Subject: - auroral_image.pro (1/1) Auroral_image.pro mapping of irregularly gridded data
Posted by Robert.M.Candey on Tue, 12 Nov 1996 08:00:00 GMT
View Forum Message <> Reply to Message

```
function doByteScale, Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, is1dim
  common deviceTypeC,deviceType,file;required for inverting grayscale Postscript
  if (Zsize(Zsize(0)+1) ne 1) then $ ; not Byte array
    Zb = bytscl(Z2, min=minZ1, max=maxZ1, top=!d.n_colors-3)+1B else Zb = Z2
 ; reserve black and white at ends of colorscale
; replace bad values with 0 but keep 2-dim array
  if not is1dim then if (wBad(0) ge 0) then Zb(wBad) = 0B ; minZ1
  if (n_elements(deviceType) ne 0) then if (deviceType eq 2) then $
 Zb = (!d.n_colors-1B) - Zb ; invert grayscale for Postscript
  if flipColorBar then Zb = (!d.n_colors-1B) - Zb ; invert for inverted cscale
return, Zb
end ; doByteScale

pro auroral_image, Z, Lon, Lat, centerLonLat=centerLonLat, $
  centerPole=centerPole, continents=continent, nogrid=nogrid, nLat=nLat, $
  nLon=nLon, rangeLonLat=rangeLonLat, debug=debug, rotation=rotation, $
  nocolorbar=nocolorbar, ctitle=ctitle, cscale=cscale, cCharSize=cCharSize, $
  maxValue=maxValue, minValue=minValue, fillValue=fillValue, logZ=logZ, $
  method=method, slow=slow, status=status, proj=proj, tvscale=tvscale, $
  altitude=altitude, mapColor=mapColor, mapCharSize=mapCharSize, _Extra=extra
; , $ position=position
;+
; NAME:
; Auroral_Image
;
; PURPOSE:
; This function plots a 2-dimensional array Z mapped onto a polar
; projection, with each point mapped to its corresponding Longitude and
; Latitude (possible to use three 1-dim arrays also)
;
; CATEGORY:
; Graphics
;
; CALLING SEQUENCE:
; AURORAL_IMAGE, Z, Lon, Lat
;
; INPUTS:
; Z: 2-dimensional data array (nLon, nLat)
;  or 1-dimensional (requires Lon and Lat be defined)
;  (currently assumes Z is always positive)
;
; OPTIONAL INPUTS:
; Lon: Corresponding Longitudes for Z, degrees (-180:+180 deg)
```

```
;  1-dimensional data array of size Z(*,0) or 2-dim of size Z(*,*)
;  or 1-dimensional of size Z(*) if Z is 1-dim
;##### try to handle longitudes 0:360??
; Lat: Corresponding Latitudes for Z, degrees (-90:+90 deg)
;  1-dimensional data array of size Z(0,*) or 2-dim of size Z(*,*)
;  or 1-dimensional of size Z(*) if Z is 1-dim
; Assumes measurements in Z are centered on Lat, Lon values
;
; KEYWORD PARAMETERS:
; LOGZ=logZ: Scale Z data and color scale logarithmically
; CENTERPOLE=centerPole: Center on nearest pole (overrides centerLonLat)
; CENTERLONLAT=centerLonLat: Center of polar plot [Lon, Lat], degrees
; (if centerPole and centerLonLat not provided, center on middle of image)
; RANGELONLAT=rangeLonLat: Range of Longitude and Latitude, degrees
;   [minLon, minLat, maxLon, maxLat]  (Lat may be >90 or <-90 and
;   Lon may be >180 or <-180 for crossing poles and Dateline)
;  default is to plot the whole sphere (Earth)
; CONTINENT=continent: switch to plot continent outlines
;   (requires Lon and Lat in geographic coordinates)
; NOGRID=nogrid: switch to not plot map grid (Lat, Lon lines)
;  default is to plot grid lines (added by request)
; NLON=nLon: Number of pixels of output image in Longitude dim (def=50)
; NLAT=nLat: Number of pixels of output image in Latitude dim (def=50)
;  NLon and NLat only used by Map_image and TV
; MAXVALUE=maxValue: Max value of data to plot; values above are ignored
; MINVALUE=minValue: Min value of data to plot; values below are ignored
; FILLVALUE=fillValue: Data with this value are ignored
;       NOCOLORBAR=nocolorbar:  Switch to not plot a color bar on right
;  default is to plot colorbar (added by request)
; CTITLE=ctitle: String title for colorbar
; CSCALE=cscale: scale for colorbar range [min, max]
; CCHARSIZE=cCharSize: Character size for axis on color bar
; ROTATION=rotation: Rotate map, degrees (def=0), positive clockwise
; TVSCALE=tvScale: Integer scaling factor for TV method only
; METHOD=method: String indicating which method to plot with:
;   "TV":  TV bitmap of Z scaled to color table (no map, no plot)
;   "QUICK": Quick plotting of Z array only (no map)
;   "PLOTS": Use plots command
;   "POLYFILL": Use polyfill
;   "MAPIMAGE": Use map_image and tv (least preferred method)
;   "DILATE": Use convert_coord, dilate and tv
; SLOW=slow: Switch to use bilinear and quintic interpolations (def=0)
;  Only used by Map_image method
; STATUS=status:  Return 0 if plot okay, else -1 for an error,
;   status variable must be predefined before call
; PROJ=proj: Set map projection; defaults to /satellite
;   1=stereo, 2=ortho, 3=cone, 4=lamb, 5=gnom, 6=azim,
;   7=satel, 9=merc, 10=moll, 14=sinu, 15=aitoff
```

```
;   Simple cyl proj: 11=8, 12=merc, 13=moll, 18=aitoff
;   use 1, 2, 4, 6, 7 (see IDL manual for more info)
; ALTITUDE=altitude: Distance from center of sphere in units of sphere's
;   radius (for satellite (7) projection)
; MAPCOLOR=mapColor: color index for grids and continent overlays;
;   default=white
; MAPCHARSIZE=mapCharSize: character size for grid numbers; default=1.0
; _EXTRA=extra: Any extra parameters to pass on to plot outline
;     Add your own title, xtitle, ytitle
;    May be able to over-ride plot location/size with position
; DEBUG=debug: switch to print debugging statements
;
; OUTPUTS:
; No outputs.
;
; COMMON BLOCKS:
; DEVICETYPEC: deviceType
; Shared with DeviceOpen.pro to allow inverting grayscale Postscript
;
; SIDE EFFECTS:
; Creates plot to screen or file.
;
; RESTRICTIONS:
; Sets a specific X margin to allow for the colorbar.
;
; PROCEDURE:
; Uses map_set, triangulate, trigrid, map_image to map pixels to polar
; coordinates; TV to display and map routines to draw grids and continents
; A colorbar is plotted on the right if not /nocolorbar or cscale is set.
;
; EXAMPLE:
; Create a polar plot of 2 dimensional data Z = dist(60), with
; lat = [findgen(30)+60., 90.-findgen(30)], lon = findgen(60)*6.-180.
; auroral_image, Z, lon, lat
;
; MODIFICATION HISTORY:
;  Written by: Emily A. Greene, Hughes STX, emily@xfiles.gsfc.nasa.gov
; and Bobby Candey, NASA GSFC Code 632; Robert.M.Candey.1@gsfc.nasa.gov
; 1995 Sept 26 BC, original full copy
; 1995 Sept 29 BC, beta
; 1995 Oct 2 BC, beta 2
; 1995 Nov 3 BC, reversed all Lat and Lon to agree with map_image
; 1996 April 11 BC, cleanup
; 1996 April 16 BC, cleanup, add status
; 1996 April 17 BC, added colorBar to allow autoscale
; 1996 April 18 BC, added methods
; 1996 April 19 BC, added projection keyword and rearranged
; 1996 April 20 BC, added doByteScale function and surface method
```

```
; 1996 July 9 BC, added not_nearby, changed centerLonLat calc.
; 1996 July 11 BC, fixed polyfill method, added even scale to quick
; 1996 July 17 BC, cleaned up, added mapColor
; 1996 Aug 5 BC, added comments and removed old stuff
; 1996 Aug 6 BC, added dilate method
;-

common deviceTypeC, deviceType, file;required for inverting grayscale Postscript
if keyword_set(debug) then debug = 1 else debug = 0
;if (n_elements(!debug) gt 0) then if !debug then debug = 1
if (n_elements(status) gt 0) then doStatus = 1 else doStatus = 0
status = 0L

if (n_params(0) lt 1) then message, 'Auroral_image, Z, Lon, Lat'
Z1 = reform(Z) ; remove extraneous dimensions
Zsize = size(Z1)
if ((Zsize(0) lt 1) or (Zsize(0) gt 2)) then begin
  msgText = 'Requires 1- or 2-dimensional Z array'
  if doStatus then begin
    message, msgText, /info & status = -1L & return
  endif else message, msgText
endif
if (Zsize(0) eq 1) then is1dim = 1 else is1dim = 0

case n_params(0) of
1: begin ; Z only
  if (is1dim eq 0) then begin ; 2-dim
    midZ = long(Zsize(2)/2)
    Ltt = findgen(midZ)/midZ * 89. + 1
    Lat1 = ([Ltt,90.,reverse(Ltt)])(0:Zsize(2)-1)
    Lat1 =  rebin(reform(Lat1,1,Zsize(2),/overwrite),Zsize(1),Zsize(2),/ sample)
    Lat1 = (Lat1 + (randomu(seed,Zsize(1),Zsize(2)) - 0.5)*2.) > 1 < 90
    Lon1 =  rebin(findgen(Zsize(1))/Zsize(1)*360.-180.,Zsize(1),Zsize(2) ,/sample)
    Lon1 = (Lon1 + (randomu(seed,Zsize(1),Zsize(2)) - 0.5)*2.) > (-180) < 180
  endif else begin ; 1-dim
    midZ = long(Zsize(1)/2)
    Ltt = findgen(midZ)/midZ * 89. + 1
    Lat1 = ([Ltt,90.,reverse(Ltt)])(0:Zsize(1)-1)
    Lat1 = (Lat1 + (randomu(seed,Zsize(1)) - 0.5)*2.) > 1 < 90
    Lon1 = findgen(Zsize(1))/Zsize(1)*360.-180.
    Lon1 = (Lon1 + (randomu(seed,Zsize(1)) - 0.5)*2.) > (-180) < 180
  endelse
end ; Z only
3: begin ; Z, Lat, Lon
  Lat1 = 1.0*reform(Lat) & Lon1 = 1.0*reform(Lon) ; remove extraneous dimensions
  Vsize = size(Lat1)
  if is1dim then begin
    if n_elements(Lat1) ne Zsize(1) then begin
```

```
      msgText = 'Lat must be of same size as Z(*)'
      if doStatus then begin
        message, msgText, /info & status = -1L & return
      endif else message, msgText
    endif ; else Lat1 agrees in size with Z
  endif else begin ; not 1-dim
    if (n_elements(Lat1) eq Zsize(2)) then begin ; make 2-dim
      Lat1 =  rebin(reform(Lat1,1,Zsize(2),/overwrite),Zsize(1),Zsize(2),/ sample)
    endif else begin
      if not ((Vsize(1) eq Zsize(1)) and (Vsize(2) eq Zsize(2))) then begin
        msgText = 'Lat must be of same size as Z(0,*) or Z(*,*)'
        if doStatus then begin
          message, msgText, /info & status = -1L & return
        endif else message, msgText
      endif ; else Lat1 is already same size as Z
    endelse
  endelse
  Vsize = size(Lon1)
  if is1dim then begin
    if n_elements(Lon1) ne Zsize(1) then begin
      msgText = 'Lon must be of same size as Z(*)'
      if doStatus then begin
        message, msgText, /info & status = -1L & return
      endif else message, msgText
    endif ; else Lon1 agrees in size with Z
  endif else begin ; not 1-dim
    if (n_elements(Lon1) eq Zsize(1)) then begin ; make 2-dim
      Lon1 = rebin(Lon1,Zsize(1),Zsize(2),/sample)
    endif else begin
      if not ((Vsize(1) eq Zsize(1)) and (Vsize(2) eq Zsize(2))) then begin
        msgText = 'Lon must be of same size as Z(*,0) or Z(*,*)'
        if doStatus then begin
          message, msgText, /info & status = -1L & return
        endif else message, msgText
      endif ; else Lon1 is already same size as Z
    endelse
  endelse
end ; Z, Lat, Lon

else: begin
        msgText = 'Wrong number of arguments'
        if doStatus then begin
          message, msgText, /info & status = -1L & return
        endif else message, msgText
      end
endcase

;##### remove "> 0." on next 30 lines for real Z < 0
```

```
;if (n_elements(minValue) gt 0) then minZ = minValue(0) else minZ = min(Z1) > 0.
;if (n_elements(maxValue) gt 0) then maxZ = maxValue(0) else maxZ = max(Z1) > 0.
if (n_elements(minValue) gt 0) then minZ = minValue(0) else minZ = min(Z1)
if (n_elements(maxValue) gt 0) then maxZ = maxValue(0) else maxZ = max(Z1)
if (n_elements(fillValue) gt 0) then begin
  fillZ = fillValue(0)
  wZfill = where(Z1 eq fillZ, wZfillc)
  if (wZfillc gt 0) then begin
    wnotZfill = where(Z1 ne fillZ, wnotZfillc)
    if (wnotZfillc gt 0) then begin
      if (n_elements(minValue) le 0) then minZ = min(Z1(wnotZfill))
      if (n_elements(maxValue) le 0) then maxZ = max(Z1(wnotZfill))
    endif
  endif
endif else fillZ = minZ - 1

; note Lat/Lon have values of -300. which are off the earth for DE-1 SAI CDFs
isBad = (Lat1 lt -90.) or (Lat1 gt 90.) or (Lon1 lt -180.) or (Lon1 gt 180.) $
 or (Z1 lt minZ) or (Z1 gt maxZ) or (Z1 eq fillZ)
wBad = where(isBad, wBadc)
if (wBadc gt 0) then begin
  wGood = where(isBad ne 1, wGoodc)
  if (wGoodc le 0) then begin ; quit here
    msgText = 'No good values to display'
    if doStatus then begin
      plot, [0,1], [0,1], /nodata, _Extra=extra
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
;  if (n_elements(minValue) le 0) then minZ=min(Z1(wGood)) > 0.
;  if (n_elements(maxValue) le 0) then maxZ=max(Z1(wGood)) > 0.
  if (n_elements(minValue) le 0) then minZ=min(Z1(wGood))
  if (n_elements(maxValue) le 0) then maxZ=max(Z1(wGood))
  if (n_elements(fillValue) le 0) then fillZ = minZ - 1
endif

if not keyword_set(logZ) then logZ = 0
doColorBar = 0
flipColorBar = 0 ; flip color bar if cScale is inverted
if (n_elements(cscale) ne 0) then begin
  doColorBar = 1
  ; check accuracy of cscale
  if n_elements(cscale) ne 2 then begin
    msgText = 'Error in cscale dimensions, no colorbar plotted'
    if doStatus then begin
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
```

```
  if logZ then begin ; check if cscale is less than 0 and minZ
;    if (n_elements(minValue) ne 0) then minCscale = minValue(0) > 0 $
     if (n_elements(minValue) ne 0) then minCscale = minValue(0) $
       else minCscale = 0
;    minCscale = minZ > 0
     wcs = where(cscale le minCscale, wcsc)
     if wcsc gt 0 then begin
       ws = where(Z1 gt minCscale, wsc)
       if wsc gt 0 then cscale(wcs) = min(Z1(ws)) else cscale(wcs) = minCscale
     endif ; bad cscale
  endif ; logZ
  doCheck = 0
;  if keyword_set(CheckScale) then doCheck = 1
  if doCheck then begin ; check for exceeding maxZ or less than minZ
;    if (n_elements(maxZ) ne 0) then begin
;      wcs = where(cscale gt maxZ, wcsc) ; could be "ge"
     if (n_elements(maxValue) ne 0) then begin
       wcs = where(cscale gt maxValue(0), wcsc) ; could be "ge"
       if wcsc gt 0 then begin
         ws = where(Z1 le maxValue(0), wsc) ; could be "lt"
         if wsc gt 0 then cscale(wcs) = max(Z1(ws)) else cscale(wcs)=maxValue(0)
;        ws = where(Z1 le maxZ, wsc) ; could be "lt"
;        if wsc gt 0 then cscale(wcs) = max(Z1(ws)) else cscale(wcs) = maxZ
       endif
     endif
;    if (n_elements(minZ) ne 0) then begin
;      wcs = where(cscale lt minZ, wcsc) ; could be "le"
     if (n_elements(minValue) ne 0) then begin
       wcs = where(cscale lt minValue(0), wcsc) ; could be "le"
       if wcsc gt 0 then begin
         ws = where(Z1 ge minValue(0), wsc) ; could be "gt"
         if wsc gt 0 then cscale(wcs) = min(Z1(ws)) else cscale(wcs)=minValue(0)
;        ws = where(Z1 ge minZ, wsc) ; could be "gt"
;        if wsc gt 0 then cscale(wcs) = min(Z1(ws)) else cscale(wcs) = minZ
       endif
     endif
  endif ; doCheck min/max Z in cscale
  minZ = min(cscale) & maxZ = max(cscale)
  if (cscale(0) gt cscale(1)) then flipColorBar = 1
  if (n_elements(fillValue) le 0) then fillZ = minZ - 1
endif else begin; colorBar without cscale
  if not keyword_set(nocolorbar) then begin
    doColorBar = 1
    cscale = [minZ, maxZ]
  endif
endelse
if debug then message, /inform, string('DEBUG: minZ, maxZ = ', minZ, maxZ)
```

```
if (n_elements(centerLonLat) le 0) then begin
  if is1dim then centerLat = long(Lat1(Zsize(1)/2 - long(sqrt(Zsize(1))/2.))) $
     else centerLat = long(Lat1(Zsize(1)/2,Zsize(2)/2))
; #### may want to do this next part all the time
  if (centerLat lt -90) or (centerLat gt 90.) then begin
    if debug then message, string('Bad centerLat = ', centerLat), /inform
    if (wBadc gt 0) then centerLat = 90. * sign(avg(Lat1(wGood))) else $
    centerLat = 90. * sign(avg(Lat1))
  endif
  if is1dim then centerLon = long(Lon1(Zsize(1)/2 - long(sqrt(Zsize(1))/2.))) $
     else centerLon = long(Lon1(Zsize(1)/2,Zsize(2)/2))
  if (centerLon lt -180) or (centerLon gt 180.) then begin
    if debug then message, string('Bad centerLon = ', centerLon), /inform
    centerLon = 0.
  endif
endif else begin
  if (n_elements(centerLonLat) ne 2) then begin
    msgText = 'centerLonLat must be an array of size 2'
    if doStatus then begin
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
  centerLon = centerLonLat(0)
  centerLat = centerLonLat(1)
endelse
if keyword_set(centerPole) then begin ; define at closest pole
  centerLat = 90. * sign(centerLat)
  centerLon = 0.
endif
if debug then message, /inform, $
 string('DEBUG: centerLon, centerLat = ', centerLon, centerLat)

; find the min and max lat/lon values
if (n_elements(rangeLonLat) gt 0) then begin
  if (n_elements(rangeLonLat) ne 4) then begin
    msgText = 'rangeLonLat not size 4'
    if doStatus then begin
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
  minLon = rangeLonLat(0)
  minLat = rangeLonLat(1)
  maxLon = rangeLonLat(2)
  maxLat = rangeLonLat(3)
endif else begin
  minLat = (centerLat - 90.) > (-90.)   & maxLat = (centerLat + 90.) < (90.)
  if (abs(centerLat) lt 1.e-5) then begin ; near equator
    minLon = (centerLon - 90.)  & maxLon = (centerLon + 90.)
```

```
  endif else begin ; near pole
    minLon = (centerLon - 180.)  & maxLon = (centerLon + 180.)
  endelse
endelse
;               Reduce longitudes
  while minLon gt 180.    do minLon = minLon - 360.
  while minLon lt (-180.) do minLon = minLon + 360.
  while maxLon gt 180.    do maxLon = maxLon - 360.
  while maxLon lt (-180.) do maxLon = maxLon + 360.
; Wrap around?  Allow for round-off.
  if (maxLon-1.0e-5) lt minLon then maxLon = maxLon + 360.

if debug then message, /inform, 'DEBUG:minLon,minLat,maxLon,maxLat='+ $
  string(minLon,minLat,maxLon,maxLat, format='(f6.1,1x,f5.1,1x,f6.1,1x,f5.1)')

; Lat -90:90; Lon -180:180; rot -180:180 positive clockwise for map_set
  if n_elements(rotation) gt 0 then rotate1 = rotation(0) else rotate1 = 0.
; ### could try to compute a rotation matching original image, 2-dim case only

minZ1 = minZ & maxZ1 = maxZ
; #### do log before or after interpolating???
Z2 = Z1
if (Zsize(Zsize(0)+1) ne 1) then begin ; not Byte array
  if logZ then begin
    wh = where(Z1 le 0, wc)
    if (wc eq 0) then begin
      Z2 = alog10(Z1)
    endif else begin
      Z2 = Z1*0 ; all 0's
      wh = where(Z1 gt 0, wc)
      if (wc gt 0) then Z2(wh) = alog10(Z1(wh))
    endelse
    if (minZ le 0.) then minZ1 = 0. else minZ1 = alog10(minZ)
    if (maxZ le 0.) then maxZ1 = 0. else maxZ1 = alog10(maxZ)
  endif
endif

xmargin = !x.margin ; & ymargin = !y.margin
if doColorBar then if (!x.omargin(1)+!x.margin(1)) lt 14 then !x.margin(1) = 14
;pPosition = !p.position ; save for later
;w = where(pPosition eq 0, wc)
;if (n_elements(position) ne 0) then positiont = position else $
;    if (wc ne 4) then positiont = pPosition else $
; positiont = [0.1, 0.1, 0.88, 0.9]

;if n_elements(nLon) le 0 then nLon = 50 < Zsize(1)
;if n_elements(nLat) le 0 then nLat = 50 < Zsize(2) ; ###?
if n_elements(nLon) le 0 then if is1dim then nLon = long(sqrt(Zsize(1))) > 30 $
```

```
    else nLon = Zsize(1) > 30
if n_elements(nLat) le 0 then if is1dim then nLat = long(sqrt(Zsize(1))) > 30 $
    else nLat = Zsize(2) > 30

tvlct, red1, green1, blue1, /get
allColor = long(red1) + green1 + blue1
temp = min(allColor, wBlack);wBlack index into color table for lowest intensity
temp = max(allColor, wWhite);wWhite index into color table for highest intensity
if (n_elements(mapColor) le 0) then mapColor = wWhite ; or !p.background
if (n_elements(mapCharSize) le 0) then mapCharSize = 1.0

if (n_elements(altitude) le 0) then altitude=2.0 else altitude=altitude > 1.1

if (n_elements(method) le 0) then begin
  method1 = "PL" ; plots default
endif else begin
  if (strlen(method) lt 2) then begin
    method1 = "PL" ; plots default
  endif else begin
    method1 = strupcase(strmid(method, 0, 2))
  endelse
endelse ; method
if (method1 ne "TV") and (method1 ne "QU") then begin ; map setup
  limitVec = [minLat, minLon, maxLat, maxLon]
  revLimit = [minLon, minLat, maxLon, maxLat]
; ##### min/maxLon may have to exceed -180:180
  ; set up the map projection ; #### which default projection?
  if n_elements(proj) gt 0 then iproj = proj else iproj = 7 ; assume satellite
  if (iproj lt 1) or (iproj gt 18) then iproj = 7
; 1=stereo, 2=ortho, 3=cone, 4=lamb, 5=gnom, 6=azim, 7=satel, 9=merc, 10=moll,
; 14=sinu, 15=aitoff (> 8 are cylindrical except 14, add 3 for simple proj)
; use 1, 2, 4, 6, 7
  if (n_elements(rangeLonLat) ne 4) then begin
    map_set, centerLat, centerLon, rotate1, /isotropic, proj=iproj, $
 /whole_map, sat_p=[altitude,0.,0.], $
 _Extra=extra
    revLimit = !map.out([2,4,3,5]) ; if limitvec is not used
  endif else begin
    map_set, centerLat, centerLon, rotate1, /isotropic, proj=iproj, $
 /whole_map, sat_p=[altitude,0.,0.], $
 limit=limitvec, $; /advance, /horizon, $
 _Extra=extra
    revLimit = !map.out([2,4,3,5]) ; if limitvec is not used
  endelse
  if debug then message, /inform, 'DEBUG: revLimit = '+ $
 string(revLimit, format='(f6.1,1x,f5.1,1x,f6.1,1x,f5.1)')
endif ; not TV or Quick
```

```
case method1 of
"TV": begin ; TV method
  if is1dim then begin
    msgText = 'TV method requires 2-dimensional Z array'
    if doStatus then begin
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
  if (n_elements(tvScale) gt 0) then tvscale = fix(tvScale) > 1 else tvScale=1
  Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 0) ; is1dim
  xWinsize = nLon * tvScale
  yWinsize = nLat * tvScale
  maxWinSize = max([xWinsize, yWinsize])
  if (!d.flags and 1L) then $ ; scalable pixels (Postscript)
    tv, Zb, xsize=float(xWinsize)/maxWinSize, $
     ysize=float(yWinsize)/maxWinsize, /normal, _Extra=extra else $
    tv, rebin(Zb, xWinSize, yWinSize, /sample), _Extra=extra
end ; TV method

"QU": begin ; Quick method ; quick plot of raw Z array
  if is1dim then begin
    msgText = 'Quick method requires 2-dimensional Z array'
    if doStatus then begin
      message, msgText, /info & status = -1L & return
    endif else message, msgText
  endif
  Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 0) ; is1dim
;  plot, [minLon, maxLon], [minLat, maxLat], $ ; position = positiont, $
  plot, [0, nLon], [0, nLat], $ ; position = positiont, $
       /nodata, xstyle=1+4, ystyle=1+4+16, _Extra=extra
  px = !x.window*!d.x_size
  py = !y.window*!d.y_size
  xWinsize = px(1)-px(0)
  yWinsize = py(1)-py(0)
; even scale???
  xr = !x.crange(1) - !x.crange(0) ; data range in data coordinates
  yr = !y.crange(1) - !y.crange(0)
  ratio = max([float(xr)/xWinSize, float(yr)/yWinSize])
  xWinSize = xr / ratio
  yWinSize = yr / ratio
  if (!d.flags and 1L) then $ ; scalable pixels (Postscript)
    tv, Zb, px(0), py(0), xsize=xWinsize, ysize=yWinsize else $
    tv, congrid(Zb,xWinsize,yWinsize), px(0), py(0)
;  plot, [minLon, maxLon], [minLat, maxLat], $ ; position = positiont, $
  plot, [0, nLon], [0, nLat], $ ; position = positiont, $
       /nodata, xstyle=1+4, ystyle=1+4+16, _Extra=extra, /noerase
end ; Quick method
```

```
"PL": begin ; Plots method
  ; remove bad values which reduces to 1-dim arrays
    if (wBadc gt 0) then begin
      Lat1 = Lat1(wGood)  &  Lon1 = Lon1(wGood)  &  Z2 = Z2(wGood)
    endif
    Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 1) ; is1dim
    nsides = 6 ; 16
    avec = findgen(nsides) * (!pi*2/nsides)
    usersym, cos(avec), sin(avec), /fill
; plot a black background circle before plotting data to remove distractions
    nsides = 720
    avec = findgen(nsides) * (!pi*2/nsides)
    xavec = (!x.window(1)-!x.window(0))/2. * (1.+cos(avec)) + !x.window(0)
    yavec = (!y.window(1)-!y.window(0))/2. * (1.+sin(avec)) + !y.window(0)
    polyfill, xavec, yavec, color=wBlack, /normal ; black background
    plots, Lon1, Lat1, color=Zb, psym=8, noclip=0 ; , symsize=?
;#### compute based on window?
end ; Plots method

"PO": begin ; Polyfill method
  if is1dim then begin ; #### maybe 2-dim also??
  ; remove bad values which reduces to 1-dim arrays
    if (wBadc gt 0) then begin
      Lat1 = Lat1(wGood)  &  Lon1 = Lon1(wGood)  &  Z2 = Z2(wGood)
    endif
;    triangulate, Lon1, Lat1, triangles, bounds, sphere=tridata, fvalue=Z2, $
; /degrees, repeats = repeats, connectivity=list
    triangulate, Lon1, Lat1, triangles, sphere=tridata, fvalue=Z2, /degrees
; ### Z2(10150) is rearranged from before
; tridata.xyz(10150,3) ranges +/-1
; tridata.iend(10150) ranges 4-60881 for 10150 triangles
; tridata.iadj(60900) ranges 0-10150
; repeats = [-1,-1] ; bounds is small array of indices < 10150
; triangles(3,20285)
; list() where list(list(i):list(i+1)-1) has nodes adjacent to i
;    Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 1)
    pAll = convert_coord(Lon1, Lat1, /data, /to_normal)
    pLon = pAll(0,*) & pLat = pAll(1,*)
    for i=0L,n_elements(triangles(0,*))-1 do begin
      tri1 = triangles(*,i)
      Lon3 = Lon1(tri1) & Lat3 = Lat1(tri1)
;### which average scheme is best?
;      Zb1   = avg(Zb(tri1)) ; average byte values
;      Zb1   = Zb(tri1(0)) ; faster than avg and as accurate? only small bar
      Zb1 =  doByteScale([avg(Z2(tri1))],minZ1,maxZ1,Zsize,wBad,flipColor Bar,1)
      if (total(abs(pLon(tri1)-shift(pLon(tri1),1))) lt 0.1) and $
  (total(abs(pLat(tri1)-shift(pLat(tri1),1))) lt 0.1) then $
 polyfill, Lon3, Lat3, color=Zb1(0), noclip=0
```

```
    endfor
  endif else begin ; 2-dim
    Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 0) ; is1dim
  ; remove bad values but keep 2-dim arrays
    pAll = convert_coord(Lon1, Lat1, /data, /to_normal)
    pLon = pAll(0,*) & pLat = pAll(1,*)
    for i=1L,Zsize(1)-1 do for j=1L,Zsize(2)-1 do begin
; plot rhombus
      indices = [(j-1)*Zsize(1)+(i-1), (j-1)*Zsize(1)+i, $
   j*Zsize(1)+i, j*Zsize(1)+(i-1)]
;     Z3=Z2(indices) same as Z3 = [Z2(i-1,j-1), Z2(i,j-1), Z2(i,j), Z2(i-1,j)]
      Lon3 = Lon1(indices) & Lat3 = Lat1(indices) & Z3 = Z2(indices)
      pLon3 = pLon(indices) & pLat3 = pLat(indices)
; could split into 2 triangles with Zb1=Zb(i-1,j-1) and Zb1=Zb(i,j)
;      indices3a = [(j-1)*Zsize(1)+(i-1), (j-1)*Zsize(1)+i, j*Zsize(1)+i]
;      indices3b = [(j-1)*Zsize(1)+i, j*Zsize(1)+i, j*Zsize(1)+(i-1)]
;      p = convert_coord(Lon3, Lat3, /data, /to_normal)
       w = where((Zb(indices) le 0) or (Lon3 lt -180) or (Lon3 gt 180) or $
      (Lat3 lt -90) or (Lat3 gt 90) or $
  (abs(pLon3-shift(pLon3,1)) gt 0.1) or $
  (abs(pLat3-shift(pLat3,1)) gt 0.1), wc)
      if (wc le 0) then begin
; ### select one corner value or average?
;       Zb1 = Zb(i,j)
       Zb1 = doByteScale([avg(Z3)], minZ1,maxZ1, Zsize, wBad, flipColorBar, 1)
            ; is1dim
       polyfill, Lon3, Lat3, color=Zb1(0), noclip=0 ;, thick=3
      endif
    endfor
  endelse ; 2-dim
end ; Polyfill method

"MA": begin ; Map_image method
; remove bad values which reduces to 1-dim arrays
  if (wBadc gt 0) then begin
    Lat1 = Lat1(wGood)  &  Lon1 = Lon1(wGood)  &  Z2 = Z2(wGood)
  endif
  if keyword_set(slow) then begin & quintic = 1 & bilinear = 1 & endif $
        else begin & quintic = 0 & bilinear = 0 & endelse
  triangulate, Lon1, Lat1, triangles, bounds, sphere=tridata, fvalue=Z2, $
 /degrees, repeats = repeats
;  Z2 = transpose(Z2) ; #### Z2 is Lon by Lat?
;###  gridSpacing = [maxLon - minLon, maxLat - minLat] / float([nLon-1, nLat-1])
  gridSpacing = [1., 1.] ; [deltaLon, deltaLat] in degrees
  if debug then message, /inform, $
 string('DEBUG: gridSpacing (Lon, Lat)= ', gridSpacing)
; #### could not specify gridSpacing and revLimit?
  if (quintic eq 0) then begin
```

```
    mapimage = trigrid (Z2, gridSpacing, revLimit, sphere=tridata, /degrees)
; max_Value=maxZ1, min_Value=minZ1, missing=fillZ ; minZ1
  endif else begin
    mapimage = trigrid (Z2, gridSpacing, revLimit, sphere=tridata, /degrees, $
 /quintic, extrapolate=bounds)
; max_Value=maxZ1, min_Value=minZ1, missing=fillZ ; minZ1
  endelse
;###### or ,missing=!p.background
;maxLon1 = maxLon > (-180) < 180 & minLon1 = minLon > (-180) < 180
;maxLat1 = maxLat > (-90) < 90   & minLat1 = minLat > (-90) < 90
maxLon1 = maxLon & minLon1 = minLon
maxLat1 = maxLat & minLat1 = minLat
;#### is following true?
;if (maxLon1 lt 0) then maxLon1 = maxLon1 + 360. ; map_image seems to want 0:360
;if (minLon1 lt 0) then minLon1 = minLon1 + 360. ; map_image seems to want 0:360
;if (centerLon lt 0) then centerLon = centerLon + 360. ; map_image seems to
want 0:360
;#### need to do same thing for min/maxLat1 ?; need to do this for centerlat/lon
; this command warps the input image to the defined mapping
; ### mapimage = transpose(mapimage)
maxLon1 = revLimit(2) & minLon1 = revLimit(0) ; based on edges of mapimage
maxLat1 = revLimit(3) & minLat1 = revLimit(1)
  image = map_image(mapimage, startx, starty, xWinsize, yWinsize, $
 latmin=minLat1, latmax=maxLat1, lonmin=minLon1, lonmax=maxLon1, $
 bilinear=bilinear, /whole_map, missing=!p.background)
; compress=1)  ; $ ; compress def=4
; scale=0.05, $ ; Postscript scaling, def=0.02
; max_value=maxZ1, min_Value=minZ1)
;### add scale keywords???
  Zb = doByteScale(image, minZ1, maxZ1, Zsize, wBad, flipColorBar, 1) ; is1dim
  tv, Zb, startx, starty, xsize=xWinsize, ysize=yWinsize
end ; Map_image method

"DI": begin ; Dilate method
; technique copied from imagemap.pro from liam.gumley@ssec.wisc.edu CIMSS/SSEC,
; 26-JUL-1996  <http://cimss.ssec.wisc.edu/~gumley/index.html>
; suggested by Hermann Mannstein (h.mannstein@dlr.de)
; remove bad values which reduces to 1-dim arrays
  if (wBadc gt 0) then begin
    Lat1 = Lat1(wGood)  &  Lon1 = Lon1(wGood)  &  Z2 = Z2(wGood)
  endif
  maxLon1 = revLimit(2) & minLon1 = revLimit(0) ; based on edges of mapimage
  maxLat1 = revLimit(3) & minLat1 = revLimit(1)
  Zb = doByteScale(Z2, minZ1, maxZ1, Zsize, wBad, flipColorBar, 1) ; is1dim
  px = !x.window*!d.x_size
  py = !y.window*!d.y_size
  xWinsize = px(1)-px(0)
  yWinsize = py(1)-py(0)
```

```
;- create resampled byte image
  p = convert_coord(Lon1, Lat1, /data, /to_device )
  newimage = bytarr( !d.x_size, !d.y_size )
  newimage( p( 0, * ), p( 1, * ) ) = Zb
;- get image coordinates of map projection corners
;  p = convert_coord( [minLon1,maxLon1], [minLat1,maxLat1], /data, /to_device )
;  if (p(0,0) gt 1.e+10) then p(0,0) = 0 ; off projection
;  if (p(1,0) gt 1.e+10) then p(1,0) = 0 ; off projection
;  if (p(0,1) gt 1.e+10) then p(0,1) = !d.x_size ; off projection
;  if (p(1,1) gt 1.e+10) then p(1,1) = !d.y_size ; off projection
;- extract portion of image corresponding to map limits
;  newimage = temporary( newimage( p(0,0):p(0,1), p(1,0):p(1,1) ) )
;- fill holes in resampled image
  fill = dilate( newimage, replicate( 1, 2, 2 ), /gray )
  loc = where( newimage eq 0, wc )
  if (wc gt 0) then newimage( loc ) = fill( loc )
  fill = 0
;- display resampled image
  tv, newimage, p( 0, 0 ), p( 1, 0 ), xsize=xWinsize, ysize=yWinsize
end ; Dilate method

else: begin
  msgText = 'Method unknown: ' + string(method)
  if doStatus then begin
    message, msgText, /info & status = -1L & return
  endif else message, msgText
end ; else case
endcase

if (method1 ne "TV") and (method1 ne "QU") then begin
  if not keyword_set(nogrid) then $
 map_grid,/label,color=mapColor,charsize=mapCharSize
  if keyword_set(continent) then map_continents, color=mapColor
endif ; not TV or Quick

if doColorBar then begin
  if (n_elements(ctitle) eq 0) then ctitle = ''
  if (n_elements(cCharSize) eq 0) then cCharSize = 0.
  offset = 0.01
  colorbar, cscale, ctitle, logZ=logZ, cCharSize=cCharSize, $
 position=[!x.window(1)+offset,      !y.window(0),$
    !x.window(1)+offset+0.03, !y.window(1)]
endif ; colorbar

;!p.Position = pPosition ; restore
!x.margin = xmargin
;!y.margin = ymargin
```

```
return
end ; auroral_image

Pro Colorbar, scale, title, logZ=logZ, position=position, cCharSize=cCharSize
; Terri Martin and Robert Candey
; 21 June 1993
; added ccharsize ; 1995 July 26
; added save on !x, !y, !p; 1995 Aug 2; BC
; reduced ncolors when not enough pixels; 1995 Sep 14 BC

; This procedure creates a colorbar for the right side of a spectrogram

common deviceTypeC, deviceType,file; required for inverting grayscale Postscript
xsave = !x & ysave = !y & zsave = !z & psave = !p

if (n_elements(position) ne 0) then positiont = position else $
  positiont = [!x.window(1)+0.01, !y.window(0), !x.window(1)+0.04, !y.window(1)]
; positiont = [0.9, 0.1, 0.93, 0.9]
if not keyword_set(logZ) then logZ=0
if (n_elements(title) eq 0) then title = ''
nColors = !d.n_colors-2. ; reserve black and white at ends of colorscale
colors = bindgen(nColors) + 1B
if (n_elements(deviceType) ne 0) then if (deviceType eq 2) then $
 colors = (!d.n_colors-1B) - colors ; invert grayscale for Postscript

fontsize = 1.0
if n_elements(cCharSize) le 0 then cCharSize = 0.
if cCharSize gt 0 then begin
  fontsize = cCharSize
endif else begin
; alternative if cCharSize is undefined or le 0
  if !p.charsize gt 0. then fontsize = !p.charsize
  if !y.charsize gt 0. then fontsize = !y.charsize * fontsize
  if (!p.multi(1)>!p.multi(2)) gt 2 then fontsize = fontsize/2.
endelse

plot, [0., 1.], [0., 1.], position = positiont, $
 /nodata, /noerase, xstyle = 4, ystyle = 1+4

if (abs(!x.window(1)-!x.window(0))*!d.x_size le 2) then begin
  message, 'Colorbar too narrow', /info
  !x = xsave & !y = ysave & !z = zsave & !p = psave ; restore original settings
  return
endif
colorStep =  ceil(float(ncolors)/(abs(!y.window(1)-!y.window(0))*!d.y_siz e)) > 1L
; could require 2 pixels per color; colorStep = colorStep * 2L
nSteps = fix(nColors / colorStep) < nColors
;if (!d.name eq 'PS')
```

```
if (!d.flags and 1L) then begin ; has scalable pixels (Postscript)
  colorStep = 1L
  nSteps = nColors
endif

for i = 0L, nSteps-1 do begin
  polyfill, [0.,1.,1.,0.], (i+[0.,0.,1.,1.])/nSteps, $
 color=colors(i*colorStep), noclip=0
endfor ; i

; replot so the box gets put back over the filled area
plot, [0., 1.], [0., 1.], position = positiont, $
 /nodata, /noerase, xstyle = 4, ystyle = 1+4
axis, yaxis = 1, ystyle = 1, yrange = scale, ytype=logZ, ycharsize=fontsize, $
 ytitle = title, ticklen = -0.02*0.78/0.04 ; adjust for narrow window

!x = xsave & !y = ysave & !z = zsave & !p = psave ; restore original settings
return
end ; colorbar
```