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

David Fanning wrote:
>
> 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

Just to keep the Astronomy department here from being one-upped....

In addition to inefficiency, there is another problem with a_in_b.
Try, for instance:

```
IDL> a=[2,4,5,6,5,5,8] & b=[5,8,2,6,5,6,4]
IDL> print,a_in_b(a,b)
      2       4       5       6       5       5       8
```

As you can see, some of the values are replicated, when what I assume you would want is the unique values in this returned vector. You could add a uniq() call, but that would make it even less efficient. The repeated calls to where() make your routine quite slow for large vectors (and unsymmetric with respect to argument interchange given one large and one small vector). It also fails for no common elements. Here is an implementation I just made up:

```
function contain,a,b
    flag=[replicate(0b,n_elements(a)),replicate(1b,n_elements(b) )]
   s=[a,b]
   srt=sort(s)
   s=s(srt) & flag=flag(srt)
   wh=where(s eq shift(s,-1) and flag ne shift(flag, -1),cnt)
   if cnt ne 0 then return, s[wh]
   return,-1
end
```

I ran some time tests on the two implementations. While a_in_b is adequate for small vectors, it is prohibitively slow for large ones. An example averages the result in seconds for two 10000 element random integer vectors on the range [0,20000].

```
Results for a_in_b:
 Average Time:        19.669667

Results for contain:
 Average Time:       0.19233332

Ratio:    102.269
```

And for 100 element vectors in the range [0,200]:

```
Results for a_in_b:
 Average Time:      0.010666664

Results for contain:
 Average Time:     0.0015666644
```

Hope it's useful.

JD
