Subject: Re: Matching 2 lists
Posted by Jeremy Bailin on Thu, 26 Aug 2010 14:10:56 GMT
View Forum Message <> Reply to Message

By request, I've created a version of this called MATCHALL_ND that
works for an arbitrary number of dimensions. It should be very fast.
It uses the VALUE_LOCATE mapping trick to deal with sparse histograms
(like in WITHINSPHRAD_VEC3D), so it will be memory efficient even when
the points occupy a miniscule fraction of the full N-dimensional space
they span - which is increasingly likely as you go to higher
dimensions.

Yes, I will get off my ass and put up a new version of JBIU with all
of these in it one of these days...

-Jeremy.

```
; normally: translates the array of indices aind into a
; single 1D index (the inverse operation of array_indices).
; If usemap is set, then it maps that 1D index using value_locate
; (i.e. returns the entry within the map corresponding to the
; 1D index) and returns -1 if it doesn't exist in the map.
function matchallmap, ngrid, aind, usemap=map
  ; inverse of array_indices:
  index = total( aind *
rebin( 1#[1,product(ngrid[0:n_elements(ngrid)-2], $
    /cumul, /int)], size(aind,/dimen), /sample), 2, /int)
  if n_elements(map) eq 0 then return, index
  result = value_locate(map, index)
  missing = where(map[result] ne index, nmissing)
  if nmissing gt 0 then result[missing]=-1
  return, result
end


;+
; NAME:
;    MATCHALL_ND
;
; PURPOSE:
;    Determines which of a set of coordinates of arbitratry dimension
are a
;    given distance from each of a vector of points. Based on JD's
MATCH_2D
;    and my WITHINSPHRAD_VEC3D.
```

```
;
; CATEGORY:
;    Astro
;
; CALLING SEQUENCE:
;    Result = MATCHALL_ND(P1, P2, Distance, Nwithin)
;
; INPUTS:
;    P1:     N1xD array of D-dimensional coordinates.
;
;    P2:     N2xD array of D-dimensional coordinates.
;
;    Distance:  Maximum D-dimensional distance.
;
; OUTPUTS:
;    The function returns the list of indices of P2 that lie within
;    Distance of each point in P1. The format of the returned array is
;    similar to the REVERSE_INDICES array from HISTOGRAM: the indices
;    into P2 that are close enough to element i of P1 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 entry
in P1.
;
; EXAMPLE:
;    Shows in two projections the points within a Gaussian 3D cloud
that are
;    within a distance of 0.1 of 100 random points within the cloud.
;
;    a = randomn(seed, 100, 3)
;    b = randomn(seed, 100000, 3)
;    result = matchall_nd(a, b, 0.1)
;    !p.multi=[0,2,1]
;    plot, psym=3, /iso, b[*,0], b[*,1], xtitle='x', ytitle='y'
;    oplot, psym=1, color=fsc_color('blue'), a[*,0], a[*,1]
;    oplot, psym=3, color=fsc_color('red'), b[result[101:*],0],
b[result[101:*],1]
;    plot, psym=3, /iso, b[*,0], b[*,2], xtitle='x', ytitle='z'
;    oplot, psym=1, color=fsc_color('blue'), a[*,0], a[*,2]
;    oplot, psym=3, color=fsc_color('red'), b[result[101:*],0],
b[result[101:*],2]
;
; 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)
;    25 May 2010    Bug fix to allow X2 and Y2 to have any dimension.
;    23 August 2010 Generalized to an arbitrary number of dimensions
and
;                   to use the manifold-mapping technique from
WITHINSPHRAD_VEC3D
;                   if the space is sparse enough as MATCHALL_ND.
;-
function matchall_nd, p1, p2, distance, nwithin

manifoldfrac=0.25   ; use the remapping trick if less than this
fraction
                    ; of the cells would be occupied

if (size(distance))[0] ne 0 then message, 'Distance must be a scalar.'
p1size = size(p1,/dimen)
p2size = size(p2,/dimen)
if n_elements(p1size) ne 2 then $
  message, 'P1 must be an N1xD dimensional array.'
if n_elements(p2size) ne 2 then $
  message, 'P2 must be an N2xD dimensional array.'
if p1size[1] ne p2size[1] then $
  message, 'P1 and P2 must have the same number of dimensions.'

ndimen = p1size[1]
n1 = p1size[0]
n2 = p2size[0]

gridlen = 2.*distance
mx=max(p2,dimen=1, min=mn)
mn-=1.5*gridlen
mx+=1.5*gridlen

ngrid = ceil( (mx-mn)/gridlen )

; which bins do points 1 and 2 fall in?
```

```
off1 = (p1 - rebin(1#mn,n1,ndimen,/sample))/gridlen
off2 = (p2 - rebin(1#mn,n2,ndimen,/sample))/gridlen
bin1 = floor(off1)
bin2 = floor(off2)

; calculate 1D indices
indices1 = matchallmap(ngrid, bin1)
indices2 = matchallmap(ngrid, bin2)
; calculate 1D indices that are used
allindices = [indices1,indices2]
allindices = allindices[uniq(allindices,sort(allindices))]
; how densely packed are they, ie. what fraction of bins are used?
fracbinused = n_elements(allindices) / product(ngrid)
; map if only a small fraction are used
if fracbinused lt manifoldfrac then map=temporary(allindices)

; map the indices of P2 if necessary (just do it here rather
; than in matchallmap because we've already calculated the
; 1D indices, and we know by construction that every element
; in indices2 must appear in the map)
if n_elements(map) ne 0 then indices2=value_locate(map,indices2)

; histogram points 2
; note the extra 0 out front - used so that when we look for bin -1
; we know there are 0 entries there.
h = [0, histogram(indices2, omin=hmin, reverse_indices=ri)]

; calculate which half of each bin the points are in
off1 = 1 - 2*((off1-bin1) lt 0.5)

rad2 = distance^2

; loop through all neighbouring cells
ncell = 2L^ndimen
powersof2 = 2L^indgen(ndimen)
for ci=0L,ncell-1 do begin
  ; array of cell direction we're working on in each dimension
  di = (ci and powersof2)/powersof2

  b = matchallmap(ngrid, bin1+rebin(1#di,n1,ndimen,/sample)*off1,
usemap=map)

  ; dual histogram method, loop by count in search bins (see JD's
code)
  h2 = histogram(h[(b-hmin+1) > 0], 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]-hmin]]
    endif else begin
      targ=[h2[k],k+om]
      these_points = ri[ri[rebin(b[these_bins]-hmin,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( total( (p2[these_points,*]-p1[these_bins,*])^2, 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 ncell x
N1 times
        plausible = [plausible,[[these_bins[within]],
[these_points[within]]]]
      endelse
    endif

  endif
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
```