Subject: Re: Help setting up an array
Posted by John-David T. Smith on Wed, 28 Mar 2001 22:16:24 GMT
View Forum Message <> Reply to Message

Peter Thorne wrote:
>
> Apologies if this is trivial, but I have been working for about 8 hours
> on it and can't even begin to see how to code it in IDL. Any help very
> gratefully received. I am setting up a function which receives the
> location of points in n-dimensional space for m fields, as well as their
> weights. On call the function does not know the size of any of these
> dimensions.
>
> Simplifying to m=1 (this m dimension should be trivial) the input is:
>
> an array of size (n x npoints) locations of each point in the
> n-dimensional space
>
> a vector of size (npoints) the weights.
>
> Now, what I want to be able to do is re-bin these points into an
> n-dimensional discretized space array which encloses all points. I can
> work out the limits of this space by simply finding min and max in each
> of the n dimensions of the location array. I then need to split this
> grid into nbox n-dimensional discrete boxes (say 50 divisions per
> dimension, boxes need not be of equal size in each dimension) and place
> the respective points in their boxes in this finite representation,
> weighted by weight (trivial).
>
> At present I am having two major problems:
>
> 1. How to declare this discretized array and the limits in n-dimensional
> space of each box into which I bin my values given that I know n and
> nbox.
>
> and doubtless much more difficult to overcome:
>
> 2. How to sensibly code selection criteria to ascertain whether a
> particular value belongs to a particular grid box.
>
> Perhaps it is not possible to code without an a priori knowledge of n -
> the dimensionality of the problem to know how many for loops to use? I
> can't see how where statements could be used.

It seems what you are trying to do is take a list of coordinates and values and construct a data cube or hyper-cube. For a concrete example, consider a list of tuples of the form:

x,y,z,density

You'd like to create a 3-d array with enough elements to adequately represent the density cube. This immediately begs the question: how is the density sampled, regularly or irregularly? If the answer is regularly, the solution is trivial:

```
s=size(c,/DIMENSIONS)
n=intarr(s[0])
for i=s[0]-1,0,-1 do begin
  ind=(c[i,*]-min(c[i,*]))/spacing[i]
  n[i]=max(ind)+1
  inds=n_elements(inds) eq 0?ind:ind+n[i]*inds
endfor
array=make_array(DIMENSION=n,type=size(w,/TYPE))
array[inds]=w
```

where "c" is the 3xn array of x,y,z coordinates (or dxn for your dimensionality), "w" is the n point vector of densities or weights, etc. And "spacing" is the n point vector of the regular spacing in each dimension.

E.g. suppose x runs from 0 to 20 with spacing .5, y runs from 0 to 50 with spacing 1, and z runs from 10 20 with spacing 2.

You'd set spacing=[.5,1,2] and you'd get a 41x51x6 array, with all the values plugged into the right place. Note that all this is doing is rearranging data which could otherwise have been constructed into a data cube in the first place.

If, as I suspect, you'd like to resample irregularly gridded higher-dimensional data, you'll need to think about what sort of sampling, and onto what size mesh you'd like to sample. There is no fundamental answer to these questions within the dataset itself, though you could of course come up with quantitative heuristics.

JD
