Subject: Re: Algorithm for lat/lon searching
Posted by JD Smith on Fri, 18 Aug 2006 17:54:29 GMT
View Forum Message <> Reply to Message

On Fri, 18 Aug 2006 10:50:56 -0400, Paul van Delst wrote:

> Hello,
>
> I want to implement a global *land* surface emissivity database (as a LUT)
> into a radiative transfer code. Fir simplicity the database is simply
> gridded by lat/lon (land and sea). Due to memory limitations, I want to
> only keep the land gridboxes in my lookup table. Obviously, doing this
> complicates searching for the actual lat/lon element since they're no
> longer stored on a grid.

Here's a simple notion:

Why not develop a "whole earth grid" in whatever binning and projection is useful (an equal area projection comes to mind), run all your land points (only) through HIST_ND, store the resulting REVERSE_INDICES, and then, for a given lat/lon, look up its position in the multi-dimensional reverse index vector, and read out the emissivity data points. You don't say how much information each of those emissivity data points would include, but storing a reverse index vector is linear in the number of bins, and would be much faster to access than sorting through constantly.

> p.s. Since the final code needs to be Fortran95, I set followups to
> comp.lang.fortran

Oh, well, that's a different story then, since you'd have to write your own HISTOGRAM from scratch ;). I guess it depends on your desired bin size, then. If you can bin the whole earth (or whatever portion thereof you're discussing) into say, 2^16 points, then map a given LAT/LON (or other more convenientq re-projected coordinate pair) to two 8bit integers (aka 1 16bit int, similar to what I showed for 64bit integers), you could keep a simple 65536 element hash table, each element of which would point to the data contained (using whatever F95 magic may or may not exist to do that: pointers to a linked lists would come to mind for us C programmers). You could implement more than 8bit of gridding per dimension, at the cost of memory. 10bits each (1024 elements) would only occupy about 4MB.

If you need much finer gridding, you could still use a coarse hashed grid of this form to get to the general area, but then perform a finer grained (e.g. bounding box) search through the data points pointed to in the indicated grid cell (or small set of adjacent grid cells).

JD