Subject: Re: Finding the closest value in an array...
Posted by JD Smith on Tue, 30 Mar 2004 18:04:00 GMT
View Forum Message <> Reply to Message

On Tue, 30 Mar 2004 01:34:07 -0800, Tim Robishaw wrote:

> Hi there.
>
> Seems like every few minutes I'm taking a scalar and trying to locate
> which value in an array it's closest to. VALUE_LOCATE() finds the
> interval of a monotonic vector that the value lives in, so it's not
> quite what I'm looking for, but it's awfully close! I end up just
> doing this:
>
> IDL> useless = min(abs(vector-value),minindx)
> IDL> closest = vector[minindx]
>
> I'm embarrassed to admit I don't know of any other way to do this. Is
> there some slick way like VALUE_LOCATE() to do this? I find it
> aesthetically unpleasant to have to set something to a useless value
> just to get at the corresponding index; however, I can't see any way
> to be clever about it. And it's pretty much to the point: I'd bet
> VALUE_LOCATE() is doing a lot more stuff behind the scenes than the
> simple two lines above (judging from the old Goddard library routine).
>
> I guess I'm surprised that I haven't found some canned routine for
> this (like in the Goddard library) given that I usually need to find
> closest values more often than intervals in which a value lives.

For monotonic arrays, you know either one or the other of the two bracketing values is the closest. VALUE_LOCATE is faster than MIN(ABS()) since it relies on the monotonicity to skip rapidly through the vector using bisection. This doesn't address your aesthetic concerns, but it's much more efficient:

```
j=value_locate(r,find)
mn=min(abs(r[j:j+1]-find),pos)
pos+=j
```

When compared to:

```
mn=min(abs(r-find),pos)
```

the former can be *much* faster, especially for long arrays. While the latter is linear in N, the former is logarithmic. For long vectors, the speedup is tremendous:

```
r=total(randomu(sd,2000000),/CUMULATIVE,/DOUBLE)
```

```
  find=max(r)/10.

  time2/time1=1230.2660
```

You can realize even bigger gains when searching for locations closest
to more than one value at once:

```
  n=2000000
  r=total(randomu(sd,n),/CUMULATIVE,/DOUBLE)
  find=max(r)*(findgen(20)/19

  j=value_locate(r,find)
  j=transpose(j)
  b=[j>0,(j+1)<(n-1)]
   mn=min(abs(r[b]-rebin(transpose(find),2,n_elements(find))),D IMENSION=1,pos2)
  pos2=j>0+(pos2 mod 2)
```

Here I've explicitly accounted for the first or last element of r
being the closest (which technically you should do even in the single
find value case). In this example, the speedup is >13000.

How about a really tough one:

```
  n=10000000
  r=total(randomu(sd,n),/CUMULATIVE,/DOUBLE)
  find=max(r)*(findgen(300)/299
```

In this case, the VALUE_LOCATE method is 126859x faster!

Anyway, it's probably worth putting this altogether in a function call,
like:

```
;; Find indices closest to find values in vector, which must be
;; monotonically increasing or decreasing, otherwise a sort vector
;; should be passed.  Find can be a vector itself.
function closest,vector,find,SORT=s
  nf=n_elements(find)
  sort=keyword_set(s) || arg_present(s)
  if sort && n_elements(s) ne n_elements(vector) then s=sort(vector)
  j=value_locate(sort?vector[s]:vector,find)
  b=[[j>0],[(j+1)<(n_elements(vector)-1)]]
  mn=min(abs((sort?vector[s[b]]:vector[b])- $
            rebin([find],nf,2)),DIMENSION=2,pos)
  pos=j>0+pos/nf
  return,sort?s[pos]:pos
end
```

This version allows you to pass a sort vector (or have it defined for

you on the first pass) for non-monotonic arrays. Note, however, that if you have to sort your array first, and are only finding a single value, there won't be much gain (and potentially loss) over the MIN(ABS()) method.

JD
