Subject: Re: Array manipulation
Posted by Jaco van Gorkom on Wed, 01 Nov 2000 17:59:16 GMT
View Forum Message <> Reply to Message

Hi,
I cannot suggest a general trick for chopping the array using subscripting.
If the mean is all that you are after, using IDL's built-in "rebin" function
for neighbourhood averaging is probably the fastest:

```
ArrDims = size(InputArr, /dimensions)
Average = rebin(InputArr, ArrDims[0]/Width, ArrDims[1]/Width)
```

This assumes a two-dimensional InputArr of which you have already forced the
dimensions to be a multiple of Width, like in your code.
This method does not allow to leave certain elements (e.g. <minval, >maxval,
or NaN) out of the calculation: NaN in one of the elements will cause the
average for an entire block to be NaN. I suppose this could be avoided by
setting the unwanted elements to zero so they do not really add to the mean,
and then applying an array of correction factors. So if e.g. 3 out of 9
elements in a block were set to zero, the calculated average from rebin
should be multiplied by 1.5 or, alternatively, divided by the fraction of
valid elements in the block. For just NaN-correction the code would look
something like this:

```
ArrDims = size(InputArr, /dimensions)
CorrArr = $
  rebin(float(finite(InputArr)), ArrDims[0]/Width, ArrDims[1]/Width)
InputArr[where(finite(InputArr, /NaN))] = 0
Average = rebin(InputArr, ArrDims[0]/Width, ArrDims[1]/Width)
CorrArr[where(CorrArr eq 0.)] = !values.f_nan    ; to avoid division by zero
Average = Average / CorrArr
```

I don't know if this will still be faster than smart subscripting in a loop,
it probably depends on the size of the arrays. It also uses somewhat more
memory. I hope that someone else knows a better solution. It would be best
for this type of NaN-handling to be a built-in option of IDL-routines like
rebin, smooth, etc. I would certainly be using it!

Rebin works for getting the mean value of each box. Maybe for getting the
median of each box one could use the built-in "median" filter function,
followed by a resampling of the array with rebin(,/sample). Median
calculates a median filter for each element of the original array, which is
much more and maybe much slower than what you need. Because of the way rebin
resamples, you may need to shift the array first by half the box width.
Alternatively, do the resampling yourself by, once again, clever
subscripting.

Jaco

Jaco van Gorkom
FOM-Instituut voor Plasmafysica "Rijnhuizen", The Netherlands
e-mail: gorkom@rijnh.nl

"Leon Majewski" <majewski@cygnus.uwa.edu.au> wrote in message news:39ffa4ae.1813674@news.uwa.edu.au...
> Hello
> I was wondering whether any array minded person could suggest a way of using array
> indicies to chop up a large array into ordered windows.
> I can't think of a way to do it with reform, translate (though i'm sure this is my
> limitation not a reform translate limitation)
>
> ---
> ie given an array of 30*30 elements
> return 100 3*3 elements
> or 36 5*5
> or...
>
> in=
>
> 00 01 02 03 04 05..
> 30 31 32 33 34 35..
> 60 61 62 63 64 65..
>
> out = blocks such as
> 00 01 02
> 30 31 32
> 60 61 62
>
> each block is then processed to one representative number (ie mean or median....) and
> returned
>
> ---
> What i've used so far is attached below (it does what i want, just slowly)
>
>
> leon