

Subject: Re: sample/empirical variogram calculation
Posted by [james-a-roo](#) on Thu, 06 Nov 2008 05:54:24 GMT
[View Forum Message](#) <> [Reply to Message](#)

Even though this posting generated zero interest in the last week, there were two old posts on the topic and I have a pet peeve when people don't post their answers to their questions. So here's the code i wrote. I'll make another post following this one with a few crude examples. I'm fairly confident I have the results correct, though I havent run numbers against any other routine, say in R.

Right now, this routine accepts 1-3 dimensional data. Bins in each dimension are fairly flexibly specified either as regular or irregular bins. Regular bins are efficiently processed directly by JD Smith's `hist_nd`, irregular bins are transformed to regular indices via `value_locate` before histogramming. I think there's just no way around the looping nature of this beast. I first wrote a code to avoid that, but it won't function for input arrays over even a modest size as it constructs several $n \times n$ matrices from n input points. The most obvious drawback with this currently is that angle bins (spherical coords using θ and ϕ) are not permitted to be rotated relative to the data. I think this should be really easy to implement, just add or subtract the desired angle from the data once it's transformed to spherical coordinates, but I haven't done that and don't have the time to check it for those purposes right now.

The main concern with this code is improving the efficiency of what's happening inside the loop, as the number of laps is potentially high. I'm not the best versed in what kind of array manipulations and declarations are fastest and most memory efficient. So there's probably some easy room for speed-up. I also didnt see any way to calculate the largest necessary bin in radius in advance of the loop, these calculations slow the looping in the case that the largest bin bounds in radius are not supplied.

I hope someone gets use out of this! If you are seeing any highly desirable feautres that appear lacking, do post or email me to see if I may have already added them, esp if some time has gone by.

```
; NAME: variogram
;
;
; PURPOSE:
; Calculate the EMPIRICAL variogram of a large data set in 1-3
; dimensions for arbitrary
; distance bins. If 2D, also
```

```

; an arbitrary number of bins in [0,180) in the x-y plane. If 3D, also
an arbitrary number of
; bins in the azimuthal direction.
; Note: 3D is not fully implemented as of right now.
; Note: The euclidean norm is currently the measure of distance
between 2 points.
;
; CATEGORY: spatial stats
;
; CALLING SEQUENCE:
;
; INPUTS:
; x - cartesian, primary dimension of location of observations, length
nobs
; y - cartesian, secondary dimension of location of obs, length nobs.
; z - cartesian, tertiary dimension of location of obs, length nobs.
; v - the value of the observation at the corresponding location,
length nobs.

; OPTIONAL INPUTS:
; radiusbins -
;   Default or no input returns [0,maxrad+fudge,1] where 0 is the
start of the smallest bin and 1
;   is the radius bin width. Maxrad+fudge is the lower bound of the
last bin which contains the
;   largest radius found in the data.
;   A scalar implies the binwidth which is applied from 0 to include
the largest radius found
;   in the data. In this case, [0,maxrad+fudge,binwidth] is returned.
;   A three element vector [min,max,binwidth] specifies the lower
(inclusive) bounds of the first and
;   last bin and the width of the bins intervening, which requires
that (max-min) is an integer.
;   Any other number of elements implies the bin starts and ends and
may be completely irregular.
;   Each bin start is included in that bin and the final value is also
included in the final bin.
;   In this last case, elements must be positive and in monotonically
increasing order. Example:
;   eg: [0,1,!pi,8] -> bins: [0,1), [1,!pi), [!pi,8]
; thetabins - For 2D or 3D data.
;   Undefined implies 1 bin, isotropic calculation.
;   Scalar value implies the number of bins in [0,!pi).
;   More than 1 element implies bin boundaries. Elements must be
positive, monotonically increasing,
;   and in the interval [-!pi/2,!pi/2). If the first and last values
are not
;   -!pi/2 and !pi/2, respectively, then these are appended to the

```

appropriate ends of the array.

- ; The returned value is the list of lower bounds (inclusive) on each bin starting with 0 and
- ; going through the $\pi/2$.
- ; phibins - For 3D data.
- ; Same as thetabins but for 3D data. This is the azimuthal direction (as measured from
- ; the z axis).
- ; Example: $[45,90,135]*\pi/180 \rightarrow [0,45,90,135,180]*\pi/180 \rightarrow$
- ; $[0,45), [45,90), [90,135), [135,180)$ all $*\pi/180$.
- ; subsetinds - By default the empirical variogram is calculated as the expected value in each bin
- ; (bins are defined by the spatial partitioning of the above 3 parameters) over ALL
- ; supplied observations. If the expectation is only desired over some subset of input points,
- ; these are the indices of these observation n-tuplets corresponding to the location in the
- ; observation vectors $x, (y, (z)), v$.
- ; double - Indicates calculations to be done in double precision. As histogram is a "razor's" edge,
- ; inputs need to be in double precision for this to work. A message will alert the user if this is
- ; not the case for any inputs involved in the calculation.
- ; semi - Returns the semivariogram=variogram/2 (=variance of a stationary random field w/o spatial dependence).
- ; KEYWORD PARAMETERS:
- ;
- ; OUTPUTS:
- ; return = [nradiusbins] , [nradiusbins,nthetabins],
- [nradiusbins,nthetabins,nphibins]
- ; depending on dimensions of input.
- ; OPTIONAL OUTPUTS:
- ;
- ; COMMON BLOCKS:
- ;
- ; EXTERNAL CALLS:
- ;
- ; SIDE EFFECTS:
- ;
- ; RESTRICTIONS:
- ;
- ; PROCEDURE:
- ;
- ; EXAMPLE:
- ; $x=[0,1,1,0,.2,1,0,-.2,1.8]$
- ; $y=[0,0,1,1,.2,-1,-1,-.2,0]$
- ; $v=[0,1,1,1,.1,3,3,-.2,1.6]$

```

; radiusbins=.8
; thetabins=2
;
r=variogram_big(x=x,y=y,radiusbins=radiusbins,thetabins=thetabins,v=v);
; more examples are posted with the code on the google groups site
; http://groups.google.com/group/comp.lang.idl-pvwave
;
; REFERENCES: see wikipedia: http://en.wikipedia.org/wiki/Variogram
;
; MODIFICATION HISTORY:
; 10/30/08 - jlm
; Please post improvement, corrections, modifications back to the
google page from whence this may have come.
;
;*****
; Use this code entirely at your own risk. No warranties of any sort,
express, implied,
; or inferred regarding anything are offered, including software,
hardware, user sanity.
; Nothing is offered except the code.
; The only requirement of use or reproduction is that you keep the
author credits in
; tact and contribute to the improvement when possible.
;*****
;-
function variogram, x=x, y=y, z=z, v=v, semi=semi, $
    radiusbins=radiusbins, thetabins=thetabins,
phibins=phibins, $
    subset_inds=subset_inds, double=double,
cumulative_hist=cumulative_hist

nx=n_elements(x)          ; x coordinates
nv=n_elements(v)          ; value at coordinates
IF keyword_set(y) THEN ny=n_elements(y) ; y coordinates
IF keyword_set(z) THEN nz=n_elements(z) ; y coordinates
message=0
IF nx NE nv THEN message=1
IF keyword_set(y) THEN IF nx NE ny THEN message=1
IF keyword_set(z) THEN IF nx NE nz THEN message=1
IF message THEN message,'Input data vectors must have same lengths'
nobs=nx
undefine,nx,ny,nz,nv

;; Type defaults to float if double is not set.
type = keyword_set(double) OR size(v,/type) EQ 5 ? 5L : 4L
badtypes=0
IF size(x,/type) NE type THEN badtypes=badtypes+1
IF keyword_set(y) THEN IF size(x,/type) NE type THEN badtypes=badtypes

```

```

+1
IF keyword_set(z) THEN IF size(x,/type) NE type THEN badtypes=badtypes
+1
IF keyword_set(v) THEN IF size(x,/type) NE type THEN badtypes=badtypes
+1
IF keyword_set(radiusbins) THEN IF size(x,/type) NE type THEN
badtypes=badtypes+1
IF keyword_set(thetabins) THEN IF size(x,/type) NE type THEN
badtypes=badtypes+1
IF keyword_set(phibins) THEN IF size(x,/type) NE type THEN
badtypes=badtypes+1
IF badtypes GT 0 THEN message,'There are '+string(badtypes,'(l1.1)')
+' input variables of the wrong type.'

```

```

undefine,cumulative_hist

```

```

zero = type EQ 4 ? 0. : 0.D
one = type EQ 4 ? 1. : 1.D
negone = type EQ 4 ? -1. : -1.D
two = type EQ 4 ? 2. : 2.D
pi = type EQ 4 ? !pi : !dpi
pi2 = type EQ 4 ? !pi/2. : !dpi/2.D
precision=(machar(double=type-4)).eps

```

```

;; Treat the radiusbins keyword.

```

```

;; Note that we dont necessarily know the number of bins in radius a
priori.

```

```

IF ~keyword_set(radiusbins) OR n_elements(radiusbins) EQ 1 THEN
BEGIN

```

```

    radius_bounds_specified=0

```

```

    radiusmin=zero

```

```

    radiusmax=zero ;; must initialize this less than any possible
maximum radius

```

```

    ;; If no bin iformation was specified, use binwidth of 1

```

```

    radiusbinwidth= ~keyword_set(radiusbins) ? one : radiusbins

```

```

    radius_regular=1

```

```

ENDIF ELSE BEGIN

```

```

    radius_bounds_specified=1

```

```

    radiusmin=radiusbins[0]

```

```

    radiusmax=radiusbins[n_elements(radiusbins)-1]

```

```

; radiusbins = radiusbins

```

```

    nradiusbins=n_elements(radiusbins)-1

```

```

    ;; Check if the passed array is evenly spaced

```

```

    radius_regular = $

```

```

    min( (radiusbins[1]-radiusbins[0])-(radiusbins[1:nradiusbins]-
radiusbins[0:nradiusbins-1]) LT precision )

```

```

    IF radius_regular THEN radiusbinwidth = radiusbins[1]-radiusbins[0]

```

```

ELSE BEGIN

```

```

    message, 'Bin spacing in the radius dimension is irregular. This
will greatly slow the variogram '+$
    'calculation. Consider a transform beforehand to regular
spacing, if possible.',/info
;; If irregular, there will be a "coordinate" transform on the
data which will give the following bounds
    radiusmin=zero
    radiusmax=nradiusbins
    radiusbinwidth=one
ENDELSE
ENDELSE
IF radiusmin GT 0 THEN message,'*** Warning: The lowest radius bin is
greater than zero',/info

;; If 2D or 3D get the xy plane bins, thetabins
IF keyword_set(y) THEN BEGIN
    thetamin=negone*pi2
    thetamax=pi2
    IF ~keyword_set(thetabins) THEN BEGIN
        nthetabins=1
        thetabinwidth=pi
        thetabins= type EQ 4 ? findgen(2)*thetabinwidth-pi2 :
dindgen(2)*thetabinwidth-pi2
        theta_regular=1
    ENDIF ELSE BEGIN
        IF n_elements(thetabins) EQ 1 THEN BEGIN
            nthetabins=fix(ceil(thetabins[0]),type=type) ;; make it an
"integer" in case it's not already.
            thetabinwidth=pi/nthetabins
            thetabins = type EQ 4 ? findgen(nthetabins+1)*thetabinwidth-
pi2 : dindgen(nthetabins+1)*thetabinwidth-pi2
            theta_regular=1
        ENDIF ELSE BEGIN
            ;; check that bins are within bounds and monotonically
increasing.
            IF max(abs(thetabins)) GT pi2 OR max(sort(thetabins)-
indgen(n_elements(thetabins))) NE 0 THEN $
                message,'*** Warning: Supplied thetabins is not of the
required form.'
            IF abs(thetabins[0]) NE pi2 THEN thetabins=[negone*pi2,
thetabins]
            IF thetabins[n_elements(thetabins)-1] NE pi2 THEN
thetabins=[thetabins,pi2]
            nthetabins=n_elements(thetabins)-1
            ;; Check if the passed array was evenly spaced
            theta_regular = $
                min( abs((thetabins[1]-thetabins[0])-(thetabins[1:nthetabins]-
thetabins[0:nthetabins-1])) LE precision )

```

```

    IF theta_regular THEN thetabinwidth = thetabins[1]-thetabins[0]
ELSE BEGIN
    message, 'Bin spacing in the theta dimension is irregular.
This will greatly slow the variogram '+$
    'calculation. Consider a transform beforehand to
regular spacing, if possible.',/info
    ;; If irregular, there will be a "coordinate" transform on
the data which will give the follwing bounds
    thetamin=zero
    thetamax=nthetabins
    thetabinwidth=one
    ENDELSE
    ENDELSE
    ENDELSE
ENDIF

```

```

;; If 3D get the azimuthal bins, phibins
IF keyword_set(z) THEN BEGIN
    phimin=zero
    phimax=pi
    IF ~keyword_set(phibins) THEN BEGIN
        nphibins=1
        phibinwidth=fix(pi,type=type)
        phibins= type EQ 4 ? findgen( 2 )*thetabinwidth :
dindgen( 2 )*thetabinwidth
        phi_regular=1
    ENDIF ELSE BEGIN
        IF n_elements(phibins) EQ 1 THEN BEGIN
            nphibins=fix(ceil(phibins[0]),type=type) ;; make it an integer
in case it's not already.
            phibinwidth=pi/phibins
            phibins = type EQ 4 ? findgen(nphibins+1)*phibinwidth :
dindgen(nphibins+1)*phibinwidth
            phi_regular=1
        ENDIF ELSE BEGIN
            ;; check that bins are within bounds and monotonically
increasing
            IF min(phibins) LT 0 OR max(phibins) GT pi OR $
max(sort(phibins)-indgen(n_elements(phibins))) NE 0 THEN $
            message, '*** Warning: Supplied phibins is not of the
required form.'
            IF phibins[0] NE zero THEN phibins=[zero,phibins]
            IF phibins[n_elements(phibins)-1] NE pi THEN
phibins=[phibins,pi]
            nphibins=n_elements(phibins)-1
            ;; Check if the passed array was evenly spaced
            phi_regular = min( abs((phibins[1]-phibins[0])-
(phibins[1:nphibins]-phibins[0:nphibins-1])) LE precision )

```

```

    IF phi_regular THEN phibinwidth = phibins[1]-phibins[0] ELSE
BEGIN
    ;; If irregular, there will be a "coordinate" transform on
the data which will give the following bounds
    message, 'Bin spacing in the phi dimension is irregular. This
will greatly slow the variogram '+$
    'calculation. Consider a transform beforehand to
regular spacing, if possible.',/info
    phimin=zero
    phimax=nphibins
    phibinwidth=one
    ENDELSE
    ENDELSE
    ENDELSE
ENDIF

;; Set up parameters outside the loop.
;; The following number (inspired by chmod) describes which
dimensions have regular bins.
bins_regular=0
IF radius_regular THEN bins_regular=bins_regular+1
IF keyword_set(y) THEN IF theta_regular THEN bins_regular=bins_regular
+2
IF keyword_set(z) THEN IF phi_regular THEN bins_regular=bins_regular
+4
;; Flag here if a certain dimension needs a coordinate transform,
reduce the # of ops in the loop.
radius_xform = max(bins_regular EQ [0, 2, 4, 6]) ; 1 if any is true
0 if none are <-> OR
theta_xform = keyword_set(y) ? max(bins_regular EQ [0, 1, 4, 5]) : 0
phi_xform = keyword_set(z) ? max(bins_regular EQ [0, 1, 2, 3]) : 0
;; Might as well.
ndims= keyword_set(x) + keyword_set(y) + keyword_set(z)

CASE ndims OF
1: BEGIN
    min = [radiusmin]
    max = [radiusmax] ;; radiusmax may be changing below
    binsize = [radiusbinwidth]
    END
2: BEGIN
    min = [radiusmin, thetamin]
    max = [radiusmax, thetamax] ;; radiusmax may be changing below
    binsize = [radiusbinwidth, thetabinwidth]
    END
3: BEGIN
    min = [radiusmin, thetamin, phimin]
    max = [radiusmax, thetamax, phimax] ;; radiusmax may be changing

```

below

```
    binsize = [radiusbinwidth, thetabinwidth, phibinwidth]
  END
ENDCASE

IF ~keyword_set(subset_inds) THEN subset_inds=lindgen(nobs) ELSE $
  IF max(subset_inds) GE nobs THEN message,'The passed subset_inds is
out of range of the input data.'
  nss_inds=n_elements(subset_inds)

;; loop over all points
FOR ss=0L,nss_inds-1 DO BEGIN

  xdists = x-x[ss]
  ydists = keyword_set(y) ? y-y[ss] : zero ;these are just dummies if
not defined.
  zdists = keyword_set(z) ? z-z[ss] : zero

  ;; *** Go to spherical coords: radius, theta, phi ***
  ;; Get the (euclidean) vector distances/radii
  radius=sqrt( xdists^two + ydists^two + zdists^two )
  ;; Get the theta vector directions if 2D
  IF keyword_set(y) THEN BEGIN
    yonx=temporary(ydists)/temporary(xdists)
    theta=atan( yonx ) ;; result in [-pi/2,pi/2] rads (crazy!)
    ;; When the divisor is zero, the point lies along 0 radians
    (straight up or down).
    wh0rad=where( finite(yonx) EQ 0 )
    IF wh0rad[0] NE -1 THEN theta[wh0rad]= negone*pi2 ;; want pts in
[-pi/2,pi/2) rads
  ENDIF
  ;; Get the phi (azimuthal) vector directions if 3D
  IF keyword_set(z) THEN BEGIN
    phi=acos( zdists/radius ) ;; result in [0, pi]
rads
    wh0rad=where( finite(phi) EQ 0 )
    IF wh0rad[0] NE -1 THEN phi[wh0rad]= pi2 ;; dosent really matter
where this goes, rad=0 is a spec case.
    phi= phi MOD pi ;; result in [0, pi)
rads
  ENDIF

  ;; *** Convert any data with irregular bins to regular bins ***
  ;; This is the afore mentioned "coordinate" transform, bins
boundaries have already been adjusted.
  IF radius_xform THEN radius=value_locate( radiusbins, radius )
  IF theta_xform THEN theta=value_locate( thetabins, theta )
  IF phi_xform THEN phi=value_locate( phibins, phi )
```

```

;; Assemble data
CASE ndims OF
  1: histin = transpose(radius)
  2: histin = [ transpose(radius), transpose(theta) ]
  3: histin = [ transpose(radius), transpose(theta),
transpose(phi) ]
ENDCASE

;; If a larger radius is found
IF max(radius) GT max[0] THEN BEGIN
  IF radius_bounds_specified THEN BEGIN
    IF max(radiusbins) LT max(radius) THEN $
      message, '*** Warning: Supplied radiusbins do not include all
distances found in the data.', /info
    ENDIF ELSE BEGIN
      ;; If the radius bounds arent fixed, we have to grow smaller
cumulative_histo and variogram
      ;; to accomodate
      newnbins=fix(ceil(max(radius)/radiusbinwidth),type=type)
      oldnbins=fix(ceil(max[0]/radiusbinwidth),type=type)
      nnewbins=newnbins-oldnbins
      max[0]= fix(newnbins,type=type)*radiusbinwidth
      CASE ndims OF
        1: add_dims=[nnewbins+1]
        2: add_dims=[nnewbins+1,nthetabins+1]
        3: add_dims=[nnewbins+1,nthetabins+1,nphibins+1]
      ENDCASE
      cumulative_hist= n_elements(cumulative_hist) EQ 0 ?
make_array( add_dims, type=3) : $
      [cumulative_hist, make_array( add_dims,
type=3) ]
      variogram=n_elements(variogram) EQ 0 ? make_array( add_dims,
type=type) : $
      [variogram, make_array( add_dims, type=type) ]
    ENDELSE
  ENDIF

;; *** Histogram! Thanks JD. ***
hist=hist_nd( histin, binsize, min=min, max=max, rev=ri )
;; Accumulate histogram
IF n_elements(cumulative_hist) EQ 0 THEN cumulative_hist=hist ELSE
$
  cumulative_hist=cumulative_hist+hist

;; *** Calculate the squared difference of the values ***
vdists2=(v-v[ss])^two
;; Put the vdists2 in the right bins with reverse indices

```

```

variogramtemp = fix(hist*0.,type=type)
whistne0=where(hist NE 0)
FOR ww=0,n_elements(whistne0)-1 DO $
    variogramtemp[whistne0[ww]] =
total(vdists2[ ri[ ri[whistne0[ww]]:ri[whistne0[ww]+1]-1 ] ])
;; This is really a cumulative vdists2 in each bin. Outside the
loop, the expected value
;; is found by dividing by the cumulative_hist.
variogram = n_elements(variogram) EQ 0 ? variogramtemp : variogram
+variogramtemp

```

```

;; If there is a bin containing 0 radius, we need to double count
every pt-to-self vdist2=0 (2)
;; in ALL directional bins (1).
;; 1) We need this over all bins for some continuity in the nugget
over directions. For instance,
;; if the bins contained only pts with zero spatial separation,
there really is no difference
;; in direction.
;; 2) Because symmetry between distinct points is double counting
all pt-to-pt sq differences while
;; only single counting pt-to-self sq differences, we need to
double count pt-to-self sq
;; differences. Think of the variance of the process and the
nugget effect, if there
;; are points at the same location with positive sqdifferences
(vdist2) then these will be
;; weighted twice as strongly in the nugget that point-to-self
zeros.

```

```

IF min[0] EQ 0 THEN BEGIN
CASE ndims OF
1: cumulative_hist[0]=cumulative_hist[0]+1
2: cumulative_hist[0,*]=cumulative_hist[0,*]+1
3: cumulative_hist[0*,*]=cumulative_hist[0*,*]+1
ENDCASE
ENDIF

```

```

ENDFOR ;; end looping over the subset of obs to accumulate the
variogram and histogram.

```

```

;; Calculate the variogram - the mean/expected value in each bin.
;; If a bin has no member, this returns a nan, which is desirable
IMO.

```

```

variogram= variogram / fix(cumulative_hist,type=type)
IF keyword_set(semi) THEN variogram=variogram/two

```

```

;; Trim off these end point bins in the likely or certain event they
are empty

```

```

size=size(variogram,/dim)
CASE ndims OF
  1: IF ~max( finite(variogram[size[0]-1]) ) THEN BEGIN
    variogram=variogram[0:size[0]-2] ;; likely
    cumulative_hist=cumulative_hist[0:size[0]-2]
  ENDIF
  2: BEGIN
    IF ~max( finite(variogram[size[0]-1,*]) ) THEN BEGIN
      variogram=variogram[0:size[0]-2,*] ;; likely
      cumulative_hist=cumulative_hist[0:size[0]-2,*]
    ENDIF
    variogram=variogram[:,0:size[1]-2] ;; by construction of the
angle bins, pi/2 -> -pi/2
    cumulative_hist=cumulative_hist[:,0:size[1]-2]
  END
  3: BEGIN
    IF ~max( finite(variogram[size[0]-1,*,*]) ) THEN BEGIN
      variogram=variogram[0:size[0]-2,*,*] ;; likely
      cumulative_hist=cumulative_hist[0:size[0]-2,*,*]
    ENDIF
    variogram=variogram[:,0:size[1]-2,*] ;; again, pi/2 -> -pi/2
    cumulative_hist=cumulative_hist[:,0:size[1]-2,*]
    variogram=variogram[:,*,0:size[2]-2] ;; here, by construction, pi
-> 0
    cumulative_hist=cumulative_hist[:,*,0:size[2]-2]
  END
ENDCASE

;; Create the output for radius bins
;; If radius is on an irregular grid, then radius_bounds_specified=1
and this wont happen as it's
;; already set. No need to worry about the "coordinate xform" bounds.
IF ~radius_bounds_specified THEN $
  radiusbins= findgen( (max[0]-min[0])/radiusbinwidth +1 ) *
radiusbinwidth + min[0]

return,variogram

END

```
