Subject: weird behavior of Triangulate
Posted by envi35@yahoo.ca on Sat, 01 Sep 2012 04:08:55 GMT
View Forum Message <> Reply to Message

Hi, I'm trying to use GRIDDATA to convert two sets of regular grids (lat/lon) to a projected grid (Equal Area). The first set of lat/lon has dimensions of 180 and 720, and spacing of 0.5 deg, with values of the following:
lat1: 0.25,0.75,1.25, ...89.75;
lon1: -179.750, -179.250, -178.750, ...179.75

The second set of lat/lon has dimensions of 120 and 480, and spacing of 0.75 deg, with values of:
lat2: 0.75, 1.50, 2.25,...90.0
lon2: -180, -179.25, -178.5, -177.75, ...179.25

The first set works fine, however, I got co-linear error for the second set. which is weird, as the two sets of lat/lon look similar to me. (I did try to remove lat=90.0 in the secod set but with no luck!). Does anybody know why? Here is part of my code:

```
ysize = Size(lat, /DIMENSION)
xsize = Size(lon, /DIMENSION)
print,xsize,ysize
lats = Rebin(Reform(lat, 1, ysize), xsize, ysize)
lons = Rebin(lon, xsize, ysize)

 mapStruct = Map_Proj_Init(111,semimajor_axis=6371228,$
       semiminor_axis=6371228,CENTER_LONGITUDE=0.0,
center_latitude=90.0)

xy = Map_Proj_Forward(lons, lats, MAP_STRUCTURE=mapStruct)
x = Reform(xy[0,*], xsize, ysize)
y = Reform(xy[1,*], xsize, ysize)

Triangulate, x ,y, triangles, TOLERANCE=1.0
```

Thanks,
Jenny

Subject: Re: weird behavior of Triangulate
Posted by Klemen on Thu, 06 Sep 2012 10:13:17 GMT
View Forum Message <> Reply to Message

Hi Jenny,

I would suggest removing the line in your data having 90 degrees latitude. I think it should work

then.

I am using griddata often and it works well if your data-set is not too large. If yes, you can always process it by overlapping tiles. The only real disadvantage is gridding to spherical coordinates - griding to Cartesian coordinates is much faster.

Klemen

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Thu, 06 Sep 2012 14:31:01 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

>
> On Saturday, 1 September 2012 16:45:46 UTC+2, David Fanning wrote:
>
>> My conclusion is that if you need things regridded (and
>> if you work with satellite images, this is *always*
>> required, eventually), you will have to use something other
>> than IDL to do the job.
>
> Could you elaborate a bit on that conclusion?
>
> I work in an institute where IDL has been used as the main tool for analysis of remote sensing data for nearly 3 decades, so this statement puzzles me...
>
> I can only think of the following issues:
>
> * IDL didn't have support for map projections until version 5.6 or so. Before that, we wrapped the PROJ.4 library to do the job. But the MAP_PROJ_* functionality has now been available for almost a decade (with some inherent problems that are possible to work around).
>
> * Satellite data are large, so GRIDDATA might not work too well if you operate directly on the entire data set. Solution: divide-and-conquer. Divide your output grid in blocks, and process separately. Should normally be possible to make an efficient solution based on GRIDDATA and/or INTERPOLATE.
>
> But I have a feeling you have something else in mind?

Typically, when you work with satellite data you have some
notion of a "study area". For example, in my lab we are
doing some research on the big High Park fire that burned
a large portion of the forest near here this summer.
Typically, the study area is a rectangular region or grid
that you place on top of the area of the Earth you wish to
study.

Then, you look for data that covers the study area. Many times the data only covers a portion of the study area. The data you do find is often in different resolutions (LandSat, 30m, Quickbird 5m, etc.) To make sense of this data, it needs to be gridded according to the resolution of the study area grid. This is a difficult problem, and there are various ways that the data can be gridded (nearest neighbor, natural neighbor, weighted gridding, etc.).

At NSIDC, we used a C program named mapx to do this gridding for us. Given a study grid resolution and map projection, data from various sensors could be "gridded" (and mosaicked, if needed) in various ways. This is the part I don't know how to do in IDL.

It seems as if GridData would be useful. But, unless you are working with 50x50 pixel blocks of data, GridData is ungodly slow! Interpolate is probably better, but you don't have many gridding options (linear, bilinear, cubic).

Once you get the data into your rectangular study grid, you generally set up a map coordinate system that describes it. As you can see this morning, this is either impossible to do in the new IDL function graphics system, or so opaque as to be invisible.

In any case, I'd be interested to know how you solve these problems. :-)

Cheers,

David

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

Subject: Re: weird behavior of Triangulate
Posted by DavidF[1] on Thu, 06 Sep 2012 16:05:39 GMT
View Forum Message <> Reply to Message

I wrote earlier:

> It seems as if GridData would be useful. But, unless you are
> working with 50x50 pixel blocks of data, GridData is ungodly

> slow! Interpolate is probably better, but you don't have
> many gridding options (linear, bilinear, cubic).

By the way, I've demonstrated that GridData *does* do the right thing (sorta) *if* the data set resolution AND the output grid resolution is small enough. (And, maybe, if you run it on the right machine.) But, I need a more robust solution that works with the kinds of satellite data I use daily.

http://www.idlcoyote.com/code_tips/usegriddata.html

Cheers,

David

Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Mon, 10 Sep 2012 09:00:38 GMT
View Forum Message <> Reply to Message

On Thursday, 6 September 2012 18:05:39 UTC+2, Coyote wrote:
> I wrote earlier:
>
>
>
>> It seems as if GridData would be useful. But, unless you are
>
>> working with 50x50 pixel blocks of data, GridData is ungodly
>
>> slow! Interpolate is probably better, but you don't have
>
>> many gridding options (linear, bilinear, cubic).
>
>
>
> By the way, I've demonstrated that GridData *does* do the right
>
> thing (sorta) *if* the data set resolution AND the output grid
>
> resolution is small enough. (And, maybe, if you run it on the
>
> right machine.) But, I need a more robust solution that works
>
> with the kinds of satellite data I use daily.
>

So your problem is basically that GRIDDATA is slow?

From my point of view, GRIDDATA is for gridding _irregular_ data, which is a hard problem. If your data is already on some regular grid, why would you want to triangulate? Regular interpolation is all that is needed if you do it the right way.

In general, I do the following to transform data from one grid to another:

1) Divide the _output_ grid in manageable blocks. What "manageable" means, will depend on many things, but for satellite data mostly on memory limitations. Overlap between the blocks might be necessary if you are going to include filtering/interpolation.

Repeat the following for your each of the output blocks:
2a) Calculate coordinates in _input_ grid corresponding to the grid points in your current output grid.
2b) Convert these coordinates to indices in the input data grid, including subpixels.
2c) Extract from your input dataset a "big enough" tile from the input grid. Or keep the entire input dataset in memory if it is already small enough.
2d) Choose gridding mode. If your output grid resolution is approximately the same or higher than the input grid resolution, nothing is needed here. However, if the output resolution is coarser, you might want to do something to avoid undersampling. E.g., smooth input data to match output resolution. Of course, if you have some missing data in the input grid, you must be careful at this point.
2e) Perform the actual regridding, using mapping indices from (2b), say XIND and YIND.
Nearest neighbour: outdata = indata[round(xind), round(yind)]
Bilinear interpolation: outdata = interpolate(indata, xind, yind)
Cubic interpolation: outdata = interpolate(indata, xind, yind, cubic=-0.5)
Again, take care if you have missing data in your input.

3) Glue together output blocks after elimination of possible block overlap.

Notes:
* For the regridding operation in (2e), cubic interpolation kernels might for some reason still not be good enough for you. In that case, it is not hard to implement fairly fast kernel based interpolators in IDL.

* For some kinds of satellite data with high dynamic range, it is sometimes better to perform the interpolation on the logarithm of the original data.

--
Yngvar

Subject: Re: weird behavior of Triangulate
Posted by DavidF[1] on Mon, 10 Sep 2012 15:26:29 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

> So your problem is basically that GRIDDATA is slow?

"Ungodly slow" is what I think I said. ;-)

> From my point of view, GRIDDATA is for gridding _irregular_ data, which is a hard problem. If your data is already on some regular grid, why would you want to triangulate? Regular interpolation is all that is needed if you do it the right way.

It would be wonderful to give people a pointer to the "right way" then. I've apparently been doing it the "wrong way" because I can't get it to work properly, even though I have been working on the problem, off and on, for years.

> In general, I do the following to transform data from one grid to another:
>
>
>
> 1) Divide the _output_ grid in manageable blocks. What "manageable" means, will depend on many things, but for satellite data mostly on memory limitations. Overlap between the blocks might be necessary if you are going to include filtering/interpolation.

It would be useful to have a robust algorithm for doing this kind of chunking. Or, in the absence of that, at least some general rules of thumb that people could use. I've never seen anything written down about this.

> Repeat the following for your each of the output blocks:
>
> 2a) Calculate coordinates in _input_ grid corresponding to the grid points in your current output grid.
>
> 2b) Convert these coordinates to indices in the input data grid, including subpixels.
>
> 2c) Extract from your input dataset a "big enough" tile from the input grid. Or keep the entire input dataset in memory if it is already small enough.
>
> 2d) Choose gridding mode. If your output grid resolution is approximately the same or higher than the input grid resolution, nothing is needed here. However, if the output resolution is coarser, you might want to do something to avoid undersampling. E.g., smooth input data to match output resolution. Of course, if you have some missing data in the input grid, you must be careful at this point.
>
> 2e) Perform the actual regridding, using mapping indices from (2b), say XIND and YIND.
>
> Nearest neighbour: outdata = indata[round(xind), round(yind)]
>
> Bilinear interpolation: outdata = interpolate(indata, xind, yind)
>
> Cubic interpolation: outdata = interpolate(indata, xind, yind, cubic=-0.5)

>
> Again, take care if you have missing data in your input.

> 3) Glue together output blocks after elimination of possible block overlap.

I appreciate the help. I'll see if I can find some time to have another go at this. I have been working on a "map_patch" alternative that sorta works. Perhaps I can fit these ideas into it in a reasonably robust fashion.

Cheers,

David

Subject: Re: weird behavior of Triangulate
Posted by envi35@yahoo.ca on Tue, 11 Sep 2012 03:21:29 GMT
View Forum Message <> Reply to Message

On Sep 6, 6:13 am, Klemen <klemen.zak...@gmail.com> wrote:
> Hi Jenny,
>
> I would suggest removing the line in your data having 90 degrees latitude. I think it should work then.
>
> I am using griddata often and it works well if your data-set is not too large. If yes, you can always process it by overlapping tiles. The only real disadvantage is gridding to spherical coordinates - griding to Cartesian coordinates is much faster.
>
> Klemen

Hi Klemen, sorry for delayed response - been away last week. Thanks very much for your suggestion. I did try removing the data at 90 degree latitude, but it still doesn't work. It is just so weird that the same code works only on my first set of data which is a bit larger than the second set. Perhaps I should try regular interpolation as suggested by Yngvar below.

Jenny

Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Tue, 11 Sep 2012 09:19:18 GMT
View Forum Message <> Reply to Message

On Monday, 10 September 2012 17:26:30 UTC+2, Coyote wrote:
> Yngvar Larsen writes:

>
>> From my point of view, GRIDDATA is for gridding _irregular_ data, which is a hard problem. If your data is already on some regular grid, why would you want to triangulate? Regular interpolation is all that is needed if you do it the right way.
>
> It would be wonderful to give people a pointer to the "right way"
> then. I've apparently been doing it the "wrong way" because I
> can't get it to work properly, even though I have been working on
> the problem, off and on, for years.

** Wrong Way **
Transform your perfectly regular input grid points to some other projection. This will then be irregular in the output domain, and you will need to use the Ungodly slooow TRIANGULATE/GRIDDATA approach to get a nice and regular grid in the output domain.

** Right Way **
Start with a regular grid in the _output_ domain. Calculate where these grid points are located in the input domain. Since your input data are on a nice and regular grid, INTERPOLATE or something similar will do the job fast and efficiently!

Basically: it is easier to interpolate an irregular grid from a regular grid than the other way around!

I fiddled around yesterday evening with the example on your web page

http://www.idlcoyote.com/code_tips/usegriddata.html

My solution to the "GridData Conundrum" is simply "Don't use GRIDDATA"...

```
8<--------------------------
;; Read input data
dataFile = 'usegriddata.dat'
nx = 144
ny = 73
indata = fltarr(nx, ny)
openr, unit, dataFile, /get_lun
readu, unit, indata
free_lun, unit

;; Create IDL coordinate mapper for polar stereographic grid.
;; See http://nsidc.org/data/polar_stereo/ps_grids.html
re = 6378273d0
e = 0.081816153d0
rp = sqrt((1-e^2)*re^2)
map = map_proj_init('polar stereographic', /gctp, $
                    center_latitude=70,          $
                    center_longitude=315,        $
                    semimajor_axis=re,           $
                    semiminor_axis=rp)
```

```
;; Define output grid.
dx = 25d3
dy = 25d3
xmin = -3.85d6
ymin = -5.35d6
nxout = 304L
nyout = 448L
mapx = xmin + dx*dindgen(nxout)
mapy = ymin + dy*dindgen(nyout)

mapx = mapx[*,lindgen(nyout)]
mapy = transpose(mapy[*,lindgen(nxout)])

;; Calculate output map grid values in input coordinates
latlon = map_proj_inverse(mapx, mapy, map_structure=map)
lat = reform(latlon[1,*], nxout, nyout)
lon = reform(latlon[0,*], nxout, nyout)

;; Transform lat/lon values to input grid indices.
xind = ((lon + 360) mod 360)/2.5
yind = (90 - lat)/2.5

;; Nearest neighbour
outdata_nn = indata[round(xind), round(yind)]
;; Linear interpolation
outdata_li = interpolate(indata, xind, yind)
;; Cubic interpolation
outdata_ci = interpolate(indata, xind, yind, cubic=-0.5)
8<--------------------------
```

Note that I used a different output projection than you did in your code. I guess you used a spherical earth equirectangular grid instead of the NSIDC polar stereographic grid you mentioned in the text just for simplicity? With a 25 km grid, it does not really matter much.

>> 1) Divide the _output_ grid in manageable blocks. What "manageable" means, will depend on many things, but for satellite data mostly on memory limitations. Overlap between the blocks might be necessary if you are going to include filtering/interpolation.
>

> It would be useful to have a robust algorithm for doing this kind of
> chunking. Or, in the absence of that, at least some general rules of
> thumb that people could use. I've never seen anything written down
> about this.

Hard to make a general rule about this, but here are some advices.

Overlap between patches/blocks: typically as long as your worst case filter length.

Size of patches/blocks: usually limited by memory, but sometimes the algorithm itself requires small pathes, e.g. TRIANGULATE. I usually buffer the regridded output blocks corresponding to a full stripe in the x-direction before writing to file so I can do it with a singe POINT_LUN+WRITEU operation instead of a long slow loop. This may limit the block size in the y-direction to avoid a very large buffer. If the input and the output projections are very different, you also have to think about the size of a rectangular input data block that is "big enough" to cover your output block. The default value of the output block in my system is 256x256, which will usually end up with a fast enough result; Not too big, which will cause memory problems. A nice dimension for FFT-based algorithms to work well, and not too many patches, so the double loop over the loops will not slow down things too much.

Your mileage may vary, so these things should be tuned to your computing system.

> I appreciate the help. I'll see if I can find some time to have another
> go at this. I have been working on a "map_patch" alternative that
> sorta works. Perhaps I can fit these ideas into it in a reasonably
> robust fashion.

A "map_patch" approach is _very_ useful since it scales very well if you suddenly want to use your algorithm for a dataset that is too big to fit in memory, or too slow to work on large blocks. Just make sure the patch you are talking about is situated in our _output_ grid, not in the _input_ grid!

At work, we have implemented this idea in an object oriented, data driven system. Each processing module is a class that basically implements a method GETDATA, which takes as input 4 parameters XSIZE/YSIZE/XPOS/YPOS that describes a patch in the output grid. GETDATA will then retrieve whatever data it needs from the previous processing module, and do its job. Then there is a base class method WRITE that assumes that this method exists, divides the output grid in blocks (or a rectangular subset of the output grid), makes a double loop over calls to GETDATA and dumps the result to file. Blocksize/overlap are settable parameters of these objects, with default values usually set in the INIT method.

Note that we use this framework also for general data processing, not only remapping. I wrote a conference paper about this many years ago. (Regridding as such is not explicitly mentioned however.)

http://earth.esa.int/fringe05/proceedings/papers/427_larsen. pdf

--
Yngvar

Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Tue, 11 Sep 2012 09:24:49 GMT
View Forum Message <> Reply to Message

On Tuesday, 11 September 2012 11:19:18 UTC+2, Yngvar Larsen wrote:

"[...] and not too many patches, so the double loop over the loops [...]"

Bah!

I meant to write "[...] a double loop over the patches [...]".

--
Yngvar

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Tue, 11 Sep 2012 19:53:28 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

```
>  ;; Create IDL coordinate mapper for polar stereographic grid.
>  ;; See http://nsidc.org/data/polar_stereo/ps_grids.html
>  re = 6378273d0
>  e = 0.081816153d0
>  rp = sqrt((1-e^2)*re^2)
>  map = map_proj_init('polar stereographic', /gctp, $
>                  center_latitude=70,          $
>                  center_longitude=315,        $
>                  semimajor_axis=re,           $
>                  semiminor_axis=rp)
>
>  ;; Define output grid.
>  dx = 25d3
>  dy = 25d3
>  xmin = -3.85d6
>  ymin = -5.35d6
>  nxout = 304L
>  nyout = 448L
>  mapx = xmin + dx*dindgen(nxout)
>  mapy = ymin + dy*dindgen(nyout)
>
>  mapx = mapx[*,lindgen(nyout)]
>  mapy = transpose(mapy[*,lindgen(nxout)])
>
>  ;; Calculate output map grid values in input coordinates
>  latlon = map_proj_inverse(mapx, mapy, map_structure=map)
```

I don't understand this step. In your explanation of the
"right way" you say:

> "Start with a regular grid in the _output_ domain.
> Calculate where these grid points are located in

the input domain."

I would have thought this means "use the map structure
of the input data" in this case. And, yet, you are using
the map structure of the output data grid. Yes, the
points will all fall within the input domain, since
that is a global domain. But, what about when the
input domain is NOT global, so that some of the points
are inside the output domain, but some are outside, too?

I can see that your program works, and it is EXTREMELY
fast. I just can't see *why* it works yet. :-)

Thanks for your help. I'm learning a lot! :-)

Cheers,

David

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

Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Wed, 12 Sep 2012 10:12:39 GMT
View Forum Message <> Reply to Message

On Tuesday, 11 September 2012 21:53:41 UTC+2, David Fanning wrote:
> Yngvar Larsen writes:
>
>> ;; Calculate output map grid values in input coordinates
>> latlon = map_proj_inverse(mapx, mapy, map_structure=map)
>
> I don't understand this step. In your explanation of the
> "right way" you say:
>
> "Start with a regular grid in the _output_ domain.
> Calculate where these grid points are located in
> the input domain."
>
> I would have thought this means "use the map structure
> of the input data" in this case.

I see. This example is a bit misleading, since the input grid is lat/lon. In general, it would look something like this:

```
;; Example: conversion from one UTM zone 33 to 34
map1 = map_proj_init('UTM', zone=33, datum=8)
map2 = map_proj_init('UTM', zone=34, datum=8)

;; Define output grid (UTM Z34 coordinates)
mapx = ...
mapy = ...

;; Calculate input grid coordinates for output grid
latlon = map_proj_inverse(mapx, mapy, map_structure=map2)
utm33 = map_proj_forward(latlon[1,*], latlon[0,*], map_structure=map1)

;; xcoord_in/ycoord_in are the coordinates of the grid points of your input grid.
;; dx_in/dy_in is the grid resolution in x/y direction.
xind = (utm33[0,*] - xcoord_in[0,0])/dx_in
yind = (utm34[1,*] - ycoord_in[0,0])/dy_in
;; or
;;   yind = (ycoord_in[0,0] - coord[1,*])/dy_in
;; if first line of the input data array is "upper row"/"northernmost row" like in your example
```

> And, yet, you are using the map structure of the output data grid.

Yes, that is the whole point! However, as I showed above, if the input grid is not lat/lon, you need also the map structure for the input data grid.

> Yes, the points will all fall within the input domain, since
> that is a global domain. But, what about when the
> input domain is NOT global, so that some of the points
> are inside the output domain, but some are outside, too?

I'm not sure if my brain parsed that sentence correctly, but I think I know what you mean. I omitted some details in my program:

* how to handle points in your defined output grid that fall outside the available input grid

In your example, the input grid was global, so no problem. For the nearest neighbour interpolation, you have to check if the calculated XIND and YIND are inside the interval [0, X/YSIZE-1] and handle the points that are not separately, e.g. insert NaN or something else in your output grid. For the other two, use the MISSING keyword to INTERPOLATE.

* how to handle missing data in your input grid (NaNs, "magic" values like -9999, etc). I.e. _almost_ regular input grid.

Your example did not have this problem. Interpolation might be tricky in this case. This is left as an exercise for the reader :)

* If you don't want to define the output grid explicitly, how to calculate automatically an output grid

that covers the available input data.

For your example, where the input data are global, I'm not sure if this is even possible. But for most cases, you just use map_proj_forward(..., map=inputmap)+ map_proj_inverse(..., map=outputmap), and use MAX/MIN + FLOOR/CEIL on the resulting coordinates.

--
Yngvar

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Wed, 12 Sep 2012 14:42:03 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

> ;; xcoord_in/ycoord_in are the coordinates of the grid points of your input grid.
> ;; dx_in/dy_in is the grid resolution in x/y direction.
> xind = (utm33[0,*] - xcoord_in[0,0])/dx_in
> yind = (utm34[1,*] - ycoord_in[0,0])/dy_in
> ;; or
> ;; yind = (ycoord_in[0,0] - coord[1,*])/dy_in
> ;; if first line of the input data array is "upper row"/"northernmost row" like in your example

Thanks, Yngvar. Your explanation now squares with
what I thought I understood. :-)

My biggest problem is figuring out how to get index
arrays. I seem to have a mental block against figuring
it out. As I pondered the problem yesterday, I discovered
that I could use Scale_Vector to create the index arrays.
Since I *do* understand Scale_Vector, this has helped
tremendously. I still get confused about the index
values for latitudes. Do they have to get reversed or
not!? Maybe not, if I already reversed the data... etc.
Sheesh!

Some data sets lend themselves to checking. Others not
so much. Throw in a deep suspicion of anything coming
out of the Map function, and you have the makings of
a deep paranoia. Still, I feel like I am making some
progress. :-)

Cheers,

David

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

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Wed, 12 Sep 2012 16:11:35 GMT
View Forum Message <> Reply to Message

Coyote writes:

> I appreciate the help. I'll see if I can find some time to have another
> go at this. I have been working on a "map_patch" alternative that
> sorta works. Perhaps I can fit these ideas into it in a reasonably
> robust fashion.

Yowser! I incorporated this "right way" interpolation
scheme into my "Map_Patch" alternative function and it
is blisteringly fast!

This is the well-behaved solution. Now I have to work
on the more difficult scenarios. But, this is exciting!

Cheers,

David

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

Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Fri, 14 Sep 2012 07:07:06 GMT
View Forum Message <> Reply to Message

On Wednesday, 12 September 2012 16:42:17 UTC+2, David Fanning wrote:
> Yngvar Larsen writes:
>
>

>
>> ;; xcoord_in/ycoord_in are the coordinates of the grid points of your input grid.
>> ;; dx_in/dy_in is the grid resolution in x/y direction.
>> xind = (utm33[0,*] - xcoord_in[0,0])/dx_in
>> yind = (utm33[1,*] - ycoord_in[0,0])/dy_in

[Edit: utm34 -> utm33 in the previous line.]

>> ;; or
>> ;; yind = (ycoord_in[0,0] - coord[1,*])/dy_in
>> ;; if first line of the input data array is "upper row"/"northernmost row" like in your example
>
> My biggest problem is figuring out how to get index
> arrays. I seem to have a mental block against figuring
> it out. As I pondered the problem yesterday, I discovered
> that I could use Scale_Vector to create the index arrays.
> Since I *do* understand Scale_Vector, this has helped
> tremendously.

Well it *is* a "scale_vector" operation, so applied correctly, I'm sure SCALE_VECTOR might ease the cognitive load.

> I still get confused about the index values for latitudes. Do they have to get reversed or
> not!? Maybe not, if I already reversed the data... etc.

That's what my comment in the code snippet above is about. ENVI and many other GIS-like software packages and data formats like to have [0,0] in the upper left corner, while IDL doesn't. ENVI also starts counting from [1,1] for some very strange reason. I really don't want to see the ENVI source code! Imagine implementing indexing from [1,1] in a language that doesn't...

Personally, I always reverse the data when I read from disk, such that [0,0] is in the lower left corner the IDL way. I can then treat x and y coordinates the same way.

--
Yngvar

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Fri, 14 Sep 2012 12:17:23 GMT
View Forum Message <> Reply to Message

Yngvar Larsen writes:

> Personally, I always reverse the data when I read from disk,
> such that [0,0] is in the lower left corner the IDL way. I
> can then treat x and y coordinates the same way.

This is what I do, too. I had some code that could

figure this out, but then I took it out, deciding
that if people couldn't tell if their data was upside
down on their own, my pointing it out to them probably
wouldn't help much. ;-)

Cheers,

David

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

Subject: Re: weird behavior of Triangulate
Posted by David Fanning on Mon, 17 Sep 2012 12:56:47 GMT
View Forum Message <> Reply to Message

Yngvar Larsen wrote a few days ago:

> From my point of view, GRIDDATA is for gridding _irregular_ data,
> which is a hard problem. If your data is already on some regular
> grid, why would you want to triangulate? Regular interpolation
> is all that is needed if you do it the right way.

Since I don't really understand something until and unless
I write it down (and you thought I maintain this web page
for you!), I have modified my article on this topic to
include (thanks to Yngvar's invaluable help) the fast
way to do this interpolation.

I even managed to reason my way out of a problem that caused
my gridded data to be an upside-down mirror image of what
I expected and wanted. In the process, I think I actually
came to understand what I was doing when I was creating
the fractional indices necessary to do interpolation
correctly.

And, since interpolation is orders of magnitude faster
than gridding the data (which may NEVER finish, as far
as I know, when using real-world satellite data), the
pain of learning this new (for me, anyway) technique
is more than offset by the benefits.

You can learn more about it at the end of this article:

http://www.idlcoyote.com/code_tips/usegriddata.html

Cheers,

David

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