Subject: Re: Efficient comparison of arrays
Posted by davidf on Sun, 10 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

David R. Klassen writes in response to an Andy Loughe question:

>> Given vectors of the type...
>>
>> a = [1,2,3,4,5]
>> b = [3,4,5,6,7]
>>
>> What is the most efficient way to determine which values that occur in
>> a also occur in b (i.e., the values [3,4,5] occur in both a and b).
> How about:
> x=where(a eq b)
> This will give you the index numbers in a of those values that are also in b.
> So, to vector of the actual values would be a(x).

Ugh, I don't think so. Maybe it *should* work that way, but it
doesn't. At least not on my computer. :-)

I don't know if this is the most efficient way (it probably isn't),
but this is my off-the-cuff way of solving this problem.

```
FUNCTION A_in_B, a, b
num = N_Elements(a)
vector = FltArr(num)
FOR j=0,num-1 DO BEGIN
  index = Where(a(j) EQ b, count)
  IF count GT 0 THEN vector(j) = 1 ELSE vector(j)=0
ENDFOR
solution = a(Where(vector EQ 1))
RETURN, solution
END
```

When I run the example case above, I get a vector with the values
[3,4,5].

It might be more efficient to sort the arrays and then use some
kind of bubble-sort routine to find the first instance of a in b.
The WHERE function is going to find *all* instances, which is
probably the most inefficient part of this program.

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
Customizable IDL Programming Courses
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com

Subject: Re: Efficient comparison of arrays
Posted by David R. Klassen on Sun, 10 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Andy Loughe wrote:
> Given vectors of the type...
>
> a = [1,2,3,4,5]
> b = [3,4,5,6,7]
>
> What is the most efficient way to determine which values that occur in
> a also occur in b (i.e., the values [3,4,5] occur in both a and b).
How about:
 x=where(a eq b)
This will give you the index numbers in a of those values that are also in b.
So, to vector of the actual values would be a(x).

--
David R. Klassen
 Department of Astronomy
 Center for Radiophysics and Space Research
 304 Space Sciences Building
 Cornell University
 Ithaca NY 14853
 phone: 607-255-6910
http://faraday.uwyo.edu/grads/dklassen/
drk14@cornell.edu

Subject: Re: Efficient comparison of arrays
Posted by wonko on Mon, 11 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

afl@cdc.noaa.gov (Andy Loughe) wrote:

> Given vectors of the type...
>
> a = [1,2,3,4,5]
> b = [3,4,5,6,7]
>

> What is the most efficient way to determine which values that occur in
> a also occur in b (i.e., the values [3,4,5] occur in both a and b).
>
> Presumably this needs to be done without loops (to be efficient), but
> an obvious solution escapes me.

What type of data would this be? If you are dealing with integers, what about this:

```
max_value = max( [ a, b ], min_value=min_value )
mask = indgen( max_value-min_value+1 )
mask(a-min_value) = 1
mask(b-min_value) = mask(b-min_value) + 1
values = where( mask eq 2, count ) + min_value
```

Voila, no loops, I would give this a try. Bad luck if you have floating point data.

Someone already came up with this question five months ago, but I don't remember the responses. Maybe you can find the thread via dejanews.

Alex

--

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

Subject: Re: Efficient comparison of arrays
Posted by David P. Steele on Mon, 11 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

This is a multi-part message in MIME format.

```
--------------4A996C43C0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
```

Marty Ryba wrote:
>
> David Fanning wrote:
>>>> Given vectors of the type...
>>>>
>>>> a = [1,2,3,4,5]
>>>> b = [3,4,5,6,7]
>>>>
>>>> What is the most efficient way to determine which values that occur in
>>>> a also occur in b (i.e., the values [3,4,5] occur in both a and b).
>> I don't know if this is the most efficient way (it probably isn't),

```
>>  but this is my off-the-cuff way of solving this problem.
>>
>>    FUNCTION A_in_B, a, b
>>    num = N_Elements(a)
>>    vector = FltArr(num)
>>    FOR j=0,num-1 DO BEGIN
>>       index = Where(a(j) EQ b, count)
>>       IF count GT 0 THEN vector(j) = 1 ELSE vector(j)=0
>>    ENDFOR
>>    solution = a(Where(vector EQ 1))
>>    RETURN, solution
>>    END
>>
>>  When I run the example case above, I get a vector with the values
>>  [3,4,5].
>
>  Here's a routine, called SYNCHRONIZE, that I whipped up thanks to some
>  help from RSI.  I had the problem of two data streams, each with a
>  structure with a tag called "dwell" that needed matching up.  This
>  routine works great except possibly for some funny behavior if a given
>  number is repeated in a data stream (nah, that shouldn't happen in real
>  data).  If you don't need the full-blown procedure, the guts of the
>  algorithm should be obvious.
>
>  --
>  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, ...
>
>      ------------------------------------------------------------ ---
>  ;+
>  ; 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
```

Here is another solution that was posted on this newsgroup several years ago. I found it useful enough to keep around. The header credits the creators.

Dave

 -- ------------
David P. Steele   Ph: (306) 966-6447
ISAS, University of Saskatchewan   Fax: (306) 966-6400
116 Science Place   David.Steele@usask.ca
Saskatoon SK S7N 5E2   DANSAS::STEELE


```
--------------4A996C43C0
Content-Type: text/plain; charset=us-ascii; name="Where_ar.pro"
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="Where_ar.pro"

;
;+
; NAME:
```

```
; WHERE_ARRAY
;
; PURPOSE:
; Return the indices where vector B exists in vector A.
; Basically a WHERE(B EQ 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
;-
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)
```

```
Ia = i mod Na
Ib = i / na

; normally (without keyword, return index of B that
; exist in A
if keyword_set(iA_in_B) then index = Ia $
else index = Ib

;make sure a valid value was found
if Ia(0) eq -1 or Ib(0) eq -1 then index = -1

return,index

END


--------------4A996C43C0--
```

Subject: Re: Efficient comparison of arrays
Posted by Marty Ryba on Mon, 11 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

This is a multi-part message in MIME format.

```
--------------66221E172010
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

David Fanning wrote:
>>>  Given vectors of the type...
>>>
>>>  a = [1,2,3,4,5]
>>>  b = [3,4,5,6,7]
>>>
>>>  What is the most efficient way to determine which values that occur in
>>>  a also occur in b (i.e., the values [3,4,5] occur in both a and b).
>  I don't know if this is the most efficient way (it probably isn't),
>  but this is my off-the-cuff way of solving this problem.
>
>     FUNCTION A_in_B, a, b
>     num = N_Elements(a)
>     vector = FltArr(num)
>     FOR j=0,num-1 DO BEGIN
>        index = Where(a(j) EQ b, count)
>        IF count GT 0 THEN vector(j) = 1 ELSE vector(j)=0
>     ENDFOR
>     solution = a(Where(vector EQ 1))
```

>   RETURN, solution
>   END
>
> When I run the example case above, I get a vector with the values
> [3,4,5].

Here's a routine, called SYNCHRONIZE, that I whipped up thanks to some help from RSI. I had the problem of two data streams, each with a structure with a tag called "dwell" that needed matching up. This routine works great except possibly for some funny behavior if a given number is repeated in a data stream (nah, that shouldn't happen in real data). If you don't need the full-blown procedure, the guts of the algorithm should be obvious.

--
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, ...

----------66221E172010
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

----------66221E172010--
```

Subject: Re: Efficient comparison of arrays
Posted by J.D. Smith on Mon, 11 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

> Here is another solution that was posted on this newsgroup several years
> ago. I found it useful enough to keep around. The header credits the
> creators.
>
> Dave
>
> END

Unfortunately, this is even slower than David's routine for all but the tiniest of vectors, and is ridiculously slow for large vectors (depending on the memory in your machine), since it requires two arrays of size N^2 where N is the length of the vector. It is clearly not the best solution to the stated problem (finding common elements, not necessarily their repitition count or location in the vectors). The contain() function I gave in my other post is a much faster alternative. For repitition counts and positions, where_array is more useful.

JD

Subject: Re: Efficient comparison of arrays
Posted by Struan Gray on Tue, 12 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

One way to take advantage of IDL's built in functions when comparing arrays is to use the REVERSE_INDICES keyword in a call to HISTOGRAM. For general problems the methods already posted will probably save you time and memory but if you have a restricted data range or if you are comparing a small set of important values with a large array a cleverly constructed set of histogram bins can save you a lot of looping.

Struan
