Subject: Re: Local Maxima of 2D array
Posted by Yngvar Larsen on Mon, 25 Jan 2010 15:55:22 GMT
View Forum Message <> Reply to Message

On Jan 19, 6:48 pm, Robin Wilson <r.t.wil...@rmplc.co.uk> wrote:
> Hi,
>
> Another question from me I'm afraid. I'm trying to implement a routine
> which needs to be able to calculate the local maxima of a small window
> moved across an array. That is, I have a large array and I will need to
> move a small 3x3 array across it, each time working out what the maximum
> value of that array is and storing its index (or selecting it in some
> other way).
>
> I've investigated various methods for doing this, including the dilate
> method, but I can't seem to get them to work properly.
>
> Is there any good (as in fast, efficient and elegant) way of doing this,
> or will I be reduced to using for loops and lots of IF statements?

Some kind of FOR loop is unavoidable, I think.

Depending of the size of your array, this code will do (most of) the
job efficiently. Elegant? Well...

```
x = [-1,  0,  1, -1, 0, 1, -1, 0, 1]
y = [-1, -1, -1,  0, 0, 0,  1, 1, 1]

; Or in general for a sliding (Kx x Ky) window
;x = lindgen(Kx)-Kx/2
;y = lindgen(Ky)-Ky/2
;x = (x[*,lindgen(Ky)])[*]
;y = (transpose(y[*,lindgen(Kx)]))[*]

sliding_3x3_max = shift(array, x[0], y[0])
for ii=1, 8 do sliding_3x3_max >= shift(array, x[ii], y[ii])
```

Note that the border case isn't handled. This is left as an exercise
for the reader :) Also, if you really need the index for each maximum
instead of the value, you must do a bit more work inside the loop.

My experience is that this method works well for operations on sliding
windows up to about 15x15, but for larger windows, the cost of the
(quite fast) SHIFT function starts to dominate when compared to the
straightforward double loop approach.

--
Yngvar