Subject: Re: algorithm question. Can I get rid of the for loop?
Posted by Jeremy Bailin on Wed, 27 Mar 2013 17:00:17 GMT
View Forum Message <> Reply to Message

Note, as alluded to in my earlier comment on this thread, this is a memory hog. It creates several arrays which each contain the number of elements that exist in every single window - if that is a large number, this will cause problems! Most of those are only used for temporary calculations, though, so they could be optimized away with judicious use of UNDEFINE once they're finished with. Chief candidates are: longarr, reph, repri, windownum, and nwindow_runningtot.

-Jeremy.

> Anything can be turned into a solution using histogram. ;-) You just
> need to create an array that *can* be binned. Which, of course, probably
> also involves histogram! It may be much less readable, and involve 3
> histograms including a histogram-of-a-histogram, but here's my
> replacement for your loop, which is about 3x faster in my tests:
>
>
>
> nwindow_per_element = upper_boundary - lower_boundary + 1
> nwindow_runningtot = total(nwindow_per_element, /cumulative, /int)
>
> ; we're going to treat all possible points in the window around
> ; each element as one gigantic long array:
> longarr = lindgen(nwindow_runningtot[ny-1])
>
> ; So we need easy ways
> ; of figuring out, for each point i in that long array, which
> ; element it is in the window of.
> ; The number of points in the window of each element is given
> ; by nwindow_per_element, so we can use that as the number
> ; of times to repeat each integer.
> ; Use the histogram trick to do the repeats - see the histogram tutorial
> reph = histogram(nwindow_runningtot-1, bin=1, min=0,$
> reverse_indices=repri)
> elementnum = repri[0:n_elements(reph)-1] - repri[0]
>
> ; We also need to know, for each point in the long array, what
> ; element in y it refers to. First figure out which point within
> ; the window that is:
> windownum = longarr - longarr[([0,nwindow_runningtot])[elementnum]]
> ; Then combine them with lower_boundary to figure out where in y that is
> ynum = lower_boundary[elementnum] + windownum
>

```
>  ; now histogram the elementnums so each window falls into
>  ; a separate bin
>  winh = histogram(elementnum, min=0, reverse_indices=winri)
>
>  ; and use the double-histogram trick so we only need to loop through
>  ; repeat counts instead of through elements - see the drizzling/chunking
>  ; page
>  h2 = histogram(winh, reverse_indices=ri2, min=1)
>  if h2[0] gt 1 then begin
>     ; single-element windows
>     vec_inds = ri2[ri2[0]:ri2[1]-1]
>     y0_jb[vec_inds] = y[ynum[winri[winri[vec_inds]]]]
>     s0_jb[vec_inds] = !value.f_nan
>  endif
>  for j=1,n_elements(h2)-1 do if h2[j] gt 0 then begin
>     ; windows of width j+1
>     element_inds = ri2[ri2[j]:ri2[j+1]-1]
>     vec_inds = rebin(winri[element_inds], h2[j], j+1, /sample) + $
>       rebin(transpose(lindgen(j+1)), h2[j], j+1, /sample)
>     y_temp = y[ynum[winri[vec_inds]]]
>     ; first dimension is which element, second is where in the window
>     ; so to do operations over the window, work on the second dimension
>     y0_jb[element_inds] = median(y_temp, dimension=2)
>     y0_temp = rebin(y0_jb[element_inds], h2[j], j+1, /sample)
>     s0_jb[element_inds] = 1.4826*median(abs(y_temp - y0_temp), dimension=2)
>  endif
>
>
>  -Jeremy.
>

I
```