Subject: Re: How to do polar plots with logarithmic axis in radial coordinate?
Posted by Charlie Zender on Thu, 08 Feb 2001 17:34:10 GMT
View Forum Message <> Reply to Message

Thanks for the suggestion, I might steal your idea of using
arc hyperbolic sin instead of log....

Mirko Vukovic wrote:

> In article <3A8159F0.6BCE06BC@ncep.noaa.gov>,
> Paul van Delst <pvandelst@ncep.noaa.gov> wrote:
>
>> Charlie Zender wrote:
>>
>>> Craig Markwardt wrote:
>>>
>>>
>>>> Could you simply take the ALOG10() logarithm of the data before
>>>> plotting it? Easier to re-label the axis than re-invent the
>>>
> world...
>
>>> This would cause the radial coordinate to be negative-valued which
>>> would have unpleasant results. It's possible someone could get
>>> this method to work but I tried without success.
>>
>
> For cases of positive numbers with a huge range, I often use the arc
> hyperbolic sine function. It is approximately linear for arguments<1,
> and logarithmic for large arguments>1. I include it way at the end of
> the post (last two routines). I use it farily often, but never
> bothered to write and accompanying tick marking routine.
>
>
>> Hope some of this is helpful, although I have to admit, the fact that
>
> IDL doesn't have a
>
>> stock polar plotting routine that produces a circular graph with the
>
> radial and concentric
>
>> circle tickmark axes is a bit ridiculous. Farting about with /POLAR
>
> and AXIS and whatnot
>
>> is sort of like using OG to plot, x, y - and in the end you still end
>

```
>  up with
>
>>  Cartesian-like axes.
>
>
>  agreed.  In some of my applications, I use MAP for polar plotting.
>  I'll excerpt parts of the code, but you will need to modify it for your
>  applications.  The code is part of an object, so some variables are
>  stored as fields of SELF.  But that is all they are, variables.  If you
>  provide them, they do not have to be parts of an object.
>
>  To give you some ideas as to what is involved, the following set-up the
>  plot:
>  ; convert rmin and rmax (stored as vector in Radial Frame Limits) to
>  latitudes
>     LatRange=self->r2Lat(RadialFrameLimits)
>  ; convert min and max angle (can be 0 to 360) to longitude
>     LonRange=self.FrameLimits[[1,3]]*!radeg
>  ; store this as part of the object in which the whole things is done
>      self.DataLatLonRanges=[LatRange[0],LonRange[0],LatRange[1],L onRange
>  [1]]
>  ; rotate map so that 0 angle points to the right
>     Rotation=-90;+self.Orient*!radeg
>     ;; the map is plotted without the default border
>  ; put up the polar grid.  We can plot data over that, discussed below.
>     map_set,90,0,Rotation,/Azimuthal,/iso,/noborder, $
>       limit=self.DataLatLonRanges,NoErase=NoErase, $
>       _extra=rPropertiesKeywordList
>
>  Now this requires two routines for conversion from data to latitudes
>  and back:
>
>  function Polar_PlotFrame::R2Lat,R
>  ;@private
>  ;
>  ;function that converts radius to latitude.  This is used to translate
>  ;data into units that MAP understands.
>
>
>     Rmax=self.FrameLimits[2]
>     LatRange=self.LatRange
>
>     RelR=R/Rmax
>     Lat=RelR*(LatRange[1]-LatRange[0])+LatRange[0]
>
>     return,Lat
>  end
>
```

```
>  function Polar_PlotFrame::Lat2R,Lat
>  ;@private
>  ; function that converts from latitutde to radius
>
>    Rmax=self.FrameLimits[1]
>    LatRange=self.LatRange
>
>    RelR=(Lat-LatRange[0])/(LatRange[1]-LatRange[0])
>    R=RelR*Rmax
>
>    return,R
>  end
>
>  For these to work, I need this somewhere at the start of the program.
>  Thus the map will have the latitude range from 90 (radius 0) to 0.
>  (max radius, to be determined later)
>
>    self.LatRange = [90.,0.]
>
>  Finally, to plot the data, I do
>  ; convert coords from data to latitude
>     self.oPlotFrame-> AdjustCoords,*self.pIndependentVariable, $
>      *self.pDependentVariable,AngleCoord,RadialCoord
>
>  ; contour will work too!
>     plots,AngleCoord,RadialCoord, $
>      _extra=rPlotPropertiesKeywordList
>
>
>  And, finally, this needs the services of AdjustCoords:
>
>  pro Polar_PlotFrame::AdjustCoords,Phase,Mag,Lon,Lat
>  ; converts coordinates from angle and radius to longitude and
>  ; latitude.
>
>     ;; convert negative magnitude to positive, and correct angle
>     iNegMag =  where(Mag LT 0,cNegMag)
>     CorrPhase = Phase
>     CorrMag = Mag
>     IF cNegMag NE 0 THEN BEGIN
>        CorrPhase[iNegMag] = CorrPhase[iNegMag]+!pi
>     CorrMag[iNegMag] = -CorrMag[iNegMag]
>     ENDIF
>     CorrPhase = CorrPhase MOD !twopi
>
>     ;; radius to latitude
>     Lat = self->r2lat(CorrMag)
>     Lon = CorrPhase*!radeg
```

```
>
>  return
>  END
>
>
>
>
>  Now for the routines for arc sinh.
>
>  Here is the asinh, that can handle scalars and vectors
>  ;+
>  ; return the inverse hyperbolic sine of the argument.  The calculation
>  is
>  ; performed in double precision because of the addition of 1 under the
>  ; square root.  It might be better to test for size and return the
>  ; approximate value of the sqarre root.
>  ;
>  ; Written by Mirko Vukovic, around 1990
>  ;-
>   FUNCTION ASINH,ARG
>
>  ;create the result array
>  type=size(arg)
>  type_res = type
>  dim = type(0)
>  type_res(dim+2) = 32
>  res = m$replicate(type_res)
>  ;fill it in with results
>  index1 = where (abs(arg) lt 1.d3,count)
>  if count ne 0  then  $
>   res(index1) = alog(arg(index1)+sqrt(arg(index1)^2+1.d00))
>  index2 = where(arg le -1.d3,count)
>  if count ne 0 then $
>   res(index2) = -alog(-2.*arg(index2))
>  index3 = where(arg ge 1.d3,count)
>  if count ne 0 then  $
>   res(index3) = alog(2.*arg(index3))
>
>
>  ; bring result to original type of the argument
>  if type(dim+2) ne 32 then res=float(res)
>  return,res
>
>
>  end
>
>  It requires mv_replicate, similar to IDL's replicate, but can handle
>  scalars (there is probably a better way)
```

```
> ;+
> ; NAME:
> ;       MV_REPLICATE
> ;
> ; PURPOSE:
> ;       To emulate the REPLICATE function of the old version of IDL
> ;
> ; CATEGORY:
> ;       Variable massaging
> ;
> ; CALLING SEQUENCE:
> ;       result=MV_REPLICATE(INFO,type=type)
> ;
> ; INPUTS:
> ;       INFO - a vector, of SIZE-like properties
> ;
> ; OPTIONAL INPUT PARAMETERS:
> ;       None
> ;
> ; KEYWORD PARAMETERS:
> ;       TYPE -- (optional) integer assigns the type of the variable.
> If not
> ;       present, the type present in INFO is assigned to the varible.
> ;       The value of type should be
> ;  1: binary
> ;  2: integer
> ;  3: long integer
> ;  4: floating
> ;  5: double precision
> ;  6: complex
> ;
> ;
> ; OUTPUTS:
> ;       RESULT - a variable specified according to INFO
> ;
> ; OPTIONAL OUTPUT PARAMETERS:
> ;       None
> ;
> ; COMMON BLOCKS:
> ;       None
> ;
> ; SIDE EFFECTS:
> ;       If INFO does not have the total number of elements in the
> ;       variable, that is added to it.
> ;
> ; RESTRICTIONS:
> ;       None
> ;
```

```
>  ; PROCEDURE:
>  ;       Straightforward.  Checking is done to see if the total number
>  of
>  ;       elements in the variable is present in INFO.  If not, it is
>  ;       calculated and added to it.
>  ;
>  ; MODIFICATION HISTORY:
>  ;       Written and performed by Mirko Vukovic, sometimes around 1990
>  ;
>  ;-
>  function mv_replicate,info,type=type
>
>  nod = info(0)
>  infod = n_elements(info)
>
>  ; make the INFO array complete
>  if infod ne nod+3 then begin  ; total no. of elemets is
>  missing
>   t=1
>   for i=1,nod do t=t*info(i)
>   info=[info,t]
>  endif
>
>  if info(0) ne 0 then begin   ; this is for an array
>          if keyword_set(type) then res=make_array(size=info,type=type) $
>          else res = make_array(size=info)
>  endif else begin
>   if keyword_set(type) then begin
>   case type of ; and this for a scalar, info(1) has variable type
>  info
>   0:  begin
>    print, 'MV_REPLICATE: cannot make variable of undefined
>  type.'
>    stop
>    end
>   1: res=0b
>   2: res=0
>   3: res=long(0)
>   4: res=0.
>   5: res=0.d00
>   6: res=complex(0.,0.)
>   else:  begin
>    print, 'MV_REPLICATE: cannot make structure or string
>  variable.'
>    stop
>    end
>   endcase
>   endif else begin
```

```
>   case info(1) of ; and this for a scalar, info(1) has variable
>  type info
>   0:  begin
>    print, 'MV_REPLICATE: cannot make variable of undefined
>  type.'
>    stop
>    end
>   1: res=0b
>   2: res=0
>   3: res=long(0)
>   4: res=0.
>   5: res=0.d00
>   6: res=complex(0.,0.)
>   else:  begin
>    print, 'MV_REPLICATE: cannot make structure or string
>  variable.'
>    stop
>    end
>   endcase
>   endelse
>  endelse
>  return,res
>  end
>
>
>  Sent via Deja.com
>  http://www.deja.com/
```

--
Charlie Zender zender@uci.edu (949) 824-2987/FAX-3256, Department of
Earth System Science, University of California, Irvine CA 92697-3100