Subject: STANDARD DEVIATON
Posted by Ben Marriage on Tue, 01 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Hi all,

I want to calculate the standard deviation of a 3x3 pixel area for each element of a satellite image (5000x2000 pixels). At the moment I use for loops to scan through the image and then use the built-in IDL standard deviation routines. I've tried to speed this up with the use of convolution and whatnot but never seemed to solve the problem. Does anyone know of a simple and elegant (read "quick") method of doing this without using loops?

Thanks,
Ben Marriage
Department of Meteorology, University of Edinburgh

Subject: Re: STANDARD DEVIATON
Posted by Struan Gray on Fri, 04 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I think this works:

```
; ************************************

function smg_imageSD, image

  fimage = float(image)
  localmean = smooth(fimage, 3)
  sum = (fimage - localmean)^2
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1, 1))^2,-1,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 0, 1))^2, 0,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1, 1))^2, 1,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1, 0))^2,-1, 0)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1, 0))^2, 1, 0)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1,-1))^2,-1, 1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 0,-1))^2, 0, 1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1,-1))^2, 1, 1)
```

```
  sum = sqrt(temporary(sum)/8)

  dims = size(sum, /dim)
  sum[0,*] = 0.0
  sum[*,0] = 0.0
  sum[dims(0)-1,*] = 0.0
  sum[*,dims(1)-1] = 0.0

  return, sum
end

; ***********************************
```

You can generalise the shifting and put it into a double loop over the kernal indices. You can also deal with edge-effects (and avoid the zeroing of edge elements) if you creat an oversize image and pad it appropriately. On my machine this is approx ten times faster than Ben_imageSD.

Struan

Subject: Re: STANDARD DEVIATON
Posted by Struan Gray on Fri, 04 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I wrote:

> It's possible I'm missing something

and then, about four seconds after I hit the submit button, realised what it was I was missing: for a corner pixel of any given 9x9 area my method subtracts the mean of a different 9x9 area centred on the corner pixel.

Still, my method is about twenty times faster than Ben_imageSD which in turn is about four times faster than IDL_imageSD, so there's a lot of incentive to find a hack.

The traditional way to speed up the loops would be to buffer the calculation of the running average. For example, instead of adding elements i-1, i, and i+1 of each row you remember the sum from last time, subtract i-2 and add i+1. With a 9-element kernal this won't help a lot, but it will a bit.

If you're not too worried about edge effects (and your posted code wasn't), you could always create nine copies of the (float(image) -

localmean) array from my function, each with the localmean array shifted by different amounts (use the SHIFT function to avoid recalculation). Then you'd just have to shift them back to line up the pixels, square them and take the sum. This could be done as a loop over the 9x9 kernal elements if you don't have the memory or disk space to hold all the copies in memory at once.

Struan

Subject: Re: STANDARD DEVIATON
Posted by Struan Gray on Fri, 04 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Ben Marriage, ben@met.ed.ac.uk writes:

> I believe that this gives different results to
> the inbuilt IDL routines.

It's possible I'm missing something, but the result of my function should be the same as yours, but multiplied by sqrt(8/9). You and IDL are dividing by (N-1) in the definition of variance, I am dividing by N. Statisticians love to argue about which is appropriate and where, but for an image of the s.d. it doesn't matter much. The box smooth always divides by N, but you can always add a renormalisation step and it'll still be faster than nested loops.

I use this as part of a routine to do so-called statistical differencing of images, which brings out fine detail on a varying background (it's like an unsharp mask weighted by the local image statistics). In this case the the s.d. image is multiplied by an arbitrary, user-selectable factor, so the difference between N and N-1 is irrelevant.

Struan

Subject: Re: STANDARD DEVIATON
Posted by Ben Marriage on Fri, 04 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:
root.
>
>

```
>      function imageSD, image
>
>        localmean = smooth(float(image), 3, /edge_truncate)
>        localsd = (float(image)-temporary(localmean))^^2
>        localsd = smooth(temporary(localsd), 3, /edge_truncate)
>        localsd = sqrt(temporary(localsd))
>
>        return, localsd
>
>      end
>
>
>  Struan
```

I believe that this gives different results to the inbuilt IDL routines.
The problem is a wee bit more complicated than it looks.

My routine:

```
 ;============================================================ ==
function ben_imageSD, image

siz = size(image,/dim)

standdev = fltarr(siz[0],siz[1])

for i=1,siz[0]-2 do begin
   for j=1,siz[1]-2 do begin
      standdev[i,j] = sqrt(total((image[i-1:i+1,j-1:j+1]-$

total(image[i-1:i+1,j-1:j+1])/9.)^2)/8.)
   endfor          ;tricky bit ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
endfor             ;to do without loops
return,standdev
end
 ;============================================================ ==
```

IDL's version:

```
 ;============================================================ ==
function IDL_imageSD, image

siz = size(image,/dim)

standdev = fltarr(siz[0],siz[1])

for i=1,siz[0]-2 do begin
   for j=1,siz[1]-2 do begin
```

```
      standdev[i,j]=stddev(image[i-1:i+1,j-1:j+1])
   endfor
endfor
return,standdev
end
 ;============================================================ ==
```

They do similar things but with loops. Can't figure out how to do this
without loops!

Thanks,
Ben

Subject: Re: STANDARD DEVIATON
Posted by Ben Marriage on Fri, 04 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Benno Puetz wrote:
>
> Alex Schuster wrote:
>
>> For some 3x3 pixels x_i (and a mean of x_mean), the standard deviation
>> is
>> s = 1/9 * sqrt(sum((x_i-x_mean)^2)). img=img-smooth(img,3) does the
>> inner subtraction of the x_means,
>
> unfortunately, that's not quite true (except for special cases) since the x_i
> get only their respective mean subtracted, not the center pixel mean. I came
> up with the same solution initially but it would not stand the test ...
>
> haven't had a better idea since :(
>

I think that's what I discovered too - hence the question to the NG.

I'll try and have another think about it.

Thanks,
Ben

Subject: Re: STANDARD DEVIATON
Posted by Ben Marriage on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

In article <8mei3g$6dl$1@news.lth.se>,

Struan Gray <struan.gray@sljus.lu.se> wrote:

> You can generalise the shifting and put it into a double loop over the kernal
> indices. You can also deal with edge-effects (and avoid the zeroing of edge
> elements) if you creat an oversize image and pad it appropriately. On my
> machine this is approx ten times faster than Ben_imageSD.
>

Yes, this seems to work quite nicely. Much better than loops. It shouldn't balloon in size as you are using the temporary function too.

Thanks for the solution,
Ben