Introduction to IDL
Dealing with Data

Week 2
Rachel McCrary
3-31-2009
Goals for today!

- Last week we learned how to plot to the display window (useful if you want to visualize your data quickly, not useful for publication)
- Today we are going to learn how to:
  - add color to your graphics
  - create PostScripts and jpsgs
  - make contour maps
  - deal with missing data when mapping
  - create colorbars for your graphics
  - create your own color table.
Adding color

• Adding color to your plots will seem quite annoying at first, but once you get the hang of it ... it will seem easy! (even though its really not :( )

• Remember that IDL is a “device” oriented model?

• This means is that you have to select the device you want to draw your graphics on (for now we are using the display window)

• Then you have to configure the graphics device ("backing store", decomposed vs indexed color, color display (8 bits vs 24 bits).

• This involves a whole lot of “computer language” that I don’t really understand myself, so I am just going to tell you what to do ... and if you want more info I can help after class.
When plotting to the window

- More than likely your computer will have a 24-bit (or better??) graphics card. This means that when plotting to the window you want to be using a true color visual class (not pseudocolor or direct color). When plotting to the window we need to configure our device to use the 24-bit mode: `device, true = 24`

- You also need to set up how the contents of graphics windows will be refreshed when windows are moved in front of or behind each other (this is called “backing store”). To do this you need to set the retain keyword, I suggest using: `device, retain = 2`, which means that IDL is responsible for maintaining backing store.
Colors

- A color in IDL is composed of three specific values. We call these values a color triple and write the color as (red, green, blue or RGB). Where the red, green, and blue values represent the amount of red, green and blue light that should be assigned that color on the display. Each value ranges from 0 to 255, thus a color may be made up of 256 shades or gradients of red, 256 shades of green, and 256 shades of blue.

- This means that there are 16.7 million colors that can be represented on the display by IDL.

<table>
<thead>
<tr>
<th>Commonly Used Color Triples</th>
<th>Color Triple Indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Red</td>
</tr>
<tr>
<td>white</td>
<td>255</td>
</tr>
<tr>
<td>black</td>
<td>0</td>
</tr>
<tr>
<td>red</td>
<td>255</td>
</tr>
<tr>
<td>orange</td>
<td>255</td>
</tr>
<tr>
<td>yellow</td>
<td>255</td>
</tr>
<tr>
<td>green</td>
<td>0</td>
</tr>
<tr>
<td>blue</td>
<td>0</td>
</tr>
<tr>
<td>purple</td>
<td>82</td>
</tr>
</tbody>
</table>
Color models

- Two different color models can be used to specify colors on a 24-bit graphics display, and some of the direct graphics commands work differently depending on the model currently in use.

- The two different models are called decomposed color and indexed color. (decomposed on and decomposed off)

- When configuring the device decomposed = 0 (index model), decomposed = 1 (decomposed color model).

- By default, when you set the 24-bit Truecolor environment, it will be using the decomposed model, which means that you can set the color using an RGB triple directly or: color = [255,255,0] (yellow)

- I am going to teach you how to use the “index model” - because this is what we will need to use to make PostScripts.
Device Configuration

• To configure our device so that we can draw graphics to the screen and everything works out so that we don’t get crazy flashing and unexpected colors do the following.

 device,true=24,decompose=0,retain=2

• Here we are using the truecolor color class, using the indexed color mode, and allowing IDL to manage backing store.

• If you are like me ... you will put this into a procedure that you can run and you won’t think about it again (until you want to make PostScripts)
So how do colors work in IDL?

• IDL comes with a number of pre-defined color tables.

• Color tables are indexed by a number (0-40).

• Load a color table into IDL using the `loadct`, command.

• Use `xpalette` (comes with IDL) or `cindex` (download script) to view the colors in each color table.

• The default color table in IDL is called `B-W linear`.

• A good color table to start with is color table 39.
**B-W Linear (loadct, 0)**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Color Index Numbers
Rainbow + white (loadct, 39)

<table>
<thead>
<tr>
<th>255</th>
<th>254</th>
<th>253</th>
<th>252</th>
<th>251</th>
<th>250</th>
<th>249</th>
<th>248</th>
<th>247</th>
<th>246</th>
<th>245</th>
<th>244</th>
<th>243</th>
<th>242</th>
<th>241</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>238</td>
<td>237</td>
<td>236</td>
<td>235</td>
<td>234</td>
<td>233</td>
<td>232</td>
<td>231</td>
<td>230</td>
<td>229</td>
<td>228</td>
<td>227</td>
<td>226</td>
<td>225</td>
<td>224</td>
<td>223</td>
</tr>
<tr>
<td>222</td>
<td>221</td>
<td>220</td>
<td>219</td>
<td>218</td>
<td>217</td>
<td>216</td>
<td>215</td>
<td>214</td>
<td>213</td>
<td>212</td>
<td>211</td>
<td>210</td>
<td>209</td>
<td>208</td>
<td>207</td>
</tr>
<tr>
<td>206</td>
<td>205</td>
<td>204</td>
<td>203</td>
<td>202</td>
<td>201</td>
<td>200</td>
<td>199</td>
<td>198</td>
<td>197</td>
<td>196</td>
<td>195</td>
<td>194</td>
<td>193</td>
<td>192</td>
<td>191</td>
</tr>
<tr>
<td>190</td>
<td>189</td>
<td>188</td>
<td>187</td>
<td>186</td>
<td>185</td>
<td>184</td>
<td>183</td>
<td>182</td>
<td>181</td>
<td>180</td>
<td>179</td>
<td>178</td>
<td>177</td>
<td>176</td>
<td>175</td>
</tr>
<tr>
<td>174</td>
<td>173</td>
<td>172</td>
<td>171</td>
<td>170</td>
<td>169</td>
<td>168</td>
<td>167</td>
<td>166</td>
<td>165</td>
<td>164</td>
<td>163</td>
<td>162</td>
<td>161</td>
<td>160</td>
<td>159</td>
</tr>
<tr>
<td>158</td>
<td>157</td>
<td>156</td>
<td>155</td>
<td>154</td>
<td>153</td>
<td>152</td>
<td>151</td>
<td>150</td>
<td>149</td>
<td>148</td>
<td>147</td>
<td>146</td>
<td>145</td>
<td>144</td>
<td>143</td>
</tr>
<tr>
<td>142</td>
<td>141</td>
<td>140</td>
<td>139</td>
<td>138</td>
<td>137</td>
<td>136</td>
<td>135</td>
<td>134</td>
<td>133</td>
<td>132</td>
<td>131</td>
<td>130</td>
<td>129</td>
<td>128</td>
<td>127</td>
</tr>
<tr>
<td>126</td>
<td>125</td>
<td>124</td>
<td>123</td>
<td>122</td>
<td>121</td>
<td>120</td>
<td>119</td>
<td>118</td>
<td>117</td>
<td>116</td>
<td>115</td>
<td>114</td>
<td>113</td>
<td>112</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>109</td>
<td>108</td>
<td>107</td>
<td>106</td>
<td>105</td>
<td>104</td>
<td>103</td>
<td>102</td>
<td>101</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>97</td>
<td>96</td>
<td>95</td>
</tr>
<tr>
<td>94</td>
<td>93</td>
<td>92</td>
<td>91</td>
<td>90</td>
<td>89</td>
<td>88</td>
<td>87</td>
<td>86</td>
<td>85</td>
<td>84</td>
<td>83</td>
<td>82</td>
<td>81</td>
<td>80</td>
<td>79</td>
</tr>
<tr>
<td>78</td>
<td>77</td>
<td>76</td>
<td>75</td>
<td>74</td>
<td>73</td>
<td>72</td>
<td>71</td>
<td>70</td>
<td>69</td>
<td>68</td>
<td>67</td>
<td>66</td>
<td>65</td>
<td>64</td>
<td>63</td>
</tr>
<tr>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
</tr>
<tr>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Setting the background and plot colors in IDL

• IDL’s default setting sets the background of the plot as black and the font as white. To change this use the background and color keywords. Set background and color to the desired color “index” in the selected color table.

• When using the B-W linear color table do the following:
  • plot, x, y, background = 255, color = 0
  • remember that you have to properly configure your device so that you don’t get a red background!!!
System Variables

• System variables are a special class of predefined variables available to all program units. Their names always begin with an exclamation mark (!)

• System variables are used to set the options for plotting. These variables start with !p. and all correspond directly with keyword parameters in the plot, oplot, contour and surface commands.

• For example, in the previous example we set the background and font colors inside of the call to the plot procedure.

• You can also set the background and colors of plots using the !p.background and !p.color system variables

• !p.background = 255 ;sets the background to color 255

• !p.color = 0 ;sets the axis colors/labels to color 0 in the table
When selecting color tables ...

- Some color tables do not have white or black - this means that you cannot have a white background or black axes labels.
- For my purposes I always try to pick a color table that has the colors I want, as well as white and black.
Plotting multiple data sets with different colors

- See: `plotting_colors.pro`
- Set and configure your device to plot to the screen.
- Load the color table into IDL
- Set the default background/color using `!p.background` and `!p. color`.
- Plot the axes (using `/nodata`)
- `plot` the datasets using `oplot`, set the color of each dataset in the `oplot` command.
- Consider adding a legend to make the plot more readable (use the `legend.pro` procedure)
In the previous example we first created a set of axes to plot all of our data sets on. Then we used oplot to actually plot the data. This required us to use the nodata keyword in the call to the plot command.

If this keyword is set, only the axes, titles, and annotation are drawn. No data points are plotted.

For example, to draw an empty set of axes between some given values:

```
PLOT, [XMIN, XMAX],[YMIN, YMAX], /NODATA
```
Adding a Legend

- use the legend.pro procedure to add a legend to your diagram (download script)

- Syntax:

- `legend` accepts, `color`, `psym`, `linestyle`, `position`, `/center`, `/bottom`, `/top`, `/vertical` etc.

- position in this case needs to be the position of the upper left hand corner of the legend
Fonts

- Graphic texts in IDL come in three different font “flavors” or styles:
  1. vector fonts (hershey fonts)
  2. true-type fonts
  3. hardware fonts

- The type of font is selected by using the !p.font system variable or by setting the font keyword on a graphical output command like plot or contour

- hardware fonts are typically used with PostScripts

- vector fonts tend to be too “thin” so true-type fonts are often used for other image types.

<table>
<thead>
<tr>
<th>!p.font</th>
<th>Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Vector</td>
</tr>
<tr>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>1</td>
<td>True-type</td>
</tr>
</tbody>
</table>
Publishable images - .jpeg

- To make a jpeg image, you want to use the command write_jpeg. This command puts what has been plotted to the display window into a .jpeg file

  WRITE_JPEG [, Filename], Image [, /ORDER] [, /PROGRESSIVE] [, QUALITY=value{0 to 100}] [, TRUE={1 | 2 | 3}] [, UNIT=lun]

- Since we are using truecolor (24 bits) then use the following:
  - image will be tvrd(true = 3), the tvrd function returns the contents of the specified portion of the display window
  - true = 3

- write_jpeg, 'test.jpg', tvrd(true=3), quality=100, true=3

- See: plotting_colors.pro
To make PostScript images (which I suggest when publishing because they retain amazing quality!!) you are going to have to change your device.

You will need to change your device settings to create PostScripts.

PostScripting is requires the use of the 8-bit color mode. This means, you don’t have to worry about configuring your device to the truecolor mode or setting decomposed on/off.

When making PostScripts you will use the device procedure to change fonts, page size, the file name

See: make_postscript.pro
How to make .eps files

; Set and configure your device

set_plot, 'ps'
device,/encapsulated,/HELVETICA, /BOLD, $
   filename='test_fonts.eps', $
   /color,bits=8,/portrait,/inches,xsize=8,ysize=8

; plot stuff ......

; close the device
device, /close

; set your device back to the display window
set_plot, 'x'
Contour

- The contour procedure draws a contour plot from data stored in a rectangular array (or grid). Both line contours and filled contours can be created. Note that outline and fill contours cannot be drawn at the same time. You will need to use the contour command twice to achieve this type of image. The syntax is as follows:

```
CONTOUR, Z [, X, Y] [, C_ANNOTATION=vector_of_strings]
[, C_CHARSIZE=value] [, C_CHARTHICK=integer]
[, CYLES=vector] [, C_LABELS=vector{each element 0 or 1}]
[, C_LINESTYLE=vector] [, C_ORIENTATION=degrees]
[, C_SPACING=value] [, C_THICK=vector]
[, /CELL_FILL | , /FILL] [, /CLOSED] [, /DOWNHILL]
[, NLEVELS=integer{1 to 60}] [, MAX_VALUE=value]
[, MIN_VALUE=value] [, /OVERPLOT] [{, /PATH_DATA_COORDS,
PATH_FILENAME=string, PATH_INFO=variable, PATH_XY=variable}
| , TRIANGULATION=variable] [, /PATH_DOUBLE] [, /XLOG]
[, /YLOG] [, ZAXIS={0 | 1 | 2 | 3 | 4}]
```

- Contour accepts the same graphics keywords accepted by PLOT except: linestyle, psym and symsize. (ie it accepts, title, thick, etc.)
Creating Contour Plots

• Any 2-D dataset can be displayed in IDL as a contour plot using the contour command.

• The contour procedure draws a contour plot from data stored in a rectangular array (a grid). Both line contours and filled contours can be created.

• The basic syntax of the contour procedure is:
  contour, data(x,y), x, y, ... keyword list

• line contour examples see: contour1.pro
• filled contour examples see: contour2.pro
Manipulating Contour Plots

- set the xstyle and ystyle keywords to prevent auto-scaling of the axes by IDL
- Use nlevels to contour data with N number of regularly spaced contours
- Use the /follow keyword to label every other contour level
- nlevels is sometimes treated as a “suggestion” for IDL, to explicitly set the levels use the levels keyword (this is highly suggested, and essentially mandatory when plotting filled contours).
- use the c_labels keyword to specify exactly which contour lines should be labeled
- contour plots can be modified using the same keywords as plot (title, xtitle, ytitle, position, charszie)
- use c_annotation to label each contour line with a string
- use c_linestyle to specify a vector of line style indices
- use c_thick to specify a vector of line thickness
Adding Color

- load in a color table
- select the indices for the background, and contour color.
- use the color keyword to set the axis color, use background to set the background color
- use c_colors to specify the contour colors (c_colors can also be a vector of color table indices, which is used in a cyclic fashion to draw the contour lines)
Filled Contour Plots

- To create a filled contour plot use the /fill keyword.
- Although you can use the nlevels keyword to make filled contours, it is best to define your own levels.
- Example: load 12 colors into a color table, specify your contour levels, specify the colors to fill each contour with:

```plaintext
loadct, 0
loadct, 4, ncolors = 12, bottom = 1
!p.color = 0
!p. background = 255
step = (max(peak) - min(peak)) / 12.0
clevels = indgen(12)*step+min(peak)
contour, peak, lon, lat, xstyle = 1, ystyle = 1,/fill,$
 levels = clevels, /follow, C_colors = indgen(12) +1
```
How do contour intervals work?

- `levels` is set to a vector that specifies the contour intervals for the plot.
- `c_colors` is set to a vector that specifies which color should be associated with which each contour interval.
- `levels = [0, 10, 20, 30, 40]`
- `c_colors = [1, 2, 3, 4, 5]`
- Any values > 40 are blue, any values less than 0 are set to the background color.
- Values in `c_colors` are cyclic.
/fill vs /cell_fill

- When dealing with missing data or contouring over a map projection, you will have to use /cell_fill rather than /fill to generate a filled contour map.

- /cell_fill is an older routine that is slightly less efficient, but works better with missing data and maps.

- Sometimes /cell_fill can damage the contour plot axes. To repair the axes draw the contour plot over again, without the data.

- You will need to use /nodata and /noerase in your second call of the contour procedure.

```plaintext
contour, peak, lon, lat, xstyle = 1, ystyle = 1,/cell_fill, $ levels = clevels, /follow, C_colors = indgen(12) +1
```

```plaintext
contour, peak, lon, lat, xstyle = 1, ystyle = 1,/cell_fill, $ levels = clevels, /follow, /nodata, /noerase, $ C_colors = indgen(12) +1
```
Adding a Colorbar

• IDL does not come with a predefined colorbar routine, you have to create one yourself. Fortunately lots of other people have already done this for you. For this class we are going to use the cbar.pro procedure created by David Fanning.

• You will tell the routine: the indices of the top/bottom colors to use, the number of divisions you want shown, the min/max range of your contour values, the format you want the labels to be written with, where you want it positioned etc.

```
cbar, bottom = 1, topp = 12, charsize = 1, divisions = 12, $position = [0.2, 0.15, 0.8, 0.2], range = [0, max(peak)], $format = '(F8.1)'```

Map Projections

- Often in our field we often want to project data onto a map of the globe (or a region etc).
- This requires a bit of extra work in IDL, but it is not difficult once you know what order your procedures need to go in.
- The basic format will go as follows:
  - configure the map setup that you want to project onto (position, center of map, polar, region) using map_set
  - contour the data onto your map using contour
  - add continents/states using map_continents
  - add grid using map_grid
- This also requires using /noerase and /overplot when appropriate
- See: contour3.pro
• Use the /noerase keyword when you want to preserve the information on the screen. The default for most plotting/mapping routines is to erase whatever is on the screen.
/overplot

- By setting the overplot keyword, the current graphics screen is not erased, no axes are drawn, and the previously established settings remain in effect.
Example Code

map_set,0, 180,/noerase,/isotropic,color=0,$
   position=[.2,.2,.8,.9]

contour, temp(*,*), lon, lat, /overplot, $
   /noerase, c_colors = indgen(253)+1, $
   levels = clevels, /cell_fill

map_continents,/continents,mlinethick=2,/overplot

map_grid, /label
map_set

- map_set is a procedure that establishes the axis type and coordinate conversion mechanism for mapping points on the earth’s surface, expressed in lon/lat.

- This procedure also sets up the clipping parameters of the region to be mapped, center of the map, and the polar rotation.

```
MAP_SET [, P0lat, P0lon, Rot], limit = limit, /isotropic
```

- P0lat - The latitude of the point on the earth's surface to be mapped to the center of the projection plane and must be in the range: \(-90^\circ \leq P0lat \leq 90^\circ\)

- P0lon - The longitude of the point on the earth's surface to be mapped to the center of the map projection and must be in the range: \(-180^\circ \leq P0lon \leq 180^\circ\)

- limit - sets the limits of the map (this is how you zoom in on a region)

- /isotropic makes the x and y axes use the same scale
map_continents

- the map_continents procedure draws continental boundaries, filed continents, political boundaries, coastlines, and/or rivers over a map projection

The `map_grid` procedure draws the parallels and meridians (gridlines) over a map projection.

Note `map_set` must be called before `map_grid`.

```plaintext
MAP_GRID [, /BOX_AXES | [, CLIP_TEXT=0] [, LATALIGN=value{0.0 to 1.0}] [, LONALIGN=value{0.0 to 1.0}] [, LATLAB=longitude] [, LONLAB=latitude] [, ORIENTATION=clockwise_degrees_from_horiz]] [, CHARSIZE=value] [, COLOR=index] [, /FILL_HORIZON] [, GLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, GLINETHICK=value] [, /HORIZON] [, INCREMENT=value] [, LABEL=n{label_every_nth_gridline}] [, LATDEL=degrees] [, LATNAMES=array, LATS=vector] [, LONDEL=degrees] [, LONNAMES=array, LONS=vector] [, MAP_STRUCTURE=structure] [, /NO_GRID]
```
Example Maps

- Simple Map Projection - map1.jpeg

Surface Air Temperature
Example Maps

• Map projection + labels and grid, use map_grid procedure - map2.jpeg
Example Maps

- Map Projection, grid no labels, use map_grid procedure- map3.jpeg

Surface Air Temperature
Example Maps

- Map projection, user labels, use xyouts to set axis labels - map4.jpeg

Surface Air Temperature
Example Maps

- Map Projection, zoom in on USA - map5.jpeg, Use limits keyword in the map_set procedure

Surface Air Temperature
Example Maps

• See weird white line? Let's get rid of it!
• Example: wrapping.pro

Surface Air Temperature
How to fix this problem

• To fix this problem, we need to “wrap” the data around the map.

• concatenate the first element in the longitude array to the end of the array (adding one element)

• concatenate the first element of the first dimension to the end of the gridded array

• re-plot
Example Maps

- Now it’s gone!!
- Example: wrapping.pro
Creating your own color tables

• As nice as the IDL predefined color tables are ... they are sometimes seriously lacking. For example IDL does not have a good colortable for plotting anomalies (with white in the center, cool colors on one side and warm colors on the other).

• Fortunately, you can create your own color tables in IDL!

• This fairly advanced stuff ...
color tables

• The basic concept:
  • Identify the RGB indices (or color triples) for the colors you want to put in your color table.
  • Create an \((x, 3)\) integer array that contains these color triples
  • Create a new integer array that is \((256, 3)\) and put the color triples into the order you want them in your color table
  • use the \texttt{tvlct} command to save the color table into something IDL can use!
  • See: \texttt{colors\_anomalies.pro}
# Positioning Text

<table>
<thead>
<tr>
<th>Embedded Command</th>
<th>Command Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>!A</td>
<td>Shift text above the division line</td>
</tr>
<tr>
<td>!B</td>
<td>Shift text below the division line</td>
</tr>
<tr>
<td>!C</td>
<td>Insert a carriable return and begin a new line of text</td>
</tr>
<tr>
<td>!D</td>
<td>Create a first-level subscript</td>
</tr>
<tr>
<td>!E</td>
<td>Create a first-level superscript</td>
</tr>
<tr>
<td>!I</td>
<td>Create an index-type subscript</td>
</tr>
<tr>
<td>!L</td>
<td>Create a second-level subscript</td>
</tr>
<tr>
<td>!N</td>
<td>Shift back to normal level after changing level</td>
</tr>
<tr>
<td>!R</td>
<td>Restore text position to last saved position</td>
</tr>
<tr>
<td>!S</td>
<td>Save the current text position</td>
</tr>
<tr>
<td>!U</td>
<td>Create an index-type superscript</td>
</tr>
<tr>
<td>!X</td>
<td>Return to entry font</td>
</tr>
<tr>
<td>!z</td>
<td>Display one ore more characters by their unicode value</td>
</tr>
<tr>
<td>!!</td>
<td>Display the exclamation mark !</td>
</tr>
</tbody>
</table>