Subject: Re: efficient comparing 1D and 3D arrays
Posted by Craig Markwardt on Wed, 11 Jun 2008 15:09:25 GMT
View Forum Message <> Reply to Message

Jelle <post@bio-vision.nl> writes:

> Hi All,
>
> At the moment I am trying to find pixels that fall within a certain
> value range for each pixel, as part of a recursive image exploration
> routine.
>
> Say I have the following data:
>
> imgdata = fltarr(NB, NS, NL)
> MinVals = fltarr(NB)
> MaxVals = fltarr(NB)
>
> Now I would like to efficiently find out
> where((imgdata GT MinVals) and (imgdata LT MaxVals))
>

There are two possibilities. One is to REFORM/REBIN your MinVals and MaxVals arrays so they are the same dimension as imgdata, then you can do your comparison directly.

The other possibility is to make a FOR loop. If NS*NL is large, then the overhead of the loop should be irrelevant since you are doing many vector comparisons at each loop step.

Good luck!
Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@REMOVEcow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jelle on Wed, 11 Jun 2008 15:46:17 GMT
View Forum Message <> Reply to Message

On Jun 11, 4:09 pm, Craig Markwardt
<craigm...@REMOVEcow.physics.wisc.edu> wrote:
> Jelle <p...@bio-vision.nl> writes:
>> Hi All,

>
>> At the moment I am trying to find pixels that fall within a certain
>> value range for each pixel, as part of a recursive image exploration
>> routine.
>
>> Say I have the following data:
>
>> imgdata = fltarr(NB, NS, NL)
>> MinVals = fltarr(NB)
>> MaxVals = fltarr(NB)
>
>> Now I would like to efficiently find out
>> where((imgdata GT MinVals) and (imgdata LT MaxVals))
>
> There are two possibilities. One is to REFORM/REBIN your MinVals and
> MaxVals arrays so they are the same dimension as imgdata, then you can
> do your comparison directly.
>
> The other possibility is to make a FOR loop. If NS*NL is large, then
> the overhead of the loop should be irrelevant since you are doing many
> vector comparisons at each loop step.
>
> Good luck!
> Craig
>
> --
> -- --------------
> Craig B. Markwardt, Ph.D.   EMAIL: craigm...@REMOVEcow.physics.wisc.edu
> Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
> -- --------------

Hi Craig,

Thank you for your reply. I am struggling a bit with your suggestion..

If I do a rebin on the data, I realize I can do a 'bandwise-
comparison' or an pixel-based comparison, but I could not do a
straight 3D comparison, could I?

[example snip]

; A lot of processing gets a set of boundary pixels, in WorkData.
; Here I compare for each pixel whether the band values fit between
two sets of ranges:

 Valid1 = DoComp(WorkData, Segments[SegmentID].MinVals, 'GE')
 Valid2 = DoComp(WorkData, Segments[SegmentID].MaxVals, 'LE')

```
 if(valid1 EQ 1 && valid2 EQ 1) then begin
   ; Add the pixels to the current segment & rerun routine
 endif else begin
   ; Stop growing, get new seed & start over
 endelse

; ----

; function to compare two arrays
function DoComp, Arr1, Arr2, Comp, ArrayComp = AC

IF KEYWORD_SET(AC) then begin

; we are performing the array comp with a multidim input array
  dims = size(Arr1[*,0], /dimensions)
  valid = intarr(dims[0])

  for i=0, dims[0]-1 do begin
    valid[i] = DoComp(Arr1[i,*], Arr2, Comp)
  endfor

return, valid

endif else begin

  case Comp of
    'LE': begin
      ; Is Arr1 LE Arr2
      less = where((Arr2-Arr1) LT 0, count)
      return, (count) GT 0 ? 0 : 1
    end

    'GE': begin
      less = where((Arr1-Arr2) LT 0, count)
      return, (count) GT 0 ? 0 : 1
    end
  endcase

endelse

end
```

[end example snip]

Basically, I need each band to fall between the limits set for that band, and know which pixels match that criteria. I was doing it in a loop, where I compare boundary pixels against the minvals/maxvals. But as my ROI's get bigger, the number of times you compare the same

pixels increases. So I thought I'd do the comparison for the whole image in one go, keep a binary layer with acceptable / not acceptable and use that as a masking template.

As this is a new segmentation routine I am designing, the individual comparison might have to be done 100,000+ times in complex landscapes, which is why I now would like to take it out of the loop.

Hope you can elaborate a little bit, as I cannot see how to make this comparison work without looping through all bands and pixels..

cheers,

Jelle

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jean H. on Wed, 11 Jun 2008 16:37:55 GMT
View Forum Message <> Reply to Message

>>> At the moment I am trying to find pixels that fall within a certain
>>> value range for each pixel, as part of a recursive image exploration
>>> routine.
>>> Say I have the following data:
>>> imgdata = fltarr(NB, NS, NL)
>>> MinVals = fltarr(NB)
>>> MaxVals = fltarr(NB)
>>> Now I would like to efficiently find out
>>> where((imgdata GT MinVals) and (imgdata LT MaxVals))
>> There are two possibilities. One is to REFORM/REBIN your MinVals and
>> MaxVals arrays so they are the same dimension as imgdata, then you can
>> do your comparison directly.

> If I do a rebin on the data, I realize I can do a 'bandwise-
> comparison' or an pixel-based comparison, but I could not do a
> straight 3D comparison, could I?

Hi,
Yes you can...do something like:
nb=3
ns=5
nl=5
imgData = fix(randomu(seed,nb,nl,ns)*100) ;--> doing ns,nl,nb makes more sense than nb,ns,nl... for mental representation at least (and if you print it!)

minVals = [15,30,12] ;Min val in each band
maxVals = [75,80,60] ;Max val in each band

allMin = rebin(minVals, nb,nl,ns) ;repeat the min band value for every
pixels in each band
allMax = rebin(maxVals, nb,nl,ns)

goodPixels = where(imgData gt allMin and imgData lt allMax)

==> returns the indexes of imgData that satisfy the condition in EVERY band.

Jean

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jelle on Wed, 11 Jun 2008 16:48:24 GMT
View Forum Message <> Reply to Message

On Jun 11, 5:37 pm, Jean H <jghas...@DELTHIS.ucalgary.ANDTHIS.ca>
wrote:

> Hi,
> Yes you can...do something like:
> nb=3
> ns=5
> nl=5
> imgData = fix(randomu(seed,nb,nl,ns)*100) ;--> doing ns,nl,nb makes
> more sense than nb,ns,nl... for mental representation at least (and if
> you print it!)
>
> minVals = [15,30,12] ;Min val in each band
> maxVals = [75,80,60] ;Max val in each band
>
> allMin = rebin(minVals, nb,nl,ns) ;repeat the min band value for every
> pixels in each band
> allMax = rebin(maxVals, nb,nl,ns)
>
> goodPixels = where(imgData gt allMin and imgData lt allMax)
>
> ==> returns the indexes of imgData that satisfy the condition in EVERY band.
>
> Jean

cool. I'll harvest some beans in the garden, have dinner, pour a red,
have a potter and let the results of my pottering seep through!

J
PS: Yes dinner.. I am in Europe: Dinner time, 6PM in UK

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jelle on Wed, 11 Jun 2008 17:29:23 GMT
View Forum Message <> Reply to Message

On Jun 11, 5:37 pm, Jean H <jghas...@DELTHIS.ucalgary.ANDTHIS.ca>
wrote:
> Hi,
> Yes you can...do something like:
> minVals = [15,30,12] ;Min val in each band
> maxVals = [75,80,60] ;Max val in each band
>
> allMin = rebin(minVals, nb,nl,ns) ;repeat the min band value for every
> pixels in each band
> allMax = rebin(maxVals, nb,nl,ns)
>
> goodPixels = where(imgData gt allMin and imgData lt allMax)
>
> ==> returns the indexes of imgData that satisfy the condition in EVERY band.
>
> Jean

:(unfortunately it does not return the indices I was hoping for. It
returns an array with between 0 and 75 elements, for each individual
element. So it tells me for which pixel which band matches. This I
could of course use to calculate the array_indices, subset by only
row/colum indices and reverse back into 1D positional elements, make
uniq() and then create the layer, but I am hoping there is a fell
swoop with which I can do this in one step; a way to create a [ns, nl]
bytearr with 0/1 to match the condition that each band for that bixel
falls within the desired range...

J

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jean H. on Wed, 11 Jun 2008 20:46:45 GMT
View Forum Message <> Reply to Message

Jelle wrote:
> On Jun 11, 5:37 pm, Jean H <jghas...@DELTHIS.ucalgary.ANDTHIS.ca>
> wrote:
>> Hi,
>> Yes you can...do something like:
>> minVals = [15,30,12] ;Min val in each band
>> maxVals = [75,80,60] ;Max val in each band
>>
>> allMin = rebin(minVals, nb,nl,ns) ;repeat the min band value for every

>> pixels in each band
>> allMax = rebin(maxVals, nb,nl,ns)
>>
>> goodPixels = where(imgData gt allMin and imgData lt allMax)
>>
>> ==> returns the indexes of imgData that satisfy the condition in EVERY band.
>>
>> Jean
>
> :(unfortunately it does not return the indices I was hoping for. It
> returns an array with between 0 and 75 elements, for each individual
> element. So it tells me for which pixel which band matches. This I
> *could* of course use to calculate the array_indices, subset by only
> row/colum indices and reverse back into 1D positional elements, make
> uniq() and then create the layer, but I am hoping there is a fell
> swoop with which I can do this in one step; a way to create a [ns, nl]
> bytearr with 0/1 to match the condition that each band for that bixel
> falls within the desired range...
>
>
> J

Ok, I advise you to read the histogram tutorial on David's website to
fully understand the following:

nb=3
ns=5
nl=5
imgData = fix(randomu(seed,nb,nl,ns)*100)

minVals = [15,30,12] ;Min val in each band
maxVals = [75,80,60] ;Max val in each band

allMin = rebin(minVals, nb,nl,ns) ;repeat the min band value for every
pixels in each band
allMax = rebin(maxVals, nb,nl,ns)

goodPixelsPerBand = where(imgData gt allMin and imgData lt allMax)

tmp = where(histogram(goodPixelsPerBand/nb, min=0, R=ri) eq nb)
;==>make sure nb is an integer, NOT a float!!!

Divide the subscript by NB to "remove" the dimension of the band. Then,
do an histogram on it. If you have NB times an entry, it means these
pixels satisfy the condition in all NB bands. Keep track or the
reverse_indices to find out which entry in goodPixelsPerBand satisfied
the conditions.

goodPixels = goodPixelsPerBand[ri[ri[tmp]]]
;get back to the original indices. Since you only query on ri[tmp], you
will only get the first index in GoodPixelsPerBand that satisfy all
condition (i.e. not the index in every band)...

Now, there might be another solution with total() on goodPixelsPerBand/NB

Jean

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Chris[5] on Wed, 11 Jun 2008 21:24:04 GMT
View Forum Message <> Reply to Message

> I am hoping there is a fell
> swoop with which I can do this in one step; a way to create a [ns, nl]
> bytearr with 0/1 to match the condition that each band for that bixel
> falls within the desired range...

Let me try to understand what you are trying to do:
-data is a datacube of dimensions (nb,ns,nl).
-for each pixel along the first dimension (the one with nb elements),
you want to test whether it is greater than minval and less than
maxval. These are functions of where you are along the first
dimension, so minval and maxval are vectors of length nb.
-you want to create an array, of size ns x nl, such that result[x,y]=1
if data[i,x,y] falls between minval[i] and maxval[i] for all i.

Is this a correct summary? If so, I would recommend:

pro test

nb=3
ns=2
nl=2

data=randomu(seed,nb,ns,nl);just make up data
minval=fltarr(nb)+.1
maxval=fltarr(nb)+.9

;make cubes out of these
minval=rebin(minval,nb,ns,nl)
maxval=rebin(maxval,nb,ns,nl)

print,cube

```
hit=(data gt minval) and (data lt maxval)
result=total(hit,1) eq nb

print,result

end
```

Explanation:

You turn minval and maxval into cubes such that every pixel in the data cube needs to be between the corresponding pixels in minval and maxval. You then do a pixel by pixel test of this, returning the 'hit' cube (ones and zeros). You only care about cuts through the first dimension which satisfy your bounds for every pixel along that dimension. Thus, you sum up the cube along the first dimension (returning an ns by nl array), and test whether or not it equals nb (meaning every pixel was a hit).

Aside:
I was doing something similar this weekend, and agree with the earlier poster about weighing the pros and cons of looping vs large array creation. If you have lots of pixels, allocating the memory for the minval and maxval cubes will take some time. Looping through every pixel in the cube is, of course, a dumb thing to do in idl - it would rather work with arrays than individual elements. However, if you loop through each PLANE (say, step along the fist dimension), and then at each step in the loop analyze a 2D array, you will balance IDL's efficiency at working with arrays with the time penalty associated with allocating big chunks of your system memory. I have found (to my surprise) that this kind of looping can be much, much faster than avoiding the loop altogether. I am trying to quantitatively understand this behavior more (there are lots of qualitative descriptions of this here and on David Fanning's website), but the moral seems to be the following: IDL's 'sweet spot' is to do operations on arrays as large as possible, AS LONG AS any memory allocation you need to do to allow such a procedure is small (say a few percent of the total memory you have available).

-chris

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jelle on Wed, 11 Jun 2008 23:05:17 GMT
View Forum Message <> Reply to Message

Hi Chris,

Thanks for your reply. You were spot on with summarizing my ramblings

So.. Thinking about this a bit more.. I was wondering about the memory issues too, as, as you pointed out; allocating memory takes time. And my images are not super large, but still I am working with an 14*1500*1200 values data array. So possible it might be useful to just do it over a subset of the image, in sections. Or do it for the area that is being looked at, with a trigger when the area being looked at passes a certain size, that I start working in image tiles.

ok, I at least know I am not overlooking an obvious think here. Vectorizing my routines has never been my forte, so I thought I'd check before switching the routine on on a real image, and having to wait for days!

Jelle.

Subject: Re: efficient comparing 1D and 3D arrays
Posted by Jean H. on Wed, 11 Jun 2008 23:21:38 GMT
View Forum Message <> Reply to Message

Jelle wrote:
> Hi Chris,
>
> Thanks for your reply. You were spot on with summarizing my ramblings
>
> So.. Thinking about this a bit more.. I was wondering about the memory
> issues too, as, as you pointed out; allocating memory takes time. And
> my images are not super large, but still I am working with an
> 14*1500*1200 values data array. So possible it might be useful to just
> do it over a subset of the image, in sections. Or do it for the area
> that is being looked at, with a trigger when the area being looked at
> passes a certain size, that I start working in image tiles.
>
> ok, I at least know I am not overlooking an obvious think here.
> Vectorizing my routines has never been my forte, so I thought I'd
> check before switching the routine on on a real image, and having to
> wait for days!
>
> Jelle.

You said you were applying this under a ROI... one thing you can do it to use the data only under the ROI, and dismiss the other pixels. Basically, you will change each band from 2D to 1D (so any shape would be accommodated). Just keep an index so you can map back your results to the original image.

Jean