Subject: Re: Find minimums in an array...
Posted by JD Smith on Fri, 18 Oct 2002 16:04:04 GMT
View Forum Message <> Reply to Message

On Wed, 16 Oct 2002 14:00:35 -0700, James Kuyper wrote:

> David Fanning wrote:
>>
>> James Kuyper (kuyper@saicmodis.com) writes:
>>
>>>> P.S. Let's just say that *this* is what enlightenment feels like!
>>>> :-)
>>>
>>> I feel somewhat unenlightened. Could you be a little more specific?
>>
>> Well, I just thought "ten lowest values", what a perfect job for the
>> histogram function. Divide your array into values, then Histogram it,
>> and starting looking in the lowest bins. Keep going until you find 10
>> values! Find the locations by using REVERSE_INDICES.... Oh, oh. Hang on
>> a second, I've got to go check with the guru...
>>
>>   http://www.dfanning.com/tips/histogram_tutorial.html
>
> I don't believe that the indices, within each bin, refer to the array
> entries in sort order. So you'll have to at least sort the elements of
> the last bin, to get precisely the 10 smallest items. Unless including
> that bin brings the count to precisely 10.
>
> The efficiency of such an algorithm will depend upon the bin size. The
> smaller the bin size, the fewer items you'll have to sort in the final
> bin, but there will be a corresponding increase in the number of bins
> you'll have to look at. The proper choice will depend a lot upon the
> distribution of the data. For very clumpy data, this approach won't be
> significantly faster than the sort-based approach.

Au contraire. SORT comes to mind, but is it the fastest method? It's
certainly straightforward looking:

```
a=randomu(sd,1000)

s=sort(a)
vals=a[s[0:9]]
```

It's done some unnecessary work putting all those elements in order
(true *selection* on an array is of order N, sorting is N log(N), but
IDL doesn't have a true selection routine). Can we improve the speed
with HISTOGRAM? The answer to that question is usually "yes", but it
comes at the expense of readability:

```
  h=histogram(a,NBINS=1000/10,REVERSE_INDICES=ri)
  s=0L
  for j=0L,n_elements(h)-1L do begin
    s=s+h[j]
    if s ge 10 then break
  endfor
  vals2=a[ri[ri[0]:ri[j+1]-1]]
  vals2=(vals2[sort(vals2)])[0:9]
```

I've just taken the first few bins of the histogram, such that we get at least 10 elements, and done a much smaller SORT on the data in those bins to find the smallest 10. Notice how I arranged for there to be 100 bins for this 1000-point vector, so that, for randomly distributed data anyway, you'd have 10 elements per bin on average. The run-time is not terribly sensitive to this number, as long as you divide the data up into some decent number of bins (more than 1 ;). E.g. I could just pick NBINS=100 always. Bottom line -- SORT is slow, HISTOGRAM is fast.

Also notice that I used a FOR loop to find the first bin where you've hit at least 10 elements cumulative. I expect this to be near the front, so this is much faster than a loop-free solution like: j=(where(total(h,/CUMULATIVE) ge 10))[0].

Here are the timings (in sec):

1000-point Uniform Random Data:
a=randomu(sd,1000)

```
 SORT METHOD:          0.000612
 HISTOGRAM+SORT METHOD: 0.000151
```

A nice 4x speedup.

1,000,000-point, Poisson Distributed Data:
a=randomn(sd,1000000,POISSON=100)

```
 SORT METHOD:          1.7388
 HISTOGRAM+SORT METHOD: 0.1453
```

A 12x speedup.

100,000-point, Bizarrely Clumpy Data:

```
a=[4+randomu(sd,2500),10+randomu(sd,2500), $
  150+randomu(sd,2500),200+randomu(sd,2500)]

 SORT METHOD:          0.0966
```

HISTOGRAM+SORT METHOD: 0.0377

A 2.6x speedup.

As always, you're mileage may vary. Check the performance on your equipment, using your data.

Also, lest everyone think I'm out to turn their code into piles of HISTOGRAM spaghetti: just because a fast HISTOGRAM/REBIN/REFORM solution exists, doesn't mean you should always use it (though sometimes, HISTOGRAM is the simplest approach from the beginning). The SORT method above has much clearer intent. If you're dealing with a small data set, don't loop over it many times, or aren't in a time-critical section of the code, readability and maintainability may be more compelling than speed. I typically code for readability (with some care taken to avoid trivial FOR loops), and then make another pass optimizing critical sections of the code with these techniques (carefully commenting of course ;).

Good luck,

JD
