Subject: The intersection of 2 arrays
Posted by Phil Williams on Mon, 03 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

I know that I can find the union of two arrays by doing something like

temp = [array1, array2]
union = temp(uniq(temp, sort(temp)))

But how would I go about finding the intersection of the two arrays?
i.e. Is there a nice vector way of doing it rather than brute force?
(aka: Which IDL function did I miss this time?)

Thanks in advance,
Phil
--
/** ********/
Phil Williams, Ph.D.
Research Instructor
Children's Hospital Medical Center "One man gathers what
Imaging Research Center another man spills..."
3333 Burnet Ave. -The Grateful Dead
Cincinnati, OH 45229
email: williams@irc.chmcc.org
URL: http://scuttle.chmcc.org/~williams/
/** ********/

Subject: Re: The intersection of 2 arrays
Posted by wonko on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

williams@irc.chmcc.org (Phil Williams) wrote:

> I know that I can find the union of two arrays by doing something like
>
> temp = [array1, array2]
> union = temp(uniq(temp, sort(temp)))
>
> But how would I go about finding the intersection of the two arrays?
> i.e. Is there a nice vector way of doing it rather than brute force?
> (aka: Which IDL function did I miss this time?)

How big are your elements?

My idea would be something like this:

mask = bytarr(max([array1, array2]) + 1)

mask(array1) = 1b
mask(array2) = mask(array2) + 1b
intersection = where(mask eq 2b, count)

Works for positive values only. And the intersection of [1L, 2L] and [1L, 100000000L] might be a problem...

Alex
--
Alex Schuster Wonko@weird.cologne.de
alex@pet.mpin-koeln.mpg.de

Subject: Re: The intersection of 2 arrays
Posted by J.D. Smith on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David Foster wrote:
> PS>
>
> I've written a FIND_ELEMENTS.PRO that returns subscripts of
> elements in one array that are found in another array, like:
>
> ind = Find_Elements(Array, Tofind [, Adjust_array])
>
> This searches Array for values in Tofind, returning
> the subscripts of Array for those values found. The optional
> argument Adjust_array contains the same number of elements as
> Tofind, and is a way of allowing you to adjust the elements
> of Array (the values of Adjust_array and Tofind correspond).
> We use this for finding and adjusting Regions-of-Interest in
> images which have a limited number of discrete values. It
> uses a loop to search Array for each value of Tofind, so
> it's not real fast for large search arrays. [If anyone has
> an array-oriented method I'd love to see it!!]

Check out the NASA library routine match(), which is array based. It uses a flag array and an index array, so the memory overhead is roughly 3 times the sum of the two arrays, but it's pretty fast. It's attached. Note that it takes vectors, so you've go to flatten your array upon input (with reform).

I'd be interested to see the time comparisons for, say 2 128^2-element vectors of random integers on the interval [0-2000], or some equivalent big array test, because I've got some non-optimized routines that I've been trying to convince myself to rewrite.

JD

```
pro match, a, b, suba, subb, COUNT = count
;+
; NAME:
;       MATCH
; PURPOSE:
;       Routine to match values in two vectors.
;
; CALLING SEQUENCE:
;       match, a, b, suba, subb, [ COUNT = ]
;
; INPUTS:
;       a,b - two vectors to match elements
;
; OUTPUTS:
;       suba - subscripts of elements in vector a with a match
;               in vector b
;       subb - subscripts of the positions of the elements in
;               vector b with matchs in vector a.
;
;       suba and subb are ordered such that a(suba) equals b(subb)
;
; OPTIONAL KEYWORD OUTPUT:
;       COUNT - set to the number of matches, integer scalar
;
; SIDE EFFECTS:
;       !ERR is set to the number of matches, can be used instead of COUNT
;
; RESTRICTIONS:
;       a and b should not have duplicate values within them.
;       You can use rem_dup function to remove duplicate values
;       in a vector
;
; EXAMPLE:
;       If a = [3,5,7,9,11]   & b = [5,6,7,8,9,10]
;       then
;               IDL> match, a, b, suba, subb, COUNT = count
;
;       will give suba = [1,2,3], subb = [0,2,4],  COUNT = 3
;       and       suba(a) = subb(b) = [5,7,9]
;
; HISTORY:
;       D. Lindler  Mar. 1986.
;       Fixed "indgen" call for very large arrays   W. Landsman  Sep 1991
;       Added COUNT keyword    W. Landsman   Sep. 1992
;       Fixed case where single element array supplied   W. Landsman Aug 95
;-
```

```
;---------------------------------------------------------- ----------
On_error,2

if N_params() LT 3 then begin
   print,'Syntax - match, a, b, suba, subb, [ COUNT = ]'
   print,'    a,b -- input vectors for which to match elements'
   print,'    suba,subb -- output subscript vectors of matched elements'
   return
endif

na = N_elements(a)          ;number of elements in a
nb = N_elements(b)          ;number of elements in b

; Check for a single element array

if (na EQ 1) or (nb EQ 1) then begin
      if (nb GT 1) then begin
            subb = where(b EQ a(0), nw)
            if (nw GT 0) then suba = replicate(0,nw) else suba = [-1]
      endif else begin
            suba = where(a EQ b(0), nw)
            if (nw GT 0) then subb = replicate(0,nw) else subb = [-1]
      endelse
      count = nw
      return
endif

c = [ a, b ]                ;combined list of a and b
ind = [ lindgen(na), lindgen(nb) ]      ;combined list of indices
vec = [ bytarr(na), replicate(1b,nb) ]  ;flag of which vector in  combined
                            ;list   0 - a   1 - b

; sort combined list

sub = sort(c)
c = c(sub)
ind = ind(sub)
vec = vec(sub)

; find duplicates in sorted combined list

n = na + nb                 ;total elements in c
firstdup = where( (c EQ shift(c,-1)) and (vec NE shift(vec,-1)), Count )

if Count EQ 0 then begin            ;any found?
      suba = lonarr(1)-1
      subb = lonarr(1)-1
      return
```

```
end

dup = lonarr( Count*2 )                  ;both duplicate values
even = lindgen( N_elements(firstdup))*2     ;Changed to LINDGEN 6-Sep-1991
dup(even) = firstdup
dup(even+1) = firstdup+1
ind = ind(dup)                        ;indices of duplicates
vec = vec(dup)                         ;vector id of duplicates
suba = ind( where( vec EQ 0 ) )        ;a subscripts
subb = ind( where( vec EQ 1 ) )        ;b subscripts

return
end
```

Subject: Re: The intersection of 2 arrays
Posted by Marty Ryba on Tue, 04 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

This is a multi-part message in MIME format.

--------------2E7B149D1BEF
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Phil Williams wrote:
> But how would I go about finding the intersection of the two arrays?
> i.e. Is there a nice vector way of doing it rather than brute force?
> (aka: Which IDL function did I miss this time?)

I don't know if this is *exactly* what you are looking for, but the algorithm should be useful. This is SYNCHRONIZE, which will find the intersection of two arrays of structures based on matching values of a specified tag name. I use it often in our data analysis. Many thanks go to David Stern of RSI for the algorithm.

--
Dr. Marty Ryba | Of course nothing I say here is official
MIT Lincoln Laboratory | policy, and Laboratory affililaton is
ryba@ll.mit.edu | for identification purposes only,
| blah, blah, blah, ...

--------------2E7B149D1BEF
Content-Type: text/plain; charset=us-ascii; name="synchronize.pro"
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="synchronize.pro"

```
;+
```

```
; Name:
; SYNCHRONIZE
; Purpose:
; Match up two arrays of structures by a tag name
; Usage:
; synchonize,a,b[,ause=ause][,buse=buse][,keywords=values]
; Inputs:
; A & B are arrays of structures to be sychronized.  They are
; returned containing only those members that match within the
; specified tolerance for the tag fields being matched.
; Optional Keyword Inputs:
; tags - String array containing the tag names to match by.  Defaults
;  to ['dwell','dwell'].  Case insensitive.
; tolerance - Maximum difference to declare a match.  All comparisions
;  are double precision.  Defaults to 0.0 (exact match).
; Optional Outputs:
; ause - Set of array indicies used to convert input A to output A.
;  Useful if you have 2 already synchonized arrays and need to
;  synch a third.
; buse - Same for B
; Restrictions:
;     The structures should have only one member with the tag name given.
; If there are multiple structure members with the same tag, SYNCHRONIZE
; will use the first.
;     Cannot use tags from nested structures.
; Modification History:
; M.F. Ryba, MIT/LL, June 93, Created from algorithm written by David
;  Stern of RSI.
; M.F. Ryba, Jan 95, added TEMPORARY and check for whether the
;  structure is shortened.
;-
PRO Synchronize, a, b, ause=ause, buse=buse, tags=tags, tolerance=tolerance, $
                help=help

IF keyword_set(help) THEN BEGIN
   doc_library, 'synchronize'
   return
ENDIF

IF n_elements(tags) EQ 0 THEN tags = ['dwell', 'dwell']
IF n_elements(tolerance) EQ 0 THEN tolerance = 0.0d
tags = strupcase(tags)
atags = tag_names(a)
btags = tag_names(b)

ai = where(atags EQ tags(0), cnt)
IF cnt EQ 0 THEN BEGIN
   string = 'Tag '+tags(0)+' not found in structure A'
```

```
   message, string
ENDIF
IF cnt GT 1 THEN BEGIN
   string = 'Structure A has more than 1 tag named '+tags(0)+ $
     '; will use the first'
   message, string, /informational
ENDIF

bi = where(btags EQ tags(1), cnt)
IF cnt EQ 0 THEN BEGIN
   string = 'Tag '+tags(1)+' not found in structure B'
   message, string
ENDIF
IF cnt GT 1 THEN BEGIN
   string = 'Structure B has more than 1 tag named '+tags(1)+ $
     '; will use the first'
   message, string, /informational
ENDIF

ai = ai(0) & bi = bi(0)
tmp = [double(a.(ai)), double(b.(bi))]
sortab = sort(tmp)
tmp = tmp(sortab)

match = where((tmp(1:*) - tmp) LE tolerance, cnt)

IF cnt GT 0 THEN BEGIN
   aeq = sortab(match)
   beq = sortab(match+1)
   tmp = aeq < beq
   beq = (beq > aeq) - n_elements(a)
   aeq = tmp
   IF n_elements(aeq) NE n_elements(a) THEN a = (temporary(a))(aeq)
   IF n_elements(beq) NE n_elements(b) THEN b = (temporary(b))(beq)
   ause = aeq
   buse = beq
ENDIF ELSE BEGIN
   print, 'No matches found between A.'+tags(0)+' and B.'+tags(1)
   ause = -1
   buse = -1
ENDELSE

return
END

----------2E7B149D1BEF--
```

Subject: Re: The intersection of 2 arrays
Posted by James Tappin on Thu, 06 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Phil Williams wrote:
>
> I know that I can find the union of two arrays by doing something like
>
> temp = [array1, array2]
> union = temp(uniq(temp, sort(temp)))
>
> But how would I go about finding the intersection of the two arrays?
> i.e. Is there a nice vector way of doing it rather than brute force?
> (aka: Which IDL function did I miss this time?)

Whether it's feasible depends on the size of the arrays. As far as I know
there is no really nice way to do it. But for smallish 1-D arrays I think
the following will work

```
temp = reform(array2,1,n_elements(array2))
imatch = array1(*,intarr(n_elements(array2)) eq $
 temp(intarr(n_elements(array1),*)
sum=total(imatch,1)
locs=where(sum ne 0)
inter = array1(locs)
```

Several of those lines can probably be condensed together, but even so I'd
rather not try it for two 13000 element arrays.

 +-----------------------+---------------------------------- --+---------+
James Tappin,	School of Physics & Space Research	O__
sjt@star.sr.bham.ac.uk	University of Birmingham	-- \/`
Ph: 0121-414-6462. Fax: 0121-414-3722	
 +-- --+---------+