
Subject: Re: Finding the median of a set of images
Posted by [Dyer Lytle](#) on Wed, 24 Apr 1996 07:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

Last week I wrote:

>
> Does anyone have an algorithm for finding the median at each pixel
> position for a set of equal size 2-D images? Currently the only way
> I have to do this is to extract all the values for a given pixel
> position into a 1-D array and find the median on that. Doing it
> pixel by pixel like this is inefficient in IDL so I am looking for an
> *array* based algorithm that would find all the medians in parallel.

Well, I wrote an array based median routine based on an algorithm from "Numerical Recipes, The Art of Scientific Computing" by Press, Flannery, Teukolsky, and Vetterling on pages 460-462. This is an iterative median finder and I added sigma pixel rejection.

Unfortunately, it is much slower than doing it pixel by pixel! There are two reasons for this, one is that I have to do a *lot* of array comparisons and the other is that some pixels take *many* iterations to converge. On average, convergence should only take log N passes through the data, but when you have, say, 65,536 medians to be calculated, (256 x 256 image), there are always a few that take much longer than log N passes to converge.

I'll include the function below, if anyone has any tricks to try to speed up the code significantly, I'd love to hear about them.

It was interesting.....

Cheers,

-Dyer

--

Dyer Lytle
dlytle@as.arizona.edu
HST NICMOS Project
Steward Observatory
University of Arizona

;+

```

; Name:
;   immedian
;
; Purpose:
;   This function finds the median at each pixel position for an array
;   of images. Allow iterative pixel rejection if nsig is specified.
;
; Category:
;   images, statistics
;
; Calling Sequence:
;   medimage = immedian(in, nsig, minpix)
;
; Inputs:
;   in:   A 3D array from which to form a median image along the 3rd axis.
;   nsig: The number of sigmas to use for iterative pixel rejection
;   minpix: The minimum number of pixels to allow for finding a median
;
; Outputs:
;   medimage: A 2D array containing the median data.
;
; Modification History:
;   18, April, 1996, Written by Dyer Lytle, HST NICMOS project
;   (algorithm from "Numerical Recipes, The Art of
;   Scientific Computing" by Press, Flannery,
;   Teukolsky, and Vettering, pp 460-462)
;-
;
;on_error, 2      ; Return to caller if an error occurs

```

```
function immedian, in, nsig, minpix
```

```

; Find the size of the arrays.
tmp = size(in)
dims = tmp(0)
dim1 = tmp(1)
dim2 = tmp(2)
dim3 = tmp(3)

; Create mask array to mark rejected values.
rmask = bytarr(dim1,dim2,dim3)

; Return w/ error message if input array incorrect.
if (dims ne 3 or dim3 lt 3) then begin
  print, 'immedian: Input array must be 3D'
  print, 'and the third dimension must be larger than 2.'
  return, 1

```

```

endif

; Define a few constants.
big = 1.0e30
afac = 1.05
amp = 1.05

; Initialization and allocation
a = 0.5 * (in(*,*,0) + in(*,*,dim3-1)) ; first guess for the median
eps = abs(in(*,*,dim3-1)-in(*,*,0)) ; first guess at point spacing
ap = fltarr(dim1,dim2) ; upper bound on the median
am = fltarr(dim1,dim2) ; lower bound on the median
sum = fltarr(dim1,dim2)
sumx = fltarr(dim1,dim2)

np = intarr(dim1,dim2) ; number of points above the current guess
nm = intarr(dim1,dim2) ; number of points below the current guess
xp = fltarr(dim1,dim2) ; value of the point above and closest to the guess
xm = fltarr(dim1,dim2) ; value of the point below and closest to the guess

nextit: ; next sigma iteration, initialize boundarys
ap(*,*) = big
am(*,*) = -big

one: ; next median iteration, initialize various arrays
sum(*,*) = 0.0
sumx(*,*) = 0.0
np(*,*) = 0
nm(*,*) = 0
xp(*,*) = big
xm(*,*) = -big

; For all the images in the 3rd dimension.
for j=1,dim3 do begin
xx=in(*,*,j-1)

np = np + 1 * (xx gt a) * (rmask(*,*,j-1) ne 1)
mask1 = xx lt xp and xx gt a
mask2 = xx ge xp or xx lt a or rmask(*,*,j-1) eq 1
xp = xp * mask2 + xx * mask1
nm = nm + 1 * (xx lt a)
mask1 = xx gt xm and xx lt a
mask2 = xx le xm or xx gt a or rmask(*,*,j-1) eq 1
xm = xm * mask2 + xx * mask1

; Prepare for the division below for calculating 'dum'.
tmpdum = eps+abs(xx-a)
tmpwhere = where(xx eq a,cnt)

```

```

if (cnt gt 0) then tmpdum(tmpwhere) = 10.0 ; random value to avoid div by 0
dum = 1./tmpdum

sum = sum + dum * (xx ne a) * (rmask(*,*,j-1) ne 1)
sumx = sumx + xx * dum * (xx ne a) * (rmask(*,*,j-1) ne 1)

endfor

; Check to see if we are done finding median.
tmp = where((abs(nm-np) gt 2),count)
where_nc = (abs(nm-np) gt 2)
if (count eq 0) then begin      ; got the median! (for all pixels)
    ; For even N the median is always an average.
    if ((dim3 mod 2) eq 0) then begin
        xmed = 0.5*(xp+xm) * (np eq nm) + $
        0.5*(a+xp) * (np gt nm) + $
        0.5*(xm+a) * (np lt nm)
    ; For odd N median is always one point.
    endif else begin
        xmed = a * (np eq nm) + $
        xp * (np gt nm) + $
        xm * (np lt nm)
    endelse

; If nsig is zero then don't do sigma rejection, just return.
; Otherwise, if nsig is greater than zero, goto sigma rejection.
if (nsig gt 0) then begin
    goto, nextsig
endif else begin
    goto,fin
endelse
endif

; If we got here, median for some pixels not done yet,
; recalculate and reiterate. (Guess is too low or too high.)

mask1 = (np-nm) ge 2
mask2 = (np-nm) lt 2
mask3 = (nm-np) ge 2
mask4 = (nm-np) lt 2

; New best guess.
aa = (xp+((sumx/sum-a*((sumx/sum-a) gt 0)))*amp)*mask1 + $ ; guess was low
    (xm+((sumx/sum-a*((sumx/sum-a) lt 0)))*amp)*mask3 + $ ; guess was high
    a * (abs(nm-np) lt 2) ; don't really need this last term...

; Don't let it exceed boundaries.
aa = 0.5*(a+ap)*(aa gt ap) + 0.5*(a+am)*(aa lt am) + $

```

```

aa * (aa lt ap and aa gt am)

; Calculate new boundaries.
am = am * mask2 + a * mask1
ap = ap * mask4 + a * mask3

; And a new smoothing factor.
eps = afac*abs(aa-a) * (where_nc eq 1) + eps * (where_nc eq 0)
a = aa * (where_nc eq 1) + a * (where_nc eq 0)

; Iterate median once again.
goto, one

nextsig: ; next sigma iteration

; Calculate standard deviations.
; IDL's MOMENT routine only works on 1-D arrays so write our own...

mean = total(in,3)/dim3 ; calculate all the means
mean3 = rebin(mean,dim1,dim2,dim3) ; rebin to 3D for variance calculation

; Calculate variance and standard deviation. (same var equation as IDL moment)
var = (1./(dim3-1))*(total((in-mean3)^2,3)-(1./dim3)*total((in-mean3)^2,3))
stdv = sqrt(var)

; 2D array containing number of 'good' values for each pixel before rejection.
oldmaskcount = dim3-total(rmask,3)

; Reject points. Points that are more than nsig standard deviations
; from the median have their mask values set to one.
tmp = where(in gt rebin((xmed+nsig*stdv),dim1,dim2,dim3) or $
in lt rebin((xmed-nsig*stdv),dim1,dim2,dim3),cnt)
if (cnt gt 0) then rmask(tmp) = 1 ; tmp contains list of places where true

; 2D array containing number of 'good' values for each pixel after rejection.
newmaskcount = dim3-total(rmask,3)

; How many places still have more than minpix pixels and
; had a point rejected?
tmp = where(newmaskcount gt minpix and (oldmaskcount - newmaskcount) gt 0,ct)

; If no points rejected or numpts le minpix for all array points, done.
; Else, iterate once again.
if (ct eq 0) then begin
    goto, fin
endif else begin
    goto, nextit
endelse

```

fin:

```
; Attempt to recover memory by setting arrays to scalars. (does this work?)
```

```
rmask = 0
```

```
np = 0
```

```
nm = 0
```

```
xp = 0
```

```
xm = 0
```

```
a = 0
```

```
aa = 0
```

```
an = 0
```

```
ap = 0
```

```
eps = 0
```

```
mask1 = 0
```

```
mask2 = 0
```

```
mask3 = 0
```

```
mask4 = 0
```

```
mean = 0
```

```
mean3 = 0
```

```
var = 0
```

```
stdv = 0
```

```
oldmaskcount = 0
```

```
newmaskcount = 0
```

```
; Successful return.
```

```
return, xmed
```

end

File Attachments

1) [immedian.pro](#), downloaded 106 times
