Subject: Re: Finding the closest value in an array...
Posted by Mark Hadfield on Tue, 30 Mar 2004 10:13:20 GMT
View Forum Message <> Reply to Message

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.
> -Tim.

My Motley library at

http://www.dfanning.com/hadfield/idl/README.html
http://www.dfanning.com/hadfield/idl56/README.html

has a routine called MGH_LOCATE which locates a one or more specified values in the "index space" of a 1D array. The result is a floating value, which you can then treat with FLOOR, CEIL or ROUND to get the integer index immediately below, immediately above, or closest. There is also a 2D counterpart called MGH_LOCATE2.

```
IDL> print, mgh_locate(findgen(11)^2, XOUT=30)
      5.45455
IDL> print, round(mgh_locate(findgen(11)^2, XOUT=30))
           5
```

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"

m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Finding the closest value in an array...
Posted by Wayne Landsman on Tue, 30 Mar 2004 16:54:47 GMT
View Forum Message <> Reply to Message

Tim Robishaw wrote:

> 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]

Presuming your vector is not monotonic, then your two lines perform the minimum necessary number of calculations (and are completely vectorized) -- you have to check every value in vector. Though you don't need to save the "useless" scalar value returned by MIN, this value did need to be calculated, and the extra overhead is very MINimal ;-)

If you have a monotonic vector then -- like VALUE_LOCATE() -- you can use bisection to minimize the number of values in vector that you need to check.

> 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).

You probably saw Craig Markwardt's implementation of the VALUE_LOCATE algorithm for users of IDL V5.2 and earlier, with its bizarre call SPL_INTERP. The program is not really performing cubic interpolation, but rather taking advantage of the bisection algorithm within the intrinsic SPL_INTERP function. Even with the unnecessary spline calculations this method is still faster than coding the bisection algorithm within IDL. Since V5.3 the bisection algorithm is available directly in the VALUE_LOCATE function.

--Wayne Landsman

Subject: Re: Finding the closest value in an array...
Posted by JD Smith on Tue, 30 Mar 2004 18:04:00 GMT

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

Subject: Re: Finding the closest value in an array...
Posted by timrobishaw on Wed, 31 Mar 2004 08:41:59 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> wrote in message
> 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.

Hi JD. Thanks for the advanced cleverness. That is great! That factor of 130,000 in speed is wicked awesome! So, if I do a few tests and find that the MIN(ABS()) method is faster for the case when FIND only has one element, should I (would you) add an if/then to check for this case and perform the two-line MIN(ABS()) evaluation so that the slower SORT/MIN/ABS/REBIN method is avoided? I haven't really been too aware of efficiency issues, but I'm starting to do LOTS of reduction on BIG data sets, so I'd better start thinking about this stuff! Thanks a bunch -Tim.

> ;; 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
```