Subject: Re: Help setting up an array
Posted by Jaco van Gorkom on Thu, 29 Mar 2001 11:16:31 GMT
View Forum Message <> Reply to Message

Hi,
this question seemed interesting yesterday night, when no answers were present yet. Now, finally getting back to a network with IDL license and news reader, JD's little function takes all the fun out of it. So I am not even going to test the bits of code I came up with during dinner. Posting it anyway for the sake of completeness:

Peter Thorne wrote:
>
> Apologies if this is trivial, ...
Trivial?? Well, that is *one* way to put it, I suppose. It could be done cryptically in just two statements:

```
numberdensity = reform(histogram((pos-rebin(minpos,n,npoints,/sample))*$
  rebin(nbox^indgen(n)/(maxpos-minpos)/nbox,n,npoints,/sample) ,min=0,$

 max=nbox^n-1,binsize=1,reverse_indices=R),replicate(nbox,n), /overwrite)
weighteddensity = reform(convol(total(weight[R[R[0]:*]],/cumulative)$
  [R[0:R[0]-1]-R[0]],[1,-1],center=0),replicate(nbox,n),/overw rite)
```

Now the long version...
> 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
let's call this array POS. And use the SIZE() function to find out N and NPOINTS.
> a vector of size (npoints) the weights.
So this is the vector WEIGHT.
> 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.
This gives two vectors, each of size (npoints): I name them MINPOS and MAXPOS.
> 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)
I will use the scalar NBOX as the number of divisions per dimension,

e.g.
50. Although there is no real reason to take this number equal in each dimension, of course.
> and place
> the respective points in their boxes in this finite representation, ...

What you are trying to do here - counting the number of points in each of a large set of equally spaced bins - is basically to take a histogram over the positions. This is very fortunate, since the built-in function HISTOGRAM() seems to be the most often (mis-)used routine for vectorising all kinds of operations in IDL. It can do almost anything for you, without ever resorting to FOR-loops. And what HISTOGRAM cannot accomplish by itself, can be done by deciphering its keyword REVERSE_INDICES.

Here we don't need to do anything very fancy with HISTOGRAM. Basically, a simple histogram is all we need, be it over N dimensions. HISTOGRAM itself appears to be a 1D routine, but there is a technique for using it over multiple dimensions, used in HIST_2D for the 2D case (part of the IDL distribution). Now the best path (and highly recommended!) would be to generalise HIST_2D to PT_HIST_ND for the N-dimensional case. Taking the path of least resistance, let us just use their technique. Basically the trick is to scale all the position coordinates such that they exactly fit between 0 and NBOX (50), giving a box size which is normalised to 1. Then combine the N position coordinates into one big scalar using some kind of "NBOXimal system", much like different numbers 0 to 9 can be combined in the decimal system to form two-, three-, or N-digit numbers by multiplying the Nth number by 10^(N-1). This big single scalar position coordinate is actually the same as the linear subscripts of the N-dimensional array of boxes, and thus also equal to the linear subscript of the N-dimensional histogram that we are after.

That part was not very clear, was it? Anyway, here goes:
First make all position coordinates start at zero
```
  pos = pos - rebin(minpos, n, npoints, /sample)
```
Normalise them to the boxsixe:
```
  boxsize = (maxpos - minpos) / nbox
  pos = pos / rebin(boxsize, n, npoints, /sample)
```
Finally, combine all the N coordinates into one super-coordinate
(NBOXimal):
```
  pos = pos * rebin(nbox^indgen(n), n, npoints, /sample)
  pos = total(pos, 2)   ; sum over the N dimensions
```
Now pos is a vector of size (npoints). Of course some of these
statements
could have been combined for efficiency.

Now counting the number of points in each box is easy:
```
  numberdensity = histogram(pos, min=0, max=nbox^n-1, binsize=1,
reverse_indices=R)
```

> weighted by weight (trivial).
>

Weighting by weight is now not as trivial as it sounds. However,
REVERSE_INDICES comes to the rescue! This keyword will contain the
information on which points contributed to which box, although the way
it is
coded into one array tends to give me at least a slight headache. After
initialising the destination array to the same number of elements as
NUMBERDENSITY, you'd want to do something like
```
  for i=0, n_elements(numberdensity)-1 then $
    if R[i] ne R[i+1] then $
      weighteddensity[i] = total(weight[R[R[i]:R[i+1]-1]])
```
This just adds up the contributing elements of the WEIGHT array for each
box.

Several loopless, conditionless versions of the weighting procedure are
possible and usually faster:
```
  weights_cumul = total(weight[R[R[0]:*]],/cumulative)
  weighteddensity = weights_cumul[R[1:R[0]-1]-R[0]] $
                  - weights_cumul[R[0:R[0]-2]-R[0]]
```

If space has less than nine dimensions, then the space dimensions of the
array of boxes can be represented by IDL array dimensions:
```
  if n le 8 then $
    weighteddensity =
```
reform(weighteddensity,replicate(nbox,n),/overwrite)
Note that apart from this there is (almost) no limitation on N!

Just out of curiosity: did you create this little puzzle just to test
our
brain cells, or is there a real-world application for this problem?

cheers,
  Jaco
