) ne 0 then self.tfont = tfont
if n_elements(polycolor) ne 0 then self.polycolor = polycolor

end



;select region on screen
pro objplot::grab

if (n_elements(key) eq 0) then key = 0
drawpoly, xp, yp, /data
if (n_elements(xp) lt 3) then message, 'Need >= 3 vertices for polygon.'
ptr_free, self.xpoly, self.ypoly
self.xpoly = ptr_new(xp)
self.ypoly = ptr_new(yp)

end
```

```
;get the points in dataset "key" selected by the current poly region
;or -1 if no elements selected
function objplot::selected, key

nd = n_elements(*self.data)
if n_elements(key) ne 1 then begin
  if (n_elements(key) eq 0 and nd eq 1) then key = 1 $
  else message, 'Have to select one dataset.'
endif
if (key lt 0 or key ge nd) then $
    message, 'Dataset index out of range.'
if (self.xpoly eq ptr_new()) then  $
    message, 'Region not defined--use grab first.'

inpol = pnpoly(*self.xpoly, *self.ypoly, $
    *(*self.data)[key].x, *(*self.data)[key].y)
inlist = where(inpol gt 0, count)
if (count eq 0) then print, 'No points currently selected.'

return, inlist
end




;add a dataset drawn from two columns of an array
pro objplot::addcol, x, ix, iy, _extra=e

if n_params() lt 3 then message, 'Not enough parameters.'
self->adddata, x(*,ix), x(*,iy), _extra=e

end




;add a dataset to the plot object.  It will come after all the
;previous datasets
pro objplot::adddata, x, y, name=name, psym=psym, symbol=symbol, $
          line=line, thick=thick, color=color

if n_elements(name) eq 0 then name=''
if n_elements(psym) eq 0 then psym=0
if n_elements(symbol) eq 0 then symbol=0
if n_elements(line) eq 0 then line=0
if n_elements(thick) eq 0 then thick=0
if n_elements(color) eq 0 then color=-1  ;turned into !p.color at plotting
```

```
pair = {plotpair,  $
        name:name,  $
        x:ptr_new(x),  $
        y:ptr_new(y), $
        psym:psym, $
        symbol:symbol, $
        line:line, $
        thick:thick, $
        color:color $
       }

if ptr_valid(self.data) then begin
   *self.data = [*self.data, pair]
endif else begin
  self.data = ptr_new([pair])
endelse

end




;get rid of one or all datasets
pro objplot::zapdata, key=key

if n_elements(key) eq 1 then begin
  nd = n_elements(*self.data)
  if (key ge nd) then message, 'Sorry, no such dataset.'
  keep = where(indgen(nd) ne key)
  tmp = (*self.data)[keep]
  ptr_free, self.data
  self.data = ptr_new(tmp)
  return
endif else if n_elements(key) eq 1 then begin
  ptr_free, (*self.data).x, (*self.data).y
  ptr_free, self.data
  self.data = ptr_new()
;  ptr_free, (*self.data)[i].x, (*self.data)[i].y
;  *(*self.data) = *(*self.data)
endif else begin
  print, 'Sorry, have to zap one at a time.'
endelse

end
```

```
;change properties of the dataset specified
pro objplot::changedata, x, y, key=key, name=name, $
    psym=psym, symbol=symbol, line=line, thick=thick, color=color

nd = n_elements(*self.data)
if n_elements(key) eq 0 then begin
  if (nd eq 1) then key = 0 else begin
    print, 'Please specify dataset to change:'
    key = 0
    read, key
  endelse
endif

if not ptr_valid(self.data) then return
if key ge nd then $
  message, 'Sorry, no such dataset: ' + string(key)
;pair = (*self.data)[key]
;data = *self.data

if n_params() ge 1 then begin
  ptr_free, (*self.data)[key].x
  (*self.data)[key].x = ptr_new(x)
endif
if n_params() ge 2 then begin
  ptr_free, (*self.data)[key].y
  (*self.data)[key].y = ptr_new(y)
endif


if n_elements(name) ne 0 then (*self.data)[key].name=name
if n_elements(psym) ne 0 then (*self.data)[key].psym=psym
if n_elements(symbol) ne 0 then (*self.data)[key].symbol=symbol
if n_elements(line) ne 0 then (*self.data)[key].line=line
if n_elements(thick) ne 0 then (*self.data)[key].thick=thick
if n_elements(color) ne 0 then (*self.data)[key].color=color

end




;display the plot
pro objplot::draw, drawpoly=drawpoly, _extra=e

if (not ptr_valid(self.data)) then begin
  message, 'No data to plot!'
  return
```

```
endif

plot, *(*self.data)[0].x, *(*self.data)[0].y, /nodata, $
   title=self.tfont+self.title, $
   xtitle=self.tfont+self.xtitle, $
   ytitle=self.tfont+self.ytitle, $
   _extra=e

nd = n_elements(*self.data)
for i = 0, nd-1 do begin
  if ( (*self.data)[i].psym eq 8 ) then mysym, (*self.data)[i].symbol
  oplot, *(*self.data)[i].x, *(*self.data)[i].y, $
    psym=(*self.data)[i].psym, line=(*self.data)[i].line, $
    thick=(*self.data)[i].thick, color=self->checkcolor((*self.data)[i].color)
endfor

if keyword_set(drawpoly) then begin
  npol = n_elements(*self.xpoly)
  plots, *self.xpoly, *self.ypoly, color=self->checkcolor(self.polycolor)
  plots, (*self.xpoly)[[npol-1,0]], (*self.ypoly)[[npol-1,0]], $
         color=self->checkcolor(self.polycolor)
endif

end



;order is a
;position goes like
; 10  11  12
;  7   8   9
;  4   5   6
;order is an array indicating the order of keys to use in legend
;(doesn't have to be all of them!)
pro objplot::addlegend, position=position, order=order, _extra=e

if n_elements(position) eq 0 then position=6

if n_elements(order) eq 0 then order=indgen(n_elements(*self.data))

legend_w, ((*self.data).name)[order], position=position, $
  color=self->checkcolor(((*self.data).color)[order]), $
  psym=((*self.data).psym)[order], symbol=((*self.data).psym)[order], $
  thick=((*self.data).thick)[order], linestyle=((*self.data).line)[order], $
  _extra=e

end
```

```
pro objplot::cleanup

if ptr_valid(self.data) then begin
  ptr_free, (*self.data).x, (*self.data).y
  ptr_free, self.data
endif

end



function objplot::init

self.tfont='!6'
self.title = ''
self.xtitle = ''
self.ytitle = ''
self.data = ptr_new()
self.xpoly = ptr_new()
self.ypoly = ptr_new()
self.polycolor = -1

return, 1
end



pro objplot__define

;define the object
obj = {objplot,    $
       title:    '',     $
       xtitle:   '',  $
       ytitle:   '',  $
       tfont:    '',  $
       data:     ptr_new(), $
       xpoly: ptr_new(), $
       ypoly: ptr_new(), $
       polycolor: 0    $
      }

; Define an auxiliary structure for new data x-vs-y pairs
struct={plotpair,       $
```

```
    name:   '',     $
    x:   ptr_new(), $
    y:   ptr_new(), $
    psym:  0,       $
    symbol: 0,      $
    line:  0,       $
    thick: 0,       $
    color:  0       $
   }

end
```