Subject: Re: Matching 2 lists
Posted by David Baker on Sat, 21 Aug 2010 15:16:36 GMT
View Forum Message <> Reply to Message

On Aug 21, 4:11 pm, David Baker <de...@le.ac.uk> wrote:
> Hi there,
>   I'm wondering if someone can help me. I'm trying to match
> two lists of stars together. Where I differ from the standard 1-1
> match that match_2d.pro does so well is that I would like to be able
> to compute a 1-many match. I.e find any star in list B that is a
> possible match to a single star in list A not just the closest.
>
> Many thanks for any help that someone can provide
>
> David

-oh and just in case, the coordinates are just x and y pixel
coordinates not ra and dec. It's just simple euclidean distance that
I'm using to set the search radius

Subject: Re: Matching 2 lists
Posted by Gray on Sat, 21 Aug 2010 15:28:23 GMT
View Forum Message <> Reply to Message

On Aug 21, 11:16 am, David Baker <de...@le.ac.uk> wrote:
> On Aug 21, 4:11 pm, David Baker <de...@le.ac.uk> wrote:
>
>> Hi there,
>>   I'm wondering if someone can help me. I'm trying to match
>> two lists of stars together. Where I differ from the standard 1-1
>> match that match_2d.pro does so well is that I would like to be able
>> to compute a 1-many match. I.e find any star in list B that is a
>> possible match to a single star in list A not just the closest.
>
>> Many thanks for any help that someone can provide
>
>> David
>
> -oh and just in case, the coordinates are just x and y pixel
> coordinates not ra and dec. It's just simple euclidean distance that
> I'm using to set the search radius

If you're using IDL 8.0, you could modify match_2d to return a list of
length n_a, where each element is an array of indices into b.

Subject: Re: Matching 2 lists
Posted by David Baker on Sat, 21 Aug 2010 15:35:29 GMT
View Forum Message <> Reply to Message

On Aug 21, 4:28 pm, Gray <grayliketheco...@gmail.com> wrote:
> On Aug 21, 11:16 am, David Baker <de...@le.ac.uk> wrote:
>
>
>
>> On Aug 21, 4:11 pm, David Baker <de...@le.ac.uk> wrote:
>
>>> Hi there,
>>>   I'm wondering if someone can help me. I'm trying to match
>>> two lists of stars together. Where I differ from the standard 1-1
>>> match that match_2d.pro does so well is that I would like to be able
>>> to compute a 1-many match. I.e find any star in list B that is a
>>> possible match to a single star in list A not just the closest.
>
>>> Many thanks for any help that someone can provide
>
>>> David
>
>> -oh and just in case, the coordinates are just x and y pixel
>> coordinates not ra and dec. It's just simple euclidean distance that
>> I'm using to set the search radius
>
> If you're using IDL 8.0, you could modify match_2d to return a list of
> length n_a, where each element is an array of indices into b.

Thanks for the advice, whilst I maybe comfortable programming basic
stuff I can't even begin to follow the match_2d code so wouldn't know
where exactly to code in what you suggest. But that is exactly what
I'm after.

-Cheers,
David

Subject: Re: Matching 2 lists
Posted by David Baker on Sat, 21 Aug 2010 16:37:34 GMT
View Forum Message <> Reply to Message

On Aug 21, 4:35 pm, David Baker <de...@le.ac.uk> wrote:
> On Aug 21, 4:28 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>> On Aug 21, 11:16 am, David Baker <de...@le.ac.uk> wrote:

>
>>> On Aug 21, 4:11 pm, David Baker <de...@le.ac.uk> wrote:
>
>>>> Hi there,
>>>> I'm wondering if someone can help me. I'm trying to match
>>>> two lists of stars together. Where I differ from the standard 1-1
>>>> match that match_2d.pro does so well is that I would like to be able
>>>> to compute a 1-many match. I.e find any star in list B that is a
>>>> possible match to a single star in list A not just the closest.
>
>>>> Many thanks for any help that someone can provide
>
>>>> David
>
>>> -oh and just in case, the coordinates are just x and y pixel
>>> coordinates not ra and dec. It's just simple euclidean distance that
>>> I'm using to set the search radius
>
>> If you're using IDL 8.0, you could modify match_2d to return a list of
>> length n_a, where each element is an array of indices into b.
>
> Thanks for the advice, whilst I maybe comfortable programming basic
> stuff I can't even begin to follow the match_2d code so wouldn't know
> where exactly to code in what you suggest. But that is exactly what
> I'm after.
>
> -Cheers,
> David

You mentioned using IDL 8.0. My university is currently still on
7.1.1, is there anything unique about IDL 8.0 to the solution you
propose?

Subject: Re: Matching 2 lists
Posted by penteado on Sat, 21 Aug 2010 16:52:19 GMT
View Forum Message <> Reply to Message

On Aug 21, 1:37 pm, David Baker <de...@le.ac.uk> wrote:
>>> If you're using IDL 8.0, you could modify match_2d to return a list of
>>> length n_a, where each element is an array of indices into b.
>
>> Thanks for the advice, whilst I maybe comfortable programming basic
>> stuff I can't even begin to follow the match_2d code so wouldn't know
>> where exactly to code in what you suggest. But that is exactly what
>> I'm after.
>
>> -Cheers,

>> David
>
> You mentioned using IDL 8.0. My university is currently still on
> 7.1.1, is there anything unique about IDL 8.0 to the solution you
> propose?

Lists and empty arrays. In that case, because each element of A can have a different number of matches, so each element of the list is an array of a different size (possibly empty, if there are no matches).

The most direct way to do something similar in IDL 7 is a pointer array, where each element points to the array of indices that match the corresponding element of A. Which is more awkward to use due to the need to dereference the pointer, and the need to test (with null pointers, for instance) for the no-match case.

Subject: Re: Matching 2 lists
Posted by David Baker on Sat, 21 Aug 2010 17:03:39 GMT

View Forum Message <> Reply to Message

On Aug 21, 5:52 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Aug 21, 1:37 pm, David Baker <de...@le.ac.uk> wrote:
>
>>>> If you're using IDL 8.0, you could modify match_2d to return a list of
>>>> length n_a, where each element is an array of indices into b.
>
>>> Thanks for the advice, whilst I maybe comfortable programming basic
>>> stuff I can't even begin to follow the match_2d code so wouldn't know
>>> where exactly to code in what you suggest. But that is exactly what
>>> I'm after.
>
>>> -Cheers,
>>> David
>
>> You mentioned using IDL 8.0. My university is currently still on
>> 7.1.1, is there anything unique about IDL 8.0 to the solution you
>> propose?
>
> Lists and empty arrays. In that case, because each element of A can
> have a different number of matches, so each element of the list is an
> array of a different size (possibly empty, if there are no matches).
>
> The most direct way to do something similar in IDL 7 is a pointer
> array, where each element points to the array of indices that match
> the corresponding element of A. Which is more awkward to use due to
> the need to dereference the pointer, and the need to test (with null
> pointers, for instance) for the no-match case.

OK, but it should still be possible? You're suggesting something like
a two element structure, one element contains the index from A the
other element contains a pointer listing the indicies of any object
from list B that is within the search radius? The structure would just
need to grow via a=[b,c] for each object in A that has matches to
those in B (though this would obviously cause a speed penalty I think
it might outweigh that caused by looping over my lists). The tricky
part I guess (at least for me) is finding out where match_2d prints
out the indicies from B that match to each individual index in A.

-David

Subject: Re: Matching 2 lists
Posted by Jeremy Bailin on Sat, 21 Aug 2010 19:48:52 GMT

View Forum Message <> Reply to Message

On Aug 21, 11:11 am, David Baker <de...@le.ac.uk> wrote:
> Hi there,
>   I'm wondering if someone can help me. I'm trying to match
> two lists of stars together. Where I differ from the standard 1-1
> match that match_2d.pro does so well is that I would like to be able
> to compute a 1-many match. I.e find any star in list B that is a
> possible match to a single star in list A not just the closest.
>
> Many thanks for any help that someone can provide
>
> David

This is going to be in the next JBIU release, whenever I have half a
second to run idldoc on it and tar it all up... it's based heavily on
match_2d, obviously!

-Jeremy.

```
;+
; 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)
;    25 May 2010    Bug fix to allow X2 and Y2 to have any dimension.
;-
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([reform(x2,1,n_elements(x2)),reform(y2,1,n_elements( 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
```

Subject: Re: Matching 2 lists
Posted by David Baker on Sun, 22 Aug 2010 07:06:05 GMT

View Forum Message <> Reply to Message

On Aug 21, 8:48 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Aug 21, 11:11 am, David Baker <de...@le.ac.uk> wrote:
>
>> Hi there,
>>   I'm wondering if someone can help me. I'm trying to match
>> two lists of stars together. Where I differ from the standard 1-1
>> match that match_2d.pro does so well is that I would like to be able
>> to compute a 1-many match. I.e find any star in list B that is a
>> possible match to a single star in list A not just the closest.
>
>> Many thanks for any help that someone can provide
>
>> David
>
> This is going to be in the next JBIU release, whenever I have half a
> second to run idldoc on it and tar it all up... it's based heavily on
> match_2d, obviously!
>
> -Jeremy.
>

```
> ;+
> ; 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)
> ;    25 May 2010    Bug fix to allow X2 and Y2 to have any dimension.
> ;-
> 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([reform(x2,1,n_elements(x2)),reform(y2,1,n_elements( 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
```

Jeremy thats fantastic thank you so much, already saving me many hours
of data processing my supervisor will certainly be happy.

-David

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
```