
Subject: Re: How to find second minimum elements in an array in IDL?

Posted by [Conor](#) on Fri, 16 Jan 2009 15:08:51 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Jan 15, 5:39 pm, "mgal...@gmail.com" <mgal...@gmail.com> wrote:

> On Jan 15, 10:36 am, cmanc...@gmail.com wrote:

>

>

>

>> I was curious, so I checked out your routine Mike. It looks good but

>> one problem - a for loop! I'm pretty sure you can replace:

>

>> nCandidates = 0L

>> for bin = 0L, nBins - 1L do begin

>> nCandidates += h[bin]

>> if (nCandidates ge n) then break

>> endfor

>

>> with:

>

>> max(total(h, /cumulative) < n, bin)

>

>> which should work because max will return the first maximum value. Of

>> course, I was too lazy to see if the max(total()) method is actually

>> faster (since it involves a couple different computations), but oh

>> well, sometimes laziness wins :)

>

> It turns out that it probably doesn't matter much.

>

> It's not FOR loops per se that are bad, but the execution of many

> statements. For perfectly uniformly distributed data, the FOR loop

> above will only loop once -- more times the less uniformly distributed

> the data, bounded by the number of bins (i.e. number of data

> elements / number of elements required).

>

> Averages were computed for 500 runs of finding the smallest k=100

> elements of an n=1000000 element dataset.

>

> For uniform data:

>

> mg_n_smallest(randomu(seed, n), k)

>

> vectorized: 0.035663 seconds

> loops: 0.036040 seconds

> loops are 1.1% faster

>

> For perverse data:

>

```
> mg_n_smallest([randomu(seed, k - 1), randomu(seed, n - k + 1) + n /  
> k], 100)  
>  
> vectorized: 0.279783 seconds  
> loops: 0.281627 seconds  
> vectorized is 0.7% faster  
>  
> Mike  
> --www.michaelgalloy.com  
> Tech-X Corporation  
> Associate Research Scientist
```

I didn't really expect much of a difference. I think this is just a personal preference of mine - it looks so much nice when it all fits on one line!
