Subject: Re: a behemoth bubble sort
Posted by Yngvar Larsen on Wed, 31 Oct 2012 07:47:01 GMT
View Forum Message <> Reply to Message

On Monday, 29 October 2012 22:24:07 UTC+1, fisch...@gmail.com wrote:

- > Unfortunately, the bubble sort I've employed in this code needs to run
- > through 20 billion+ data points for the program to complete, which is of
- > course impossible.

Are you really running the code below on a cube with 20 billion points?

```
> My current codes is as follows:
[...]
> ;giant for-loop that looks at each individual voxel at each velocity
> :step and places the velocity at that voxel into the new cube's v-dimension.
> for v = min,max-1 do begin; v = velocity step
>
  for x = 0, x = 0 do begin
     for y = 0, ysize-1 do begin
>
       for z = 0, zsize-1 do begin
>
          if (nifs(x,y,z) eq v) then begin; if voxel has vth velocity step
>
             flux(x,y,v-min) = v; places v at the vth plane of flux cube
>
          endif
>
        endfor
>
     endfor
  endfor
> endfor
You can do this with only the outermost loop:
for v = min, max-1 do begin
 ind = where(nifs eq v, count)
 if (count gt 0) then begin
   i3d = array_indices(nifs, ind)
   flux[i3d[0,*], i3d[1,*], i3d[2,*]-min] = v
 endif
endfor
```

If your datacube is really 20 billion points, you should really divide this into subcubes. This is left as an exercise for the reader :)

Yngvar

View Forum Message <> Reply to Message

```
On 10/29/12 5:24 PM, fischertc13@gmail.com wrote:
> Hi all,
> I am currently frustrated trying to convert information from one data cube into another and could
use some direction.
> I have a 3-d modeling program which creates a spatial geometry datacube where each voxel in
the geometry contains a velocity. Voxels outside the geometry are assigned an artificially high
velocity that is set to be transparent in the modeling program. What I would like to do is to create a
datacube with dimensions of x, y, and velocity from the spatial data cube of x,y, and z.
> Unfortunately, the bubble sort I've employed in this code needs to run
> through 20 billion+ data points for the program to complete, which is of
> course impossible. Is there some way to simplify this? Also, is
> there some way to select out the 'good' portion of the initial data cube
> to apply the conversion to instead of the entire thing? You have posted on
> array-juggling similar to this, though after reading the article I was not
> able to apply the technique to my own problem. Any help would be much
> appreciated!
>
> Cheers,
> Travis
> My current codes is as follows:
>
> pro slice_run
>
  restore, 'nifscube.dat'; restores spatial datacube 'nifs'
>
>
> size = size(nifs,/dimensions)
>
 nifs = long(nifs); turns velocities to integers
>
  max = max(nifs); bad voxels are preset to artificially high maximum velocity
>
>
> xsize = size[0]
> ysize = size[1]
> zsize = size[2]
>
> good = where(nifs ne max, complement = bad)
> ; find where all true velocity data points exist,
> ; this is not employed elsewhere yet
```

```
>
 nifs(bad) = 0; set bad pixels to zero velocity
 min = min(nifs,max=max); find max/min true velocities
>
  nifs(bad) = max+1; reset bad pixels out of velocity range
>
  vsize = max-min; set velocity space range
>
> flux = fltarr(xsize, ysize, vsize); create new velocity data cube where z =
 velocity
  ; giant for-loop that looks at each individual voxel at each velocity
  ;step and places the velocity at that voxel into the new cube's v-dimension.
  for v = min, max-1 do begin; v = velocity step
>
>
  for x = 0, xsize-1 do begin
     for y = 0, ysize-1 do begin
>
        for z = 0, zsize-1 do begin
>
           if (nifs(x,y,z) eq v) then begin ; if voxel has vth velocity step
>
             flux(x,y,v-min) = v; places v at the vth plane of flux cube
>
          endif
>
        endfor
>
     endfor
>
> endfor
  endfor
> end
> Cheers,
> Travis
I have done almost exactly this sort of thing with HISTOGRAM (of course)
before. It should look something like (untested). I can come up with
non-loop solutions too, but I think they will die for memory reasons if
your data cube is really that big.
dv=1.0 ; velocity bin size
; find true velocity range
badveldata = max(nifs, min=minvel)
maxvel = max(nifs[where(nifs It badveldata)])
; dimensions of new array
size=size(nifs, /dimen)
```

```
xsize = size[0]
ysize = size[1]
vsize = ceil((maxvel-minvel)/dv) + 1
flux = fltarr(xsize, ysize, vsize)
; this vector contains the velocity of each
; element in the 3rd dimension, see comments
; within loop.
vvector = findgen(vsize)*dv + minvel
; iterate through x,y pixels
xynpix = long(xsize)*ysize
for i=01,xynpix-1 do begin
  ; turn 1D index into separate x and y indices
 xyi = array_indices([xsize,ysize], i, /dimen)
  ; use histogram to find out which velocity
  : slices at that x,y pixel contain elements
  ; (they are the ones that have a histogram count
  ; greater than 0)
  flux[xyi[0], xyi[1], *] = histogram(nifs[xyi[0],xyi[1],*], $
   min=minvel, max=maxvel, bin=dv) gt 0
  ; instead of a 1, it looks like you want the velocity to be
  ; stored there too? I'm not sure why, since that information
  ; is redundant with the location in the 3rd dimension, but
  ; you can do it by multiplying what is currently either a
  ; 1 or 0 by a vector containing the velocity of each element...
  flux[xyi[0],xyi[1],*] *= vvector
endfor
```