Subject: match_3d.pro
Posted by Grant K on Thu, 15 Oct 2009 09:38:29 GMT
View Forum Message <> Reply to Message

Hi,

I was looking for a way to do the nearest neighbour problem in 3D and didn't find anything pre-written that would do it. I've taken J.D. Smith's match_2d.pro and turned it into match_3d.pro and have pasted it below for people who might like to use it. I'm not an IDL expert... but it appears to work and the modifications were (thankfully) minimal.

If anyone wants to tidy/fix/streamline it to make an version for more public consumption then go for it.

cheers
Grant

--

```
;+
; NAME:
;       MATCH_3D
;
; PURPOSE:
;
;       Perform a match between two sets of 3D coordinates, finding
;       the closest coordinate match (in the Euclidean sense) to the
;       search set, within some search radius. This routine is simply
;       match_2d.pro with an extra dimension added, for those who need
;       to think about where something actually is, rather than where
;       it appears to be on the sky...
;
; CALLING SEQUENCE:
;       match=MATCH_3D
(x1,y1,z1,x2,y2,z2,search_radius,MATCH_DISTANCE=)
;
; INPUTS:
;
;       x1,y1,z1: The target list to search for matches, of length n1.
;
;       x2,y2,z2: The search list of length n2, to be searched for
;         matches to [x1,y1,z1].
;
;       search_radius: The search radius within which matches will be
;         found.  Only if the closest matching coordinate in
[x2,y2,z2]
```

```
;         is within the search radius will it be returned.  This is a
;         critical variable in tuning the resources required by
;         MATCH_3D.  It can be a scalar or two element vector (for
;         asymmetric radii).  See NOTES and WARNING.
;
; KEYWORD PARAMETERS:
;
;       MATCH_DISTANCE: On output, the distances between the matches
;         and the returned coordinate from the set [x1,y1,z1]
;         (<=search_radius), is returned.
;
; OUTPUTS:
;
;       match: A 1D vector of length n1 containing the indices of
x2,y2
;         and z2 for the closest match to each [x1,y1,z1], within the
;         search_radius. If no match was found within the search
;         radius, -1 is returned at that location.
;
; EXAMPLE:
;
;       n=100000
;       x1=randomu(sd,n) & y1=randomu(sd,n) & z1=randomu(sd,n)
;       x2=randomu(sd,n) & y2=randomu(sd,n) & z2=randomu(sd,n)
;       match=match_3d(x1,y1,z1,x2,y2,z2,.002,MATCH_DISTANCE=md)
;
; SEE ALSO:
;
;       HISTOGRAM, HIST_ND
;
; NOTES:
;
;       This match program uses HIST_ND to pre-bin the 3D search
;       coordinates based on position, within some canonical
;       search_radius.  See:
;
;           http://www.dfanning.com/code_tips/matchlists.html
;
;       for a discussion of its methods.  Of principle importance in
;       the efficiency and behavior of the match is the SEARCH_RADIUS
;       parameter.  This is the maximum radial distance within which a
;       match point must lie to be returned.  Here's a diagram
;       illustrating the problem and method in 2D (for 3D simply at an
;       extra layer of boxes on top, giving 8 bins).  For each target
point,
;       four separate bins each of width 2*search_radius are searched
;       among the binned search list, depending on its location within
;       the bin.
```

```
;
;
;       +----------+----------+
;       | t1     |       |
;       |        |       |
;       |        |       |
;       |        |       |
;       |        |       |
;       +----------+----------+   -
;       |        |       |   |
;       |        | o     |   |
;       |        |       |   |  2*search_radius
;       |        |       |   |
;       |        |       |   |
;       +----------+----------+   -
;                  t2
;
;
;       The point `t2' is the closest to the search point `o', but it
;       is not within the search_radius, therefore it is not
;       considered.  Instead, point `t1' is found (and discarded),
;       despite the fact that `t2' is closer.
;
;       Ideally, the search radius should be set to something useful
;       in terms of the match (e.g. positional uncertainty, etc.).
;       However, if the input target coordinates ([x2,y2]) span a
;       large range, it may be necessary to use a larger search
;       radius to avoid an excessively large number of bins.
;       Typically there will be an optimal search radius which is
;       fastest.  The tradoff is as follows: the larger the search
;       radius, the smaller the number of bins to search, but the
;       more search points must be considered per target point.  The
;       smaller the search radius, the smaller the number of search
;       points per bin, but the greater the number of bins.
;       Something like the median inter-point separation is probably
;       close to optimal.
;
;       Since lines of longitude converge towards the poles, a simple
;       trick to obtain roughly the same matching radius in both
;       longitude and latitude is to give an asymmetric
;       search_radius: [eps/cos(mean(latitude)),eps] for [longitude,
;       latitude].  This works only when the range of latitude is
;       small.
;
;
; WARNING:
;
;       Distance is evaluated in a strict Euclidean sense.  For
```

```
;       points on a sphere, the distance between two given
;       coordinates is *not* the Euclidean distance.  As an extreme
;       example, consider two points very near the N. pole, but on
;       opposite sides (one due E, one due W).  For small patches
;       away from the poles, this Euclidean assumption is
;       approximately valid, and the method works.  See NOTES above
;       for a tip regarding obtaining a (more) uniform match
;       criterion on the sphere.  For regions very near the pole, the
;       match could be made after suitably projecting all coordinates
;       using a map projection which alleviates this behavior.
;;
;
; MODIFICATION HISTORY:
;
;      Thur Oct 2009 (G Kennedy): take match_2d.pro and add z's to
;                           make match_3d.pro. I take no credit
;       for being clever and all responsibility if this routine
doesn't
;       work as advertised.
;
;      Wed Apr 22 17:56:26 2009 (J.D. Smith): Correctly account for
;         list ranges and "one bin off" matches.
;
;      Mon Jul 30 10:56:31 2007, J.D. Smith <jdsmith@as.arizona.edu>
;
;       Written.
;
;-
 ;#####################################################################
###################
;
; LICENSE
;
;  Copyright (C) 2007, 2009 J.D. Smith
;
;  This file is free software; you can redistribute it and/or modify
;  it under the terms of the GNU General Public License as published
;  by the Free Software Foundation; either version 2, or (at your
;  option) any later version.
;
;  This file is distributed in the hope that it will be useful, but
;  WITHOUT ANY WARRANTY; without even the implied warranty of
;  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;  General Public License for more details.
;
;  You should have received a copy of the GNU General Public License
;  along with this file; see the file COPYING.  If not, write to the
;  Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
```

```
;  Boston, MA 02110-1301, USA.
;
 ;#####################################################################
####################

function
 match_3d,x1,y1,z1,x2,y2,z2,search_radius,MATCH_DISTANCE=min_ dist
  bs = 2*search_radius          ;this is the smallest binsize allowed
  mx=[max(x2,MIN=mnx2),max(y2,MIN=mny2),max(z2,MIN=mnz2)]
  mn=[mnx2,mny2,mnz2]
  mn-=1.5*bs & mx+=1.5*bs       ;expand the range by (more than) the
bin size
  h =  hist_nd([1#x2,1#y2,1#z2],bs,REVERSE_INDICES=ri,MIN=mn,MAX=mx )
  d = size(h,/DIMENSIONS)

  ;; Bin location of X1,Y1,Z1 in the X2,Y2,Z2 grid
  xoff = 0.> (x1-mn[0])/bs[0] < (d[0]-1.)
  yoff = 0.> (y1-mn[1])/(n_elements(bs) gt 1?bs[1]:bs) < (d[1]-1.)
  zoff = 0.> (z1-mn[2])/(n_elements(bs) gt 1?bs[2]:bs) < (d[2]-1.)
  xbin = floor(xoff) &  ybin = floor(yoff) &  zbin = floor(zoff)
  bin = xbin + d[0]*(ybin + d[1]*zbin)        ;The 1D index of the bin
it's in

  ;; We must search 8 bins worth for closest match, depending on
  ;; location within bin (i.e. towards any of 8 quadrant directions
  ;;  ul, ur, ll, lr and with back/forward now too...).
  xoff = 1-2*((xoff-xbin) lt 0.5)         ;add bin left or right
  yoff = 1-2*((yoff-ybin) lt 0.5)         ;add bin down or up
  zoff = 1-2*((zoff-zbin) lt 0.5)         ;add bin forward or back

  n1=n_elements(x1)
  min_pos = make_array(n1,VALUE=-1L)
  min_dist = fltarr(n1,/NOZERO)

  rad2=search_radius^2

  for i=0,1 do begin ;; Loop over 4 bins in the correct quadrant
direction
     for j=0,1 do begin
        for g=0,1 do begin ;; and one more for z direction

           ;; One of 8 search bins for all the target points
           b = 0L>(bin+i*xoff+j*yoff*d[0]+g*zoff*d[1])<(d[0]*d[1]*d
[2]-1)

           ;; Dual HISTOGRAM method, loop by count in the search bins
           h2 = histogram(h[b],OMIN=om,REVERSE_INDICES=ri2)
```

```
        ;; Process all bins with the same repeats (>= 1) at a time
        for k=long(om eq 0),n_elements(h2)-1 do begin
          if h2[k] eq 0 then continue
          these_bins = ri2[ri2[k]:ri2[k+1]-1] ;bins with k+om
search points

          if k+om eq 1 then begin ; single point
            these_points = ri[ri[b[these_bins]]]
          endif else begin  ; range over k+om points, (n x k+om)
            targ=[h2[k],k+om]
            these_points = ri[ri[rebin(b[these_bins],targ,/
SAMPLE)]+ $
                              rebin(lindgen(1,k+om),targ,/
SAMPLE)]
            these_bins = rebin(temporary(these_bins),targ,/
SAMPLE)
          endelse

          ;; Closest distance squared within this quadrant's bin
          dist = (x2[these_points]-x1[these_bins])^2 + $
                 (y2[these_points]-y1[these_bins])^2 + $
                 (z2[these_points]-z1[these_bins])^2

          if k+om gt 1 then begin ;multiple points in bin: find
closest
            dist = min(dist,DIMENSION=2,p)
            these_points = these_points[p]   ;index of closest
point in bin
            these_bins = ri2[ri2[k]:ri2[k+1]-1] ;original bin
list
          endif

          ;; See if a minimum is already set there
          set = where(min_pos[these_bins] ge 0, nset, $
                      COMPLEMENT=unset,NCOMPLEMENT=nunset)

          if nset gt 0 then begin
            ;; Only update those where the new point is closer
            closer = where(dist[set] lt min_dist[these_bins
[set]], cnt)
            if cnt gt 0 then begin
              set = set[closer]
              min_pos[these_bins[set]] = these_points[set]
              min_dist[these_bins[set]] = dist[set]
            endif
          endif

          if nunset gt 0 then begin         ;; Nothing set, it's
```

```
closest by default
            wh=where(dist[unset] lt rad2,cnt) ;demand it's within
radius
            if cnt gt 0 then begin
              unset=unset[wh]
              min_pos[these_bins[unset]] = these_points[unset]
              min_dist[these_bins[unset]] = dist[unset]
            endif
          endif
        endfor
      endfor
    endfor
  endfor
  if arg_present(min_dist) then min_dist=sqrt(min_dist)
  return,min_pos
end
```

Subject: Re: match_3d.pro
Posted by Grant K on Thu, 03 Dec 2009 14:53:17 GMT
View Forum Message <> Reply to Message

there's a wee mistake in the bin location zoff line near the
beginning. it should be n_elements(bs) gt 2 (and not gt 1 as above)
since the given binsize array might be only two long for some weird
reason...
g

Subject: Re: match_3d.pro
Posted by d.poreh on Thu, 03 Dec 2009 15:01:02 GMT
View Forum Message <> Reply to Message

On 3 Dez., 06:53, Grant K <grantkenn...@gmail.com> wrote:
> there's a wee mistake in the bin location zoff line near the
> beginning. it should be n_elements(bs) gt 2 (and not gt 1 as above)
> since the given binsize array might be only two long for some weird
> reason...
> g

Thanks for sharing.
Could you please give us an example how it is works?
For instance if we have 2 Arrays like this:
Arr1=findgen(m1,n1,d1)
Arr2=findgen(m2,n2,d2)
Cheers
Dave
