Subject: Re: Maximum value array resampling
Posted by b_efremova@yahoo.com on Fri, 05 Aug 2005 19:18:41 GMT
View Forum Message <> Reply to Message

Hi,
I'm not able to do it without loops, but I can do it with less loops.
Here's a program that makes it in two ways. The first is what I think
you probably are doing, and the second is twice faster for an array
2048x2048 ...you probably figured that much yourself, but since I did
it... here it is:

```
PRO test,im,y1,y2
x=im
ss=size(im,/dimensions)
n1=ss[0]/2.
n2=ss[1]/2.

y1=fltarr(n1,n2)
y2=y1
;---------  first way-----------
t10=systime(1)
   for i=0,n1-1 do for j=0,n2-1 do begin
        y1[i,j]=x[2*i,2*j] > x[2*i+1,2*j] > x[2*i,2*j+1]>x[2*i+1,2*j+1]
   end
print,'time1=',systime(1)-t10

;--------- secon way-------------
t20=systime(1)
  for i=0,n1-1 do x[i,*]=x[2*i,*] > x[2*i+1,*]
  for i=0,n2-1 do y2[*,i]=x[0:n1-1,2*i] > x[0:n1-1,2*i+1]
print,'time2=',systime(1)-t20

;------
END
```

Subject: Re: Maximum value array resampling
Posted by Richard French on Fri, 05 Aug 2005 19:59:16 GMT
View Forum Message <> Reply to Message

On 8/5/05 1:55 PM, in article
1123264545.837438.93790@g47g2000cwa.googlegroups.com, "rechoncho@yahoo.com"
<rechoncho@yahoo.com> wrote:

> I'm trying to figure out an IDL-efficient way to resample a series of
> images. I know how to do this is a ruinously laborious fashion using
> loops but I know there's any easier way.
>

> Consider the following 4x4 array:
>
> x = [$
> [0,3,4,5],$
> [1,2,7,0],$
> [3,2,9,0],$
> [7,0,5,6]]
>
> I want to resample this to y, a 2x2 array. Each element would contain
> the maximum value of the corresponding 4 pixels. y would then look like
>
> [$
> [3,7],$
> [7,9]]
>
> So element [0,0] in y is max(x[0:1,0:1]) etc. As I understand it,
> rebin/congrid won't do this. Each image is about 5000x2000 and there
> are several hundred to process.
>
> Thanks!
>

Try this (I don't know how it scales with image size, but give it a try!)

The basic idea is to make four rebinned copies of the image with the upper left, upper right, lower left, and lower right pixels in each 2x2 pair and to replace the array elements by the largest value as you compare each of the rebinned images in turn. Probably you can cut down on memory use by using TEMPORARY() and there may be other increases in efficiency as well. Good luck!

Dick French

```
x=[$
[0,3,4,5,3,7],$
[1,2,7,0,2,9],$
[3,2,9,0,4,2],$
[7,0,5,6,1,5]]

Print,x

; get size

size=size(x,/DIM)

nx=size[0]
ny=size[1]
```

```
nx2=nx/2
ny2=ny/2

; get indices of upper left element of each 2x2 cell

 l=rebin(nx*2#lindgen(ny2),nx2,ny2)+rebin(lindgen(nx2)#2,nx2, ny2)


; get indices of ul,ur, ll, lr of each 2x2 cell
offsets=[0,1,nx,nx+1]

for n=0,3 do begin
        xtemp=rebin(x[l+offsets[n]],nx,ny,/sample)
        x=x+((xtemp-x)>0)   ; replace x by current largest value
endfor

xfinal=rebin(x,nx2,ny2)
print
print,xfinal

End

IDL> .run trythis
       0       3       4       5       3       7
       1       2       7       0       2       9
       3       2       9       0       4       2
       7       0       5       6       1       5

       3       7       9
       7       9       5
```

Subject: Re: Maximum value array resampling
Posted by Richard French on Fri, 05 Aug 2005 20:17:19 GMT

View Forum Message <> Reply to Message

Well, the algorithm I supplied works but is not very fast. Instead of
rebinning up, it would be faster to subsample down and make the comparison
between images of the desired final size. I should have thought of that
before. I'll see if it is faster than the loop method in the end. For a test
case that I just ran, the loop method is faster than the rebinning method.
Sorry! I'll get back to you with revised code and time tests.
Dick

>
>> I'm trying to figure out an IDL-efficient way to resample a series of
>> images. I know how to do this is a ruinously laborious fashion using

```
>>  loops but I know there's any easier way.
>>
>>  Consider the following 4x4 array:
>>
>>  x = [$
>>  [0,3,4,5],$
>>  [1,2,7,0],$
>>  [3,2,9,0],$
>>  [7,0,5,6]]
>>
>>  I want to resample this to y, a 2x2 array. Each element would contain
>>  the maximum value of the corresponding 4 pixels. y would then look like
>>
>>  [$
>>  [3,7],$
>>  [7,9]]
>>
>>  So element [0,0] in y is max(x[0:1,0:1]) etc. As I understand it,
>>  rebin/congrid won't do this. Each image is about 5000x2000 and there
>>  are several hundred to process.
>>
>>  Thanks!
>>
>
>  Try this (I don't know how it scales with image size, but give it a try!)
>
>  The basic idea is to make four rebinned copies of the image  with the upper
>  left, upper right, lower left, and lower right pixels in each 2x2 pair and
>  to replace the array elements by the largest value as you compare each of
>  the rebinned images in turn. Probably you can cut down on memory use by
>  using TEMPORARY() and there may be other increases in efficiency as well.
>  Good luck!
>
>  Dick French
>
>
>  x=[$
>  [0,3,4,5,3,7],$
>  [1,2,7,0,2,9],$
>  [3,2,9,0,4,2],$
>  [7,0,5,6,1,5]]
>
>  Print,x
>
>  ; get size
>
>  size=size(x,/DIM)
>
```

```
>  nx=size[0]
>  ny=size[1]
>
>  nx2=nx/2
>  ny2=ny/2
>
>  ; get indices of upper left element of each 2x2 cell
>
>   l=rebin(nx*2#lindgen(ny2),nx2,ny2)+rebin(lindgen(nx2)#2,nx2, ny2)
>
>
>  ; get indices of ul,ur, ll, lr of each 2x2 cell
>  offsets=[0,1,nx,nx+1]
>
>  for n=0,3 do begin
>          xtemp=rebin(x[l+offsets[n]],nx,ny,/sample)
>          x=x+((xtemp-x)>0)   ; replace x by current largest value
>  endfor
>
>  xfinal=rebin(x,nx2,ny2)
>  print
>  print,xfinal
>
>  End
>
>  IDL> .run trythis
>        0      3      4      5      3      7
>        1      2      7      0      2      9
>        3      2      9      0      4      2
>        7      0      5      6      1      5
>
>        3      7      9
>        7      9      5
>
>
```

Subject: Re: Maximum value array resampling
Posted by b_efremova@yahoo.com on Fri, 05 Aug 2005 20:34:53 GMT
View Forum Message <> Reply to Message

Hi,
I just want to mention, that you have to test it with a large array to
see the difference.
For your example both my loops work at the same speed. But if you take
a large array you can see the difference.
Cheers
Boryana.

Subject: Re: Maximum value array resampling
Posted by Richard French on Fri, 05 Aug 2005 20:45:06 GMT
View Forum Message <> Reply to Message

This works much better! I was being brain dead. Once you know the indices of each of the four entries in each cell, you can do a direct compare. It took 7-8 seconds on my Powerbook G4 for a 4000 x 5000 image, compared to 40-45 seconds using loops:

```
nx=4000L
ny=5000L

; put in random values (this is the slow step!)

X=rebin(fix(1000*(randomu(seed,nx*ny))),nx,ny)
nx2=nx/2L
ny2=ny/2L

; get indices of upper left element of each 2x2 cell

 l=rebin(nx*2#lindgen(ny2),nx2,ny2)+rebin(lindgen(nx2)#2,nx2, ny2)


; compare with  indices of ul,ur, ll, lr of each 2x2 cell

print,'Start....'
T10=systime(1)
xfinal=x[l]>x[l+1]>x[l+nx]>x[l+nx+1]
print,'Time=',systime(1)-t10

t20=systime(1)
y=intarr(nx2,ny2)
  for i=0,nx2-1 do x[i,*]=x[2*i,*] > x[2*i+1,*]
  for i=0,ny2-1 do y[*,i]=x[0:nx2-1,2*i] > x[0:nx2-1,2*i+1]
print,'Time=',systime(1)-t20

print,max(abs(xfinal-y)) ; confirm that we get same result

end
```

~

Subject: Re: Maximum value array resampling
Posted by b_efremova@yahoo.com on Fri, 05 Aug 2005 21:06:59 GMT

View Forum Message <> Reply to Message

Yeah, it's really faster. Even if you count the time for the index
determination.

Subject: Re: Maximum value array resampling
Posted by Richard French on Fri, 05 Aug 2005 21:50:52 GMT
View Forum Message <> Reply to Message

On 8/5/05 5:06 PM, in article
1123276019.641777.66880@g49g2000cwa.googlegroups.com, "b_efremova@yahoo.com"
<b_efremova@yahoo.com> wrote:

> Yeah, it's really faster. Even if you count the time for the index
> determination.
>

That needs to be done only for the first image, and there is probably a
faster way to compute the indices as well (and a bit less obscure!).

Dick

Subject: Re: Maximum value array resampling
Posted by JD Smith on Fri, 05 Aug 2005 23:09:55 GMT
View Forum Message <> Reply to Message

On Fri, 05 Aug 2005 10:55:46 -0700, rechoncho@yahoo.com wrote:

```
> I'm trying to figure out an IDL-efficient way to resample a series of
> images. I know how to do this is a ruinously laborious fashion using loops
> but I know there's any easier way.
>
> Consider the following 4x4 array:
>
> x = [$
> [0,3,4,5],$
> [1,2,7,0],$
> [3,2,9,0],$
> [7,0,5,6]]
>
> I want to resample this to y, a 2x2 array. Each element would contain the
> maximum value of the corresponding 4 pixels. y would then look like
>
> [$
> [3,7],$
> [7,9]]
```

>
> So element [0,0] in y is max(x[0:1,0:1]) etc. As I understand it,
> rebin/congrid won't do this. Each image is about 5000x2000 and there are
> several hundred to process.

For arbitrary images with both dimensions even:

```
d=size(x,/DIMENSIONS) & nx=d[0]/2 & ny=d[1]/2
 y=transpose(max(reform(transpose(reform(x,2,nx,2*ny),[0,2,1] ), $
                    4,ny,nx),DIMENSION=1))
```

How does it work? It juggles dimensions so that the indices of all the
2x2 sub-arrays are next to each other in memory, and then uses
max(/DIMENSION) to collapse over them. The inner call to REFORM puts
them adjacent to each other, but in the wrong dimension, then TRANSPOSE
makes them adjacent in the fast-changing dimension. The rest is
straightforward.

There may be a quicker way with only one call to TRANSPOSE, but I
couldn't find it (anyone?). Also, if you don't care to keep X, throw a
couple of /OVERWRITE keywords for both REFORM statements, to save some
memory and time.

JD

Subject: Re: Maximum value array resampling
Posted by Richard French on Fri, 05 Aug 2005 23:57:55 GMT

View Forum Message <> Reply to Message

```
>
> For arbitrary images with both dimensions even:
>
> d=size(x,/DIMENSIONS) & nx=d[0]/2 & ny=d[1]/2
>  y=transpose(max(reform(transpose(reform(x,2,nx,2*ny),[0,2,1] ), $
>                     4,ny,nx),DIMENSION=1))
>
> How does it work?  It juggles dimensions so that the indices of all the
> 2x2 sub-arrays are next to each other in memory, and then uses
> max(/DIMENSION) to collapse over them.  The inner call to REFORM puts
> them adjacent to each other, but in the wrong dimension, then TRANSPOSE
> makes them adjacent in the fast-changing dimension.  The rest is
> straightforward.
>
> There may be a quicker way with only one call to TRANSPOSE, but I
> couldn't find it (anyone?).  Also, if you don't care to keep X, throw a
> couple of /OVERWRITE keywords for both REFORM statements, to save some
> memory and time.
```

>
> JD
>

On a time test, I found that this clever approach takes 18 sec for a 5000 x
4000 image, compared to the inelegant routine I posted that takes 4 seconds
to compute the indices of each element and then 7 seconds for the resampling
of each image. For lots of images, the speed gain is about 2.5x compared to
this routine. Now we just need to find a way to speed up the clever routine
and we'll be all set! I had not known about the /DIMENSION keyword to the
MAX() function - thanks for the lead on that! Are you sure there isn't a way
to use HISTOGRAM to do this?

Dick

Subject: Re: Maximum value array resampling
Posted by JD Smith on Mon, 08 Aug 2005 17:48:59 GMT
View Forum Message <> Reply to Message

On Fri, 05 Aug 2005 19:57:55 -0400, Richard G. French wrote:

```
>
>>
>>  For arbitrary images with both dimensions even:
>>
>>  d=size(x,/DIMENSIONS) & nx=d[0]/2 & ny=d[1]/2
>>   y=transpose(max(reform(transpose(reform(x,2,nx,2*ny),[0,2,1] ), $
>>                       4,ny,nx),DIMENSION=1))
>>
>>  How does it work?  It juggles dimensions so that the indices of all the
>>  2x2 sub-arrays are next to each other in memory, and then uses
>>  max(/DIMENSION) to collapse over them.  The inner call to REFORM puts
>>  them adjacent to each other, but in the wrong dimension, then TRANSPOSE
>>  makes them adjacent in the fast-changing dimension.  The rest is
>>  straightforward.
>>
>>  There may be a quicker way with only one call to TRANSPOSE, but I
>>  couldn't find it (anyone?).  Also, if you don't care to keep X, throw a
>>  couple of /OVERWRITE keywords for both REFORM statements, to save some
>>  memory and time.
>>
>>  JD
>>
>
>  On a time test, I found that this clever approach takes 18 sec for a 5000 x
>  4000 image, compared to the inelegant routine I posted that takes 4 seconds
>  to compute the indices of each element and then 7 seconds for the resampling
```

> of each image. For lots of images, the speed gain is about 2.5x compared to
> this routine. Now we just need to find a way to speed up the clever routine
> and we'll be all set! I had not known about the /DIMENSION keyword to the
> MAX() function - thanks for the lead on that! Are you sure there isn't a way
> to use HISTOGRAM to do this?

I find just the opposite: your loop+REBIN method is much slower using 5000x4000 long integer arrays:

```
IDL> .run /home/jdsmith/idl/test/max_local.pro
% Compiled module: $MAIN$.
no loop [5000,4000]: 3.8933
French index loop [5000,4000]: 33.1060
modified index loop [5000,4000]: 1.0784
```

I.e. yours is about 8-10x slower on this size image! All of this of course depends on memory (1GB here). I suspect your multiple REBIN'ing of those large images is to blame. That said, I tried with a much smaller image, but the results were similar:

```
IDL> .run /home/jdsmith/idl/test/max_local.pro
% Compiled module: $MAIN$.
no loop [1000,1000]: 0.0975
French index loop [1000,1000]: 0.5524
modified index loop [1000,1000]: 0.0589
```

So, what is the modified index loop which beats both of ours? Your method inspired this significant simplification:

```
d=size(x,/DIMENSIONS) & nx=d[0] & ny=d[1]
nx2=nx/2 & ny2=ny/2

inds=rebin(lindgen(nx2)*2L,nx2,ny2,/SAMPLE)+ $
    rebin(transpose(lindgen(ny2)*2L*nx),nx2,ny2,/SAMPLE)

xmax=x[inds]
offsets=[0L,1L,nx,nx+1L]
for i=1,3 do xmax >= x[inds+offsets[i]]
```

So, at least in this case, the fastest (and definitely the most straightforward to understand) method involves a loop! It can even be extended to nxm (instead of 2x2) local box sizes quite straightforwardly. Heresy, you say? As many have pointed out, if you keep the amount of work per loop iteration large, you don't feel IDL's

loop penalty. Indexing and then comparison operating on images of
size 2500x2000 definitely counts as a "large amount of work". The
other take away point: relative algorithm speed can depend sensitively
on your memory and other local environment. The only sure way to pick
the speed winner is to test various options on your data with your
equipment.

JD

Subject: Re: Maximum value array resampling
Posted by JD Smith on Mon, 08 Aug 2005 18:20:39 GMT
View Forum Message <> Reply to Message

By the way, since for processing many images of the same size you can
pre-compute the indices, you might consider a small change to the
modified loop method:

```
;; Pre-compute the indices:
d=size(x,/DIMENSIONS) & nx=d[0] & ny=d[1]
nx2=nx/2 & ny2=ny/2

inds1=rebin(lindgen(nx2)*2L,nx2,ny2,/SAMPLE)+ $
      rebin(transpose(lindgen(ny2)*2L*nx),nx2,ny2,/SAMPLE)
inds2=inds1+1L
inds3=inds1+nx
inds4=inds1+nx+1L

;; Form the sub-sampled image (for each image)
xmax=x[inds1]>x[inds2]>x[inds3]>x[inds4]
```

This brings the total processing time per 5000x4000 image to under
0.5s on my not-so-fast Linux box (and doesn't have any loops ;).

JD

Subject: Re: Maximum value array resampling
Posted by davis anderton on Mon, 08 Aug 2005 23:04:50 GMT
View Forum Message <> Reply to Message

Many thanks to all! This was my first posting to the group and I'm very
grateful for the responses. I ended up using the
xmax=x[inds1]>x[inds2]>x[inds3]>x[inds4]
method. For my 4943x2104 array resizing on a Dual 2.7 GHz PowerPC G5

with 4.5 GB memory the resize command is taking about 0.15 seconds.

Subject: Re: Maximum value array resampling
Posted by Richard French on Sun, 14 Aug 2005 01:05:54 GMT
View Forum Message <> Reply to Message

On 8/8/05 2:20 PM, in article pan.2005.08.08.18.20.38.917907@as.arizona.edu,
"JD Smith" <jdsmith@as.arizona.edu> wrote:

```
>
> By the way, since for processing many images of the same size you can
> pre-compute the indices, you might consider a small change to the
> modified loop method:
>
>
> ;; Pre-compute the indices:
> d=size(x,/DIMENSIONS) & nx=d[0] & ny=d[1]
> nx2=nx/2 & ny2=ny/2
>
> inds1=rebin(lindgen(nx2)*2L,nx2,ny2,/SAMPLE)+ $
>      rebin(transpose(lindgen(ny2)*2L*nx),nx2,ny2,/SAMPLE)
> inds2=inds1+1L
> inds3=inds1+nx
> inds4=inds1+nx+1L
>
> ;; Form the sub-sampled image (for each image)
> xmax=x[inds1]>x[inds2]>x[inds3]>x[inds4]
>
>
> This brings the total processing time per 5000x4000 image to under
> 0.5s on my not-so-fast Linux box (and doesn't have any loops ;).
>
> JD
>
```

JD also wrote:

I find just the opposite: your loop+REBIN method is much slower using
5000x4000 long integer arrays:

```
IDL> .run /home/jdsmith/idl/test/max_local.pro
% Compiled module: $MAIN$.
no loop [5000,4000]:    3.8933
French index loop [5000,4000]:    33.1060
modified index loop [5000,4000]:    1.0784
```

I.e. yours is about 8-10x slower on this size image! All of this of
course depends on memory (1GB here). I suspect your multiple
REBIN'ing of those large images is to blame. That said, I tried with
a much smaller image, but the results were similar...

(end of quote)

John - Just to clarify things, your 8/8/05 routine above is essentially
identical to the revised routine I posted on 8/5/05 (see below), and I was
referring to this new routine when I said it was faster than what you had
posted. I had also posted a lame rebinning approach to the problem
previously, and I think that is the one you used when you found such poor
performance.
Dick

My previous posting is below:

```
nx=4000L
ny=5000L

; put in random values (this is the slow step!)

X=rebin(fix(1000*(randomu(seed,nx*ny))),nx,ny)
nx2=nx/2L
ny2=ny/2L

; get indices of upper left element of each 2x2 cell

 l=rebin(nx*2#lindgen(ny2),nx2,ny2)+rebin(lindgen(nx2)#2,nx2, ny2)



; compare with  indices of ul,ur, ll, lr of each 2x2 cell

print,'Start....'
T10=systime(1)
xfinal=x[l]>x[l+1]>x[l+nx]>x[l+nx+1]
print,'Time=',systime(1)-t10
```

Subject: Re: Maximum value array resampling
Posted by JD Smith on Mon, 15 Aug 2005 18:13:10 GMT
View Forum Message <> Reply to Message

>

> Just to clarify things, your 8/8/05 routine above is essentially
> identical to the revised routine I posted on 8/5/05 (see below), and I was
> referring to this new routine when I said it was faster than what you had
> posted. I had also posted a lame rebinning approach to the problem
> previously, and I think that is the one you used when you found such poor
> performance.

Sorry about that Dick... looks very similar (except of course for
reform vs #). Here's a version which can do max or min, and any box
size (not just 2x2). Keep in mind that for arrays which are not
multiples of the box size, the edges will be lost. Another
interesting question is how to do a sliding box max/min efficiently,
ala MEDIAN.

JD

```
;; BOX_MAX: Compute local maximum (or minimum. with /MIN) box
;; downsampling, with box size boxx, boxy (default 2,2).  Pre-computed
;; INDS may be passed.
;;   JD Smith (c) 2005.
function box_max,array,boxx,boxy,INDS=inds, MIN=min
  if n_elements(boxx) eq 0 then boxx=2
  if n_elements(boxy) eq 0 then boxy=2
  min=keyword_set(min)
  d=size(array,/DIMENSIONS)
  nx=d[0] & ny=d[1]

  if n_elements(inds) eq 0 then begin
     nx_out=nx/boxx & ny_out=ny/boxy
     inds=rebin(lindgen(nx_out)*boxx,nx_out,ny_out,/SAMPLE)+ $
          rebin(transpose(lindgen(ny_out)*boxy*nx),nx_out,ny_out,/SAMP LE)
  endif

  ret=array[inds]
  for i=0L,boxx-1L do begin
     for j=0L,boxy-1L do begin
        if i eq 0 && j eq 0 then continue
        if min then ret <= array[inds+i+j*nx] else ret >= array[inds+i+j*nx]
     endfor
  endfor

  return,ret
end
```