Subject: Re: Gradient idl
Posted by siumtesfai on Tue, 10 Mar 2015 16:39:03 GMT
View Forum Message <> Reply to Message

On Tuesday, March 10, 2015 at 12:13:31 PM UTC-4, Jeremy Bailin wrote:
> On Tuesday, March 10, 2015 at 11:00:30 AM UTC-5, siumt...@gmail.com wrote:
>> On Tuesday, March 10, 2015 at 1:12:31 AM UTC-4, siumt...@gmail.com wrote:
>>> Hello all,
>>>
>>> How can calculate gradient a 2-Dimensional scalar field with missing data.
>>>
>>>
>>>
>>> Best regards
>>
>> do you mean to calculate gradient of a scalar function (2ddata) can be solved this way
>>
>> kernel = [[-1,0,1],[-1,0,1]]
>> result = CONVOL(2Ddata, kernel,/NAN)
>
> No, you'll need two separate convolutions, one for the x-component of the gradient and one for the y-component:
>
> result_x = CONVOL(Data, [-1,0,1], /NAN)
> result_y = CONVOL(Data, TRANSPOSE([-1,0,1]), /NAN)
>
> See the CONVOL help page for details about edge options, etc.
>
> -Jeremy.

Hello

I am interested in the magnitude of the gradient

Result_xy=sqrt(result_x^2 +result_y^2)
right ?

I looked at the following function but it does not handle missing data

```
function gradient, image, vector=vector, norm=norm,  $
     forward=forward,central=central,$
     scale=scale
;+
; NAME:
; gradient
; PURPOSE:
; Compute the gradient vector at every pixel of an image using
; neighbor pixels. Default is to return the Euclidean norm of gradient
```

```
; (2-D array), or specify keywords for other options.
; CALLING:
; gradim = gradient( image )
; INPUTS:
; image = 2-D array.
; KEYWORDS:
; scale=scale,or [xscale,yscale]; scale means [scale,scale] for a single value,
scale=[xscale,yscale]
; /vector then 2 images are returned (3-D array) with d/dx and d/dy.
; /forward then x & y-partial derivatives computed as forward difference.
;       /central, or default cenral difference
; /norm then magnitude of gradient is computed as norm.
; OUTPUTS:
; Function returns gradient norm (2-D array) or gradient vector (3-D).
; HISTORY:
;                 10-Feb-2011, by Ding Yuan( Ding.Yuan@warwick.ac.uk)
;
;-
if ~exist(image) then on_error
sz = size( image )
if (sz[0] ne 2) then message,"Please input an 2D image (array)"
nx=sz[1] & ny=sz[2]
option=0
if exist(forward) then option=1
case option of
    0: begin
    didx = ( shift( image, -1,  0 ) - shift( image, 1, 0 ) ) * 0.5
    didx[0,*] = image(1,*) - image(0,*)
    didy = ( shift( image,  0, -1 ) - shift( image, 0, 1 ) ) * 0.5
    didy[*,0] = image(*,1) - image(*,0)
  end
    1:  begin
          didx = shift( image, -1,  0 ) - image
          didy = shift( image,  0, -1 ) - image
  end
else: on_error,2
endcase
  didx[nx-1,*] = image[nx-1,*] - image[nx-2,*]
  didy[*,ny-1] = image[*,ny-1] - image[*,ny-2]
xscale=1 & yscale=1
case n_elements(scale) of
     0: begin
           xscale=1 & yscale=1
   end
     1:begin
           xscale=scale & yscale=scale
   end
      2:begin
```

```
            xscale=scale[0] & yscale=scale[2]
    end
      else: message,'scale should be a scalar or 2 element vector'
endcase
didx = didx * xscale
didy = didy * yscale
if keyword_set(vector ) then  return, [ [[didx]], [[didy]] ] else return , sqrt( didx*didx + didy*didy )
end
```