Subject: Re: Subscripting multidimensional arrays
Posted by Jonathan Greenberg on Sat, 13 Dec 2003 20:19:40 GMT
View Forum Message <> Reply to Message

Hey all, JDs nice little function worked great (thanks! i give you credit in the classifier i'm coding right now -- don't worry, its not commercial)-- i did notice there was an 8-d limitation on arrays, but you could get by the by just making your own subscripts, since, as this little process has shown, the arrays are unidimensional in nature anyway... (e.g. there really isn't a functional difference, as I understand it, between:

1 2
3 4

and

1 2 3 4

You just need to know the order of subscript indexing...

--j

"JD Smith" <jdsmith@as.arizona.edu> wrote in message
news:pan.2003.12.12.16.53.23.571408.24862@as.arizona.edu...
> On Fri, 12 Dec 2003 07:55:16 -0700, Christopher Lee wrote:
>
>> In article <nwfCb.37807$SU2.20541@newssvr29.news.prodigy.com>, "Jonathan
>> Greenberg" <greenberg@ucdavis.edu> wrote:
>>
>>
>>> Hi all -- I was hoping to get some help with converting a vector which
>>> contains the x,y,z position for a value I want to exract from a
>>> multidimensional array -- I understand that using an array to subscript
>>> another array requires knowing the linear subscript position. For
>>> example: a = 0 10 20
>>> 30 40 50
>>> 60 70 80
>>>
>>> 90 100 110
>>> 120 130 140
>>> 150 160 170
>>> I have a vector which is defined as:
>>> locationvector=[2,2,2]
>>> I want to extract the value at that position (e.g. a[2,2,2] = 170), but
>>> I can't do a:
>>> a[locationvector] --> I apparently have to convert the locationvector
>>> to that linear position. How do I do this? Does IDL have a built in

>>> function that will do this conversion, or is there an easy formula for
>>> doing this conversion in ANY dimension? Thanks! --j
>>>
>>>
>> function element, array, loc_vector
>>
>> s=size(array)
>> d=s[1:s[0]] ;dimensions
>> e=lonarr(s[0]) ;product of dimensions e[0]=1L for i=1L, s[0]-1 do
>> e[i]=e[i-1]*d[i-1] ;e is the number of elements each dimension contains
>> return, total(loc_vector*e)
>>
>> end
>>
>> seems to work, there must be a better way though...
>
> PRODUCT works nicely for this:
>
> function linear_indices,array,vec_indices
>   s=size(array,/DIMENSIONS)
>   nd=n_elements(s)
>   if nd eq 1 then return,s[0]
>   return,long(total([1.,product(s[0:nd-1],/CUMULATIVE)]*vec_in dices))
> end
>
> to go the other direction, IDL6 offers ARRAY_INDICES. Or you can always
> just resort to:
>
>   a[vec[0],vec[1],vec[2]]
>
> A take home problem would be to modify this such that NxM input vectors,
> where N is the number of dimensions of "array", will return a vector of
> length M containing all the 1-D indices. Hints: REBIN/REFORM and the
> "dimension" argument to TOTAL.
>
> JD
