Subject: Re: hist_nd question
Posted by JD Smith on Mon, 05 Mar 2007 19:16:22 GMT
View Forum Message <> Reply to Message

On Mon, 05 Mar 2007 16:17:09 +0100, Wox wrote:

> Hi all,
>
> I have been using JD's hist_nd before, but I just found something strange.
> Maybe this is already corrected somewhere or maybe this is a "feature",
> but N-dimensional points falling outside of the given min/max boundries,
> are mapped to the first and the last bin. To clarify:

Good catch. The problem is HIST_ND assumes that the 1D indices are unique, such that things outside the bounds of one axis "wrap into" the next dimension. This is not a problem if you let the code derive the N-dimensional bounding box itself, but if you hand specify a range with MIN/MAX, you can feel this issue. HIST_2D solves this by brute force setting to -1 any location which falls outside the bounds. I've modified HIST_ND to do something similar. Please find (and test) the updated version here:

turtle.as.arizona.edu/idl/hist_nd.pro

This version requires IDL v6.1 or later.

An interesting issue arose while making these changes. Comparing the newly slimmed-down loop-based version of the single index creation:

```
h=long((V[s[0]-1,*]-mn[s[0]-1])/bs[s[0]-1])
for i=s[0]-2,0,-1 do h=nbins[i]*h + long((V[i,*]-mn[i])/bs[i])
```

to the fully threaded, loop-free version:

```
h=total(long((V-rebin(mn,s,/SAMP))/rebin(bs,s,/SAMP)) * $
        rebin([1L,product(nbins[0:s[0]-2],/CUMULATIVE,/PRESERVE_TYPE )], $
            s,/SAMP), 1,/PRESERVE_TYPE)
```

I found the former to perform better by 20% or more for large arrays, which results purely from the overhead of REBIN'ing small arrays 3 times, to the size of the input array (e.g. 3 x 5 million for a large array), with it's resulting greater demands on memory. As is often pointed out, small loops which perform lots of work per iteration do not feel the loop penalty, and depending on the exact memory requirements, can actually excel over loop-free methods (yes, you hear me saying this).

JD