Subject: Re: match_2d
Posted by Jeremy Bailin on Fri, 07 May 2010 12:30:04 GMT
View Forum Message <> Reply to Message

On May 7, 8:06 am, Dave Poreh <d.po...@gmail.com> wrote:
> On May 6, 11:11 am, wlandsman <wlands...@gmail.com> wrote:
>
>
>
>
>
>>> Actually there is no error, but I can't understand the procedure. At
>>> the end output is a 1D array (match). What I want is for each pixel of
>>> SAT array find the closest data from laser array (but I don't know how
>>> to do that).
>>> Cheers
>>> Dave
>
>> In your example
>
>> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)
>
>> 'match' gives the indices of x2,y2 which are the closest match to
>> x1,y1. So the closest point to x1[0],y1[0] is
>> x2[match[0]],y2[match[0]].
>
>> --Wayne
>
> Thank you. Another question: how could I find the other points of x2-
> y2 that surrounded in the search area (in this example 0.008) because
> maybe there is more than 1 point.
> Cheers
> Dave

You use a routine that I coincidentally wrote yesterday:
matchall_2d.pro, which is based heavily on JD's code (see below - I
haven't yet put it up in JBIU but I'll do it soon). Note that if your
coordinates are things like lat, lon and cover a larger area, 2D
distances won't be correct and you should use
withinsphrad_vec3d.pro... which is also apparently not yet in JBIU.
Jeez, I'm slacking. If you need it, ask me and I'll send it... and it
will hopefully be online soon too.

```
;+
; NAME:
;    MATCHALL_2D
;
```

```
; PURPOSE:
;    Determines which of a set of 2D coordinates are a given distance
from
;    each of a vector of points. Based on JD's MATCH_2D and my
WITHINSPHRAD_VEC3D
;    (in fact, it's basically WITHINSPHRAD_VEC3D tuned back down to a
;    Euclidean surface).
;
; CATEGORY:
;    Astro
;
; CALLING SEQUENCE:
;    Result = MATCHALL_2D(X1, Y1, X2, Y2, Distance, Nwithin)
;
; INPUTS:
;    X1:     Vector of X coordinates.
;
;    Y1:     Vector of Y coordinates.
;
;    X2:     Vector of X coordinates.
;
;    Y2:     Vector of Y coordinates.
;
;    Distance:  Maximum distance.
;
; OUTPUTS:
;    The function returns the list of indices of X2, Y2 that lie
within
;    Sphrad of each point X1,Y1. The format of the returned array is
;    similar to the REVERSE_INDICES array from HISTOGRAM: the indices
;    into X2,Y2 that are close enough to element i of X1,Y1 are
;    contained in Result[Result[i]:Result[i+1]-1] (note, however, that
;    these indices are not guaranteed to be sorted). If there are no
matches,
;    then Result[i] eq Result[i+1].
;
; OPTIONAL OUTPUTS:
;    Nwithin: A vector containing the number of matches for each of
X1,Y1.
;
; EXAMPLE:
;    Note that the routine is similar to finding
;      WHERE( (X2-X1[i])^2 + (Y2-Y1[i])^2 LE Distance^2, Nwithin)
;    for each element of X1 and Y1, but is much more efficient.
;
;    Shows which random points are within 0.1 of various coordinates:
;     FIXME
;
```

```
;    seed=43
;    nrandcoords = 5000l
;    xrand = 2. * RANDOMU(seed, nrandcoords) - 1.
;    yrand = 2. * RANDOMU(seed, nrandcoords) - 1.
;    xcoords = [0.25, 0.5, 0.75]
;    ycoords = [0.75, 0.5, 0.25]
;    ncoords = N_ELEMENTS(xcoords)
;    matches = MATCHALL_2D(xcoords, ycoords, xrand, yrand, 0.1,
nmatches)
;    PLOT, /ISO, PSYM=3, xrand, yrand
;    OPLOT, PSYM=1, COLOR=FSC_COLOR('blue'), xcoords, ycoords
;    OPLOT, PSYM=3, COLOR=FSC_COLOR('red'), xrand[matches[ncoords
+1:*]], $
;      yrand[matches[ncoords+1:*]]
;
; MODIFICATION HISTORY:
;    Written by:    Jeremy Bailin
;    10 June 2008   Public release in JBIU as WITHINSPHRAD
;    24 April 2009  Vectorized as WITHINSPHRAD_VEC
;    25 April 2009  Polished to improve memory use
;    9 May 2009     Radical efficiency re-write as WITHINSPHRAD_VEC3D
borrowing
;                  heavily from JD Smith's MATCH_2D
;    13 May 2009    Removed * from LHS index in final remapping for
speed
;    6 May 2010     Changed to MATCHALL_2D and just using Euclidean 2D
coordinates
;                   (add a bunch of stuff back in from MATCH_2D and
take out a bunch
;                   of angle stuff)
;-
function matchall_2d, x1, y1, x2, y2, distance, nwithin

if n_elements(x2) ne n_elements(y2) then $
  message, 'X2 and Y2 must have the same number of elements.'
if n_elements(x1) ne n_elements(y1) then $
  message, 'X1 and Y1 must have the same number of elements.'
if n_elements(distance) ne 1 then $
  message, 'Distance must contain one element.'

n1 = n_elements(x1)
n2 = n_elements(x2)

gridlen = 2.*distance
mx=[max(x2,min=mnx2),max(y2,min=mny2)]
mn=[mnx2,mny2]
mn-=1.5*gridlen
mx+=1.5*gridlen
```

```
h =  hist_nd([1#x2,1#y2],gridlen,reverse_indices=ri,min=mn,max=mx )
d = size(h,/dimen)

; bin locations of 1 in the 2 grid
xoff = 0. > (x1-mn[0])/gridlen[0] < (d[0]-1.)
yoff = 0. > (y1-mn[1])/(n_elements(gridlen) gt 1?gridlen[1]:gridlen) <
(d[1]-1.)
xbin = floor(xoff) & ybin=floor(yoff)
bin = xbin + d[0]*ybin   ; 1D index

; search 4 bins for closets match - check which quadrant
xoff = 1 - 2*((xoff-xbin) lt 0.5)
yoff = 1 - 2*((yoff-ybin) lt 0.5)

rad2 = distance^2

; loop through all neighbouring cells in correct order
for xi=0,1 do begin
  for yi=0,1 do begin
    b = 0l > (bin + xi*xoff + yi*yoff*d[0]) < (d[0]*d[1]-1)

    ; dual histogram method, loop by count in search bins (see JD's
code)
    h2 = histogram(h[b], omin=om, reverse_indices=ri2)

    ; loop through repeat counts
    for k=long(om eq 0), n_elements(h2)-1 do if h2[k] gt 0 then begin
      these_bins = ri2[ri2[k]:ri2[k+1]-1]

      if k+om eq 1 then begin ; single point
        these_points = ri[ri[b[these_bins]]]
      endif else begin
        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

      ; figure out which ones are really within
      within = where( (x2[these_points]-x1[these_bins])^2 +
(y2[these_points] - $
        y1[these_bins])^2 le rad2, nwithin)

      if nwithin gt 0 then begin
        ; have there been any pairs yet?
        if n_elements(plausible) eq 0 then begin
          plausible = [[these_bins[within]],[these_points[within]]]
```

```
        endif else begin
          ; concatenation is inefficient, but we do it at most 4 x N1
times
          plausible = [plausible,[[these_bins[within]],
[these_points[within]]]]
        endelse
      endif

    endif
  endfor
endfor

if n_elements(plausible) eq 0 then begin
  nwithin=replicate(0l,n1)
  return, replicate(-1,n1+1)
endif else begin
  ; use histogram to generate a reverse_indices array that contains
  ; the relevant entries, and then map into the appropriate elements
  ; in 2
  nwithin = histogram(plausible[*,0], min=0, max=n1-1,
reverse_indices=npri)
  npri[n1+1] = plausible[npri[n1+1:*],1]
  return, npri
endelse

end
```