Subject: Re: matching lists
Posted by John-David T. Smith on Fri, 10 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Fardal wrote:
>
> Hi,
>
> I have been looking at properties of particles in a simulation, and
> sometimes need to match up the particles in two different subsets. I
> typically have (quantity A, index #) for one set of particles, and
> (quantity B, index #) for another set, and want to compare quantities
> A and B for the particles that are in both sets.
>
> As of late last night I could not think of a good way to do this;
> WHERE inside a for-loop would be very slow. Maybe I'm missing
> something easy, but in any case here's a solution inspired by the
> recently submitted SETINTERSECTION function. Hope somebody finds
> it useful.
>

The standard where_array, as posted a few years back, and modified slightly for the case of the null intersection, is attached. It will work with floating point and other data types also. It works by inflating the vectors input to 2-d and testing for equality in one go. It will also handle the case of repeated entries.

For sparse integer data sets, such as the SSN example you give in the routine documentation, it will be vastly superior. Let's take your SSN for an example. If collected from random geographic locations and age brackets, the range of integers might span a billion. Suppose you had 1000 entries in your survey. The histograms will each return a vector of length 1-billion (almost all zeroes), whereas two 1-million entry arrays would be made using where_array... the difference between uncomfortable and easy.

In the other extreme, in which the data are well grouped (not sparse) and you have a lot of them (e.g. a set of particles labelled from 1-1e6), you will be better off with histogram... and now that I think about it, why can't we formulate it like this:

```
n_hist=maxab-minab
n_arr= n_elements(a) * n_elements(b)
if n_hist lt n_arr then
 ; use set_intersection method
else
 ; use array inflation method
endif
```

to get the best of both worlds. Hmm... maybe if I'm bored someday. By the way, you could fix your method to deal with repeated entries by dealing more carefully with the reverse indices, to collect every index in reva[reva[r[i]]:reva[r[i]+1]-1] for all i. I'll leave it as a challenge to do that without a loop over i ;)

JD

--

J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

```
;+
; NAME:
;       WHERE_ARRAY
;
; PURPOSE:
;  Return the indices of those elements in vector B which
;       exist in vector A.  Basically a WHERE(B IN A)
;       where B and A are 1 dimensional arrays.
;
; CATEGORY:
;       Array
;
; CALLING SEQUENCE:
;       result = WHERE_ARRAY(A,B)
;
; INPUTS:
;       A       vector that might contains elements of vector B
;       B       vector that we would like to know which of its
;               elements exist in A
;
; OPTIONAL INPUTS:
;
; KEYWORD PARAMETERS:
;       iA_in_B         return instead the indices of A that are in
;                     (exist) in B
;
; OUTPUTS:
;       Index into B of elements found in vector A.  If no
;       matches are found -1 is returned.  If the function is called
;       with incorrect arguments, a warning is displayed, and -2 is
;       returned (see SIDE EFFECTS for more info)
;
; OPTIONAL OUTPUTS:
;
; COMMON BLOCKS:
```

```
;             None
;
; SIDE EFFECTS:
;       If the function is called incorrectly, a message is displayed
;       to the screen, and the !ERR_STRING is set to the warning
;       message.  No error code is set, because the program returns
;       -2 already
;
; RESTRICTIONS:
;       This should be used with only Vectors.  Matrices other then
;       vectors will result in -2 being returned.  Also, A and B must
;       be defined, and must not be strings!
;
; PROCEDURE:
;
; EXAMPLE:
;       IDL> A=[2,1,3,5,3,8,2,5]
;       IDL> B=[3,4,2,8,7,8]
;       IDL> result = where_array(a,b)
;       IDL> print,result
;                  0          0          2          2          3          5
; SEE ALSO:
;       where
;
; MODIFICATION HISTORY:
;       Written by:     Dan Carr at RSI (command line version) 2/6/94
;                       Stephen Strebel         3/6/94
;                             made into a function, but really DAN did all
;                             the thinking on this one!
;                       Stephen Strebel         6/6/94
;                             Changed method, because died with Strings (etc)
;                             Used ideas from Dave Landers.  Fast TOO!
;                       Strebel 30/7/94
;                             fixed checking structure check
;                       Smith, JD 9/1/98
;                        Minor Tweak to case of no overlapping members
;-
FUNCTION where_array,A,B,IA_IN_B=iA_in_B

; Check for: correct number of parameters
;                       that A and B have each only 1 dimension
;                       that A and B are defined
  if (n_params() ne 2 or (size(A))(0) ne 1 or (size(B))(0) ne 1 $
      or n_elements(A) eq 0 or n_elements(B) eq 0) then begin
    message,'Inproper parameters',/Continue
    message,'Usage: result = where_array(A,B,[IA_IN_B=ia_in_b]',/Continue
    return,-2
  endif
```

```
;parameters exist, let's make sure they are not structures
   if ((size(A))((size(A))(0)+1) eq 8 or $
       (size(B))((size(B))(0)+1) eq 8) then begin
      message,'Inproper parametrs',/Continue
      message,'Parameters cannot be of type Structure',/Continue
      return,-2
   endif

; build two matrices to compare
   Na = n_elements(a)
   Nb = n_elements(b)
   l = lindgen(Na,Nb)
   AA = A(l mod Na)
   BB = B(l / Na)

;compare the two matrices we just created
   I = where(AA eq BB,cnt)
   if cnt eq 0 then return,-1

; normally (without keyword), return index of B that exist in A
   if keyword_set(iA_in_B) then return, I mod Na
   return,I/Na

END
```

File Attachments

1) where_array.pro, downloaded 117 times