Subject: indexing arrays with arrays
Posted by ljg on Thu, 30 Jun 1994 14:31:35 GMT
View Forum Message <> Reply to Message

I have an application where I'm trying to re-bin an array of data sampled
at arbitrary coordinates into an array whose indices indicate sampling
coordinates. (First sum the data into appropriate bins, then average as
necessary, and finally interpolate as appropriate.)

I'm trying to do this "the IDL way" (IMHO) by calculating an index mapping
array and using this map to sum the data into appropriate bins. However,
the behavior isn't what I expect. A simple example:

```
IDL> a = fltarr(3)
IDL> map = [1, 1, 2]
IDL> a(map) = a(map) + [1.0, 2.0, 3.0]
IDL> print, a
     0.00000      2.00000      3.00000
```

Here "a" will hold the accumulated data and "[1, 1, 2]" is the index re-mapping
array (map the first and second data items to the second destination position
and map the third data item to the third destination position). When I try
to add the data "[1.0, 2.0, 3.0]" I had hoped that "1.0" and "2.0" would
have been summed into a(1) and "3.0" into a(2), resulting in
a = [0.0, 3.0, 3.0].

Why doesn't this work?

larry-granroth@uiowa.edu

Subject: Re: indexing arrays with arrays
Posted by GeoffS on Tue, 07 Aug 2007 19:25:08 GMT
View Forum Message <> Reply to Message

The only way I know to do that is using an "array of indices" to index
into the source array. For example, the answer to your specific
problem would be to create an array containing:

```
   indexes = [0,1, 1,2, 2,3, 3,4, 4,5]
```

then the extraction statement is simply:

```
   extract = res[indexes]
```

The extraction statement should be pretty efficient as it uses
implicit loops inside the IDL interpreter/VM. If you need to extract
the same elements many times, then this would be a reasonable
solution, but if you need an efficient/loopless way to create the
'indexes' array then it probably won't help you.

Cheers,

Geoff S.

On Aug 7, 8:39 am, Conor <cmanc...@gmail.com> wrote:
> It seems to me that a highly useful syntax for IDL would be something
> like this:
>
> res = randomu(seed,10)
> start_ind = indgen(5)
> end_ind = start_ind + 1
>
> extract = res[start_ind:end_ind]
>
> In this case, I would envision extract being a 10 element array.
> Namely, it would be the equivelent of:
>
> extract = [res[start_ind[0]:end_ind[0]], res[start_ind[1]:end_ind[1]],
> res[start_ind[2]:end_ind[2]], etc...]