Subject: Re: Nearest Neighbor ... again!
Posted by David Fanning on Thu, 08 Jul 2010 13:21:45 GMT
View Forum Message <> Reply to Message

Fabzi writes:

> You probably have been asked many times, but once again this
> apparently simple problem is driving me crazy. The problem is famous:
>
> I have a 2D grid defined by x1 (dim 2 array, for example lons) and y1
> (dim2 array, for example lats). And I want to fit it to a second grid
> x2, y2. More precisely, I want to know the indexes in GRID1 that are
> the closest to each of my points in GRID2. The output of my function
> has then the same dimension as GRID2.

Unless I completely miss the point, this seems to me
to be a simple nearest neighbor interpolation from one
grid to another. CONGRID has been used for this purpose
for years. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Nearest Neighbor ... again!
Posted by Fabzi on Thu, 08 Jul 2010 13:40:19 GMT
View Forum Message <> Reply to Message

I am sorry, I did not make clear that grid1 and grid2 are two
different irregular grids (lat1,lon1),(lat2,lon2). It is indeed
classical, I know, and I feel quite embarrassed!

So the code here:

```
  n1 = n_elements(ilon)
  triangulate, x1, y1, c ; Compute Delaunay triangulation
  out = GRIDDATA(x1,y1, LINDGEN(n1), XOUT=x2, YOUT=y2, /NEAREST_N,
TRIANGLES = c)
```

Is giving me, for each point in Grid2, the index of the closest point
in grid1. My question is, how to get the four closest points? The "bad
trick" that I used with GRIDDATA won't work here...

On Jul 8, 3:21 pm, David Fanning <n...@dfanning.com> wrote:
> Fabzi writes:
>> You probably have been asked many times, but once again this
>> apparently simple problem is driving me crazy. The problem is famous:
>
>> I have a 2D grid defined by x1 (dim 2 array, for example lons) and y1
>> (dim2 array, for example lats). And I want to fit it to a second grid
>> x2, y2. More precisely, I want to know the indexes in GRID1 that are
>> the closest to each of my points in GRID2. The output of my function
>> has then the same dimension as GRID2.
>
> Unless I completely miss the point, this seems to me
> to be a simple nearest neighbor interpolation from one
> grid to another. CONGRID has been used for this purpose
> for years. :-(
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Nearest Neighbor ... again!
Posted by jeanh on Thu, 08 Jul 2010 16:06:51 GMT
View Forum Message <> Reply to Message

> The stupid solution is (sorry for the very very ugly code):
>
> ---
> for i = 0l, n2 - 1 do begin
> quad = (x2[i] - x1)^2 + (y2[i]- y1)^2
> for j=0, 3 do begin
> minquad = min(quad, p)
> if N_ELEMENTS(p) gt 1 then p = p[0] ; it happens.....

> out[j,i] = p
> quad[p] = max(quad) * 2. ;dummy large distance
> endfor
> endfor
> ---
>
> But I just cannot find a cleverer solution with triangulation...
> Someone clever than me to help ?
>
> Thanks a lot!
>
> Fabz

While there is certainly a better solution, you can start by removing the j loop, and use sort(quad) instead of min() ... you can then take the first 4 index.
Jean

Subject: Re: Nearest Neighbor ... again!
Posted by penteado on Thu, 08 Jul 2010 16:27:31 GMT
View Forum Message <> Reply to Message

On Jul 8, 10:40 am, Fabzi <fabien.mauss...@gmail.com> wrote:
> I am sorry, I did not make clear that grid1 and grid2 are two
> different irregular grids (lat1,lon1),(lat2,lon2). It is indeed
> classical, I know, and I feel quite embarrassed!
>
> So the code here:
>
> n1 = n_elements(ilon)
> triangulate, x1, y1, c ; Compute Delaunay triangulation
> out = GRIDDATA(x1,y1, LINDGEN(n1), XOUT=x2, YOUT=y2, /NEAREST_N,
> TRIANGLES = c)
>
> Is giving me, for each point in Grid2, the index of the closest point
> in grid1. My question is, how to get the four closest points? The "bad
> trick" that I used with GRIDDATA won't work here...

If you need to interpolate from one irregular rectangular grid to the other, you can use bilinear() or interpolate(). There are several other options if the output grid is regular. If you really need the nearest points, voronoi() might serve.

Subject: Re: Nearest Neighbor ... again!

On Thu, 8 Jul 2010 06:11:37 -0700 (PDT), Fabzi
<fabien.maussion@gmail.com> wrote:

> After using a long time the (very) inefficient "for loop":
>
> -----
> n2 = N_ELEMENTS(x2)
> for i = 0l, n2 - 1 do begin
> quad = (x2[i] - x1)^2 + (y2[i]- y1)^2
> minquad = min(quad, p)
> if N_ELEMENTS(p) gt 1 then p = p[0] ; it happens.....
> out[i] = p
> endfor
> ---
>
> I finaly found the best solution:
>
> ---
> n1 = n_elements(ilon)
> triangulate, x1, y1, c ; Compute Delaunay triangulation
> out = GRIDDATA(x1,y1, LINDGEN(n1), XOUT=x2, YOUT=y2, /NEAREST_N,
> TRIANGLES =c)
> ---

These two methods give a different result. Try running the code below.
The red and green lines connect the nearest neighbours from method 1
and method 2. You should see only red lines, but you see some green
lines too...

This doesn't answer your question about finding the 4 closest
neighbours however. Couldn't you use griddata with bilinear
interpolation and get the "4 corners" from the interpolated value?
I'll think about it :-).

The real question is, what are you going to do when you have the 4
nearest neighbours? What are you trying to achieve that can't be done
with IDL's gridding and interpolation routines?

pro ConnectGrids

```
; First grid
n1 = 100
seed = -121147L
x1 = round(RANDOMU(seed, n1)*n1)
y1 = round(RANDOMU(seed, n1)*n1)

; Second grid
n2 = 100
x2 = round(RANDOMU(seed, n2)*n2)
y2 = round(RANDOMU(seed, n2)*n2)

; Find closest: method 1
triangulate, x1, y1, c ; Compute Delaunay triangulation
iconnect = GRIDDATA(x1,y1, LINDGEN(n1), Xout=x2, Yout=y2,$
 /NEAREST_N,TRIANGLES =c)

; Find closest: method 2
iconnect2=iconnect
for i=0l,n2-1 do begin
 tmp=min((x2[i]-x1)^2.+(y2[i]-y1)^2.,ind)
 iconnect2[i]=ind[0]
endfor

; Plot result
device,decompose=0
loadct,39
window
n=n1>n2
plot,x1,y1,psym=1,xrange=[-10,n+9],yrange=[-10,n+9],/xs,/ys
oplot,x2,y2,psym=1,color=100
for i=0,n2-1 do
 plots,[x1[iconnect[i]],x2[i]],[y1[iconnect[i]],y2[i]],color= 150
for i=0,n2-1 do plots,$
 [x1[iconnect2[i]],x2[i]],[y1[iconnect2[i]],y2[i]],color=250

end
```

Subject: Re: Nearest Neighbor ... again!
Posted by Fabzi on Mon, 12 Jul 2010 08:14:09 GMT
View Forum Message <> Reply to Message

> These two methods give a different result. Try running the code below.
> The red and green lines connect the nearest neighbours from method 1
> and method 2. You should see only red lines, but you see some green
> lines too...

Yes, the first method is actually the fastest (especially when working on
huge datasets).

>
> This doesn't answer your question about finding the 4 closest
> neighbours however. Couldn't you use griddata with bilinear
> interpolation and get the "4 corners" from the interpolated value?
> I'll think about it :-).

Thanks!

>
> The real question is, what are you going to do when you have the 4
> nearest neighbours? What are you trying to achieve that can't be done
> with IDL's gridding and interpolation routines?

Yes, I probably want to do something that IDL might solve better than me.
I want to fit a dataset with quality assessment (MODIS) on an other grid,
and test all four neirest points to take the best quality value. This concerns
only few grid points, as most of time the points are either "all good" or
"all bad".

It was also interesting in itself, to be able to have the four neirest
in hand !

I see that my request has something "unusual", and will probably not be solved
without long computing times. I will try to address my problem differently.

Thanks again, I learned a lot already just trying to do it.

>
> pro ConnectGrids
>
> ; First grid
> n1 = 100
> seed = -121147L
> x1 = round(RANDOMU(seed, n1)*n1)
> y1 = round(RANDOMU(seed, n1)*n1)
>
> ; Second grid
> n2 = 100

```
>  x2 = round(RANDOMU(seed, n2)*n2)
>  y2 = round(RANDOMU(seed, n2)*n2)
>
>  ; Find closest: method 1
>  triangulate, x1, y1, c ; Compute Delaunay triangulation
>  iconnect = GRIDDATA(x1,y1, LINDGEN(n1), Xout=x2, Yout=y2,$
>         /NEAREST_N,TRIANGLES =c)
>
>  ; Find closest: method 2
>  iconnect2=iconnect
>  for i=0l,n2-1 do begin
>         tmp=min((x2[i]-x1)^2.+(y2[i]-y1)^2.,ind)
>         iconnect2[i]=ind[0]
>  endfor
>
>  ; Plot result
>  device,decompose=0
>  loadct,39
>  window
>  n=n1>n2
>  plot,x1,y1,psym=1,xrange=[-10,n+9],yrange=[-10,n+9],/xs,/ys
>  oplot,x2,y2,psym=1,color=100
>  for i=0,n2-1 do
>   plots,[x1[iconnect[i]],x2[i]],[y1[iconnect[i]],y2[i]],color= 150
>  for i=0,n2-1 do plots,$
>         [x1[iconnect2[i]],x2[i]],[y1[iconnect2[i]],y2[i]],color=250
>
>  end
```