Subject: Gridding to the Surface of a Sphere
Posted by David Fanning on Sun, 14 Apr 2013 16:20:09 GMT
View Forum Message <> Reply to Message

Folks,

Quite frequently you find yourself with randomly positioned data values that are associated with a latitude and longitude value. You often want to display this kind of data as a contour plot on a map projection. Traditionally, the Triangulate/Trigrid method is used to grid random data values into a 2D grid that can be contoured. And, there is provision in this method for gridding to the "surface of a sphere," which seems like a good thing to do for latitude/longitude data.

But, you would be gravely mistaken. :-)

Personally, I think the Triangulate/Trigrid gridding method for creating a grid on the surface of a sphere is tragically flawed. (Although I would be happy to discover otherwise.) I have outlined in some detail my reasons for thinking this in the following article:

http://www.idlcoyote.com/code_tips/sphericalgrid.php

I also illustrate how this can be done correctly by using GridData to do the gridding to the sphere, rather than the Triangulate/Trigrid method.

There is one strange thing about the GridData method that I don't understand and don't mention in the article. Maybe someone can help me with this. The GridData methods I illustrate (NaturalNeighbor and InverseDistance) require that I supply Delaunay triangles to the GridData program. If I create the triangles with Triangulate, all is well. If I create the triangles with QHull, the GridData program chokes. Does anyone have any insight into why that would be?

You can find code and data in the article if you care to fool around with this.

Cheers,

David

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

Subject: Re: Gridding to the Surface of a Sphere
Posted by clive.best on Mon, 20 Mar 2017 06:12:33 GMT
View Forum Message <> Reply to Message

On Monday, 15 April 2013 02:20:09 UTC+10, David Fanning wrote:
> Folks,
>
> Quite frequently you find yourself with randomly positioned data values
> that are associated with a latitude and longitude value. You often want
> to display this kind of data as a contour plot on a map projection.
> Traditionally, the Triangulate/Trigrid method is used to grid random
> data values into a 2D grid that can be contoured. And, there is
> provision in this method for gridding to the "surface of a sphere,"
> which seems like a good thing to do for latitude/longitude data.
>
> But, you would be gravely mistaken. :-)
>
> Personally, I think the Triangulate/Trigrid gridding method for creating
> a grid on the surface of a sphere is tragically flawed. (Although I
> would be happy to discover otherwise.) I have outlined in some detail my
> reasons for thinking this in the following article:
>
>   http://www.idlcoyote.com/code_tips/sphericalgrid.php
>
> I also illustrate how this can be done correctly by using GridData to do
> the gridding to the sphere, rather than the Triangulate/Trigrid method.
>
> There is one strange thing about the GridData method that I don't
> understand and don't mention in the article. Maybe someone can help me
> with this. The GridData methods I illustrate (NaturalNeighbor and
> InverseDistance) require that I supply Delaunay triangles to the
> GridData program. If I create the triangles with Triangulate, all is
> well. If I create the triangles with QHull, the GridData program chokes.
> Does anyone have any insight into why that would be?
>
> You can find code and data in the article if you care to fool around
> with this.
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
> Sepore ma de ni thue. ("Perhaps thou speakest truth.")

I want to use the spherical triangulation itself rather than grid it to a regular grid. It turns out that

this is a smart way to perform area averaging of global temperature data. So I really want to use the output structure Sphere=s from Triangulate. This turns out to be no easy task. IDL provides no documentation. They clearly don't want you to use it directly but pass it directly through to TRIGRID.

S is a structure with the following pattern for 1880 (much larger in later years)

```
 XYZ           DOUBLE    Array[853, 3]
   IEND          LONG      Array[853]
   IADJ          LONG     Array[5118]
```

XYZ are the cartesian coordinates on a unit sphere. However the axes bear no relation to (Lat,Lon) Latitude seems to be a linear combination of X+Y while Lon spans the z-axis.

IEND is a pointer to the last triangle for each coordinate in xyz. The triangles are defined (I think) in IADJ. Ntriangles : 5118/3 = 1706 triangles as triplet pointers into XYZ.

However when I plot the grid as triangles I get strange results. Every time I think I have solved the riddle - I get a surprise.

Has anyone got a solution to how to interpret Sphere=S ?

Subject: Re: Gridding to the Surface of a Sphere
Posted by Robert.M.Candey on Tue, 21 Mar 2017 03:11:29 GMT
View Forum Message <> Reply to Message

For what it is worth, I implemented a few different mappings to a sphere in auroral_image.pro < https://spdf.gsfc.nasa.gov/pub/software/cdawlib/source/auror al_image.pro> long ago. Methods include using triangulate, trigrid, and dilate: TV bitmap, Quick plotting of Z array only (no map), plots command, polyfill, nearest neighbor filling, map_image, convert_coord and dilate, and Map Overlay. I suppose I should add some of these newer techniques. Looks like I tried to figure out the output of "sphere=" as well.

Subject: Re: Gridding to the Surface of a Sphere
Posted by Dick Jackson on Fri, 24 Mar 2017 21:31:28 GMT
View Forum Message <> Reply to Message

On Sunday, 19 March 2017 23:12:35 UTC-7, clive...@gmail.com wrote:
> On Monday, 15 April 2013 02:20:09 UTC+10, David Fanning wrote:
>> Folks,
>>
>> Quite frequently you find yourself with randomly positioned data values
>> that are associated with a latitude and longitude value. You often want
>> to display this kind of data as a contour plot on a map projection.
>> Traditionally, the Triangulate/Trigrid method is used to grid random

>> data values into a 2D grid that can be contoured. And, there is
>> provision in this method for gridding to the "surface of a sphere,"
>> which seems like a good thing to do for latitude/longitude data.
>>
>> But, you would be gravely mistaken. :-)
>>
>> Personally, I think the Triangulate/Trigrid gridding method for creating
>> a grid on the surface of a sphere is tragically flawed. (Although I
>> would be happy to discover otherwise.) I have outlined in some detail my
>> reasons for thinking this in the following article:
>>
>>   http://www.idlcoyote.com/code_tips/sphericalgrid.php
>>
>> I also illustrate how this can be done correctly by using GridData to do
>> the gridding to the sphere, rather than the Triangulate/Trigrid method.
>>
>> There is one strange thing about the GridData method that I don't
>> understand and don't mention in the article. Maybe someone can help me
>> with this. The GridData methods I illustrate (NaturalNeighbor and
>> InverseDistance) require that I supply Delaunay triangles to the
>> GridData program. If I create the triangles with Triangulate, all is
>> well. If I create the triangles with QHull, the GridData program chokes.
>> Does anyone have any insight into why that would be?
>>
>> You can find code and data in the article if you care to fool around
>> with this.
>>
>> Cheers,
>>
>> David
>>
>> --
>> David Fanning, Ph.D.
>> Fanning Software Consulting, Inc.
>> Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
>> Sepore ma de ni thue. ("Perhaps thou speakest truth.")
>
> I want to use the spherical triangulation itself rather than grid it to a regular grid. It turns out that this is a smart way to perform area averaging of global temperature data. So I really want to use the output structure Sphere=s from Triangulate. This turns out to be no easy task. IDL provides no documentation. They clearly don't want you to use it directly but pass it directly through to TRIGRID.
>
> S is a structure with the following pattern for 1880 (much larger in later years)
>
> XYZ   DOUBLE   Array[853, 3]
>   IEND   LONG   Array[853]
>   IADJ   LONG   Array[5118]

>
> XYZ are the cartesian coordinates on a unit sphere. However the axes bear no relation to (Lat,Lon) Latitude seems to be a linear combination of X+Y while Lon spans the z-axis.
>
> IEND is a pointer to the last triangle for each coordinate in xyz. The triangles are defined (I think) in IADJ. Ntriangles : 5118/3 = 1706 triangles as triplet pointers into XYZ.
>
> However when I plot the grid as triangles I get strange results. Every time I think I have solved the riddle - I get a surprise.
>
> Has anyone got a solution to how to interpret Sphere=S ?

[sorry if this appears twice… my first post attempt didn't seem to appear]

Hi,

I haven't worked much with spherical gridding, but looking at current help online:
http://www.harrisgeospatial.com/docs/TRIANGULATE.html

… there's a note that "To perform spherical gridding, you must include the FVALUE and SPHERE keywords described below." and that "On output, the elements of FVALUE are rearranged to correspond to the new ordering of X and Y (as described in the SPHERE keyword, below)."… where it says "The X and Y parameters are converted to double precision and are rearranged to match the spherical triangulation."

Could this odd implementation be the cause of your trouble? Making some random x, y and 'f' data:

```
IDL> x=RandomU(seed,10)*360-180
IDL> y=RandomU(seed,10)*180-90
IDL> f=RandomU(seed,10)
IDL> Transpose([[x],[y],[f]])
      12.644104      -68.391212       0.48885179
      164.58990      -37.729668       0.84256339
      115.67722       76.152237       0.24543986
     -93.728912      -76.539368       0.63267028
      173.01416       6.1176758      0.032040875
      80.273712      -34.140186       0.99399608
      78.493317      -72.667831      0.069946259
     -27.613220      -10.058014       0.59602672
      30.329697       66.785294       0.94615310
      138.58615       82.033173       0.22554629
IDL> Triangulate, x, y, tri, SPHERE=sphere, /DEGREES, FVALUE=f

; The (x, y, f) triples have all been juggled, but kept together as triples:
IDL> Transpose([[x],[y],[f]])
     -93.728912353515625      -76.539367675781250       0.63267028331756592
      78.493316650390625      -72.667831420898438      0.069946259260177612
```

```
      12.644104003906250      -68.391212463378906       0.48885178565979004
      80.273712158203125      -34.140186309814453       0.99399608373641968
     -27.613220214843750      -10.058013916015625       0.59602671861648560
      164.58990478515625      -37.729667663574219       0.84256339073181152
      173.01416015625000       6.1176757812500000       0.032040875405073166
      30.329696655273438       66.785293579101562       0.94615310430526733
      115.67721557617188       76.152236938476562       0.24543985724449158
      138.58615112304688       82.033172607421875       0.22554628551006317
IDL> help,sphere
** Structure <2809d28>, 3 tags, length=520, data length=520, refs=1:
   XYZ             DOUBLE    Array[10, 3]
   IEND            LONG      Array[10]
   IADJ            LONG      Array[60]
IDL> sphere
{
    "XYZ": [-0.015138866167300823, 0.059427928082923090,
0.35933613894835048, 0.13982738150556639, 0.87247861224215495,
-0.76247257294752480, -0.98692363203545608, 0.34022824979991301,
-0.10370746692854957, -0.10394304246629128, -0.23228439147375979,
0.29192329646871329, 0.080611616166783071, 0.81577007429463544,
-0.45637715920335836, 0.21016448049027153, 0.12093140548051810,
0.19904996545014692, 0.21570767589416645, 0.091682779818078652,
-0.97253009013029190, -0.95459368861448624, -0.92972001515761238,
-0.56121964440548744, -0.17464523966451509, -0.61193665247559381,
0.10657082010780514, 0.91903419375580542, 0.97093509559784741,
0.99034847998453579],
    "IEND": [5, 9, 13, 20, 26, 30, 36, 40, 44, 48],
    "IADJ": [2, 3, 5, 7, 6, 4, 3, 1, 6, 1, 2, 4, 5, 5, 3, 2, 6, 7, 9,
8, 7, 1, 3, 4, 8, 10, 4, 2, 1, 7, 4, 6, 1, 5, 10, 9, 9, 10, 5, 4, 4,
7, 10, 8, 7, 5, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
}
IDL> Transpose(sphere.xyz)
    -0.015138866167300823     -0.23228439147375979     -0.97253009013029190
     0.059427928082923090      0.29192329646871329     -0.95459368861448624
      0.35933613894835048     0.080611616166783071     -0.92972001515761238
      0.13982738150556639      0.81577007429463544     -0.56121964440548744
      0.87247861224215495     -0.45637715920335836     -0.17464523966451509
     -0.76247257294752480      0.21016448049027153     -0.61193665247559381
     -0.98692363203545608      0.12093140548051810      0.10657082010780514
      0.34022824979991301      0.19904996545014692      0.91903419375580542
     -0.10370746692854957      0.21570767589416645      0.97093509559784741
     -0.10394304246629128     0.091682779818078652      0.99034847998453579
```

These appear to be (x,y,z) locations on the unit sphere for the juggled points, if you look at the globe from above the North Pole, with the prime meridian being toward the +X axis. (point 0 is in Marie Byrd Land, Antarctica; point 7 is in NW Russia)

If we use "tri" we get a good triangulation of the points:

```
IDL> conn=[Replicate(3, [1, 16]), tri]
IDL> o=IDLgrPolygon(Transpose(sphere.xyz), POLYGON=conn, COLOR=[128,128,192])
IDL> xobjview,o
```

The sphere.iend and sphere.iadj are a bit of a puzzle, but I think I figured it out. They describe the "adjacencies" that define all the needed *edges* for the triangulation, grouped into a set of closed polygons, I'm guessing to give an efficient way to draw a wireframe without drawing all triangles which would duplicate each edge. IEND gives the ending index for each polygon (indexing from 1, not 0), so that here, the first polygon takes 5 points (1–5) from the IADJ index list, the second takes 4 points (6–9), and so on. I group the IADJ values to illustrate here:

```
   "IEND": [5, 9, 13, 20, 26, 30, 36, 40, 44, 48],
   "IADJ": [
2, 3, 5, 7, 6,
4, 3, 1, 6,
1, 2, 4, 5,
5, 3, 2, 6, 7, 9, 8,
7, 1, 3, 4, 8, 10,
4, 2, 1, 7,
4, 6, 1, 5, 10, 9,
9, 10, 5, 4,
4, 7, 10, 8,
7, 5, 8, 9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] ; all ignored
```

I don't know if there's anything special about how they're grouped… and I see some edges are duplicated (see bold "8, 10" and "10, 8")… wait a minute… there are 48 edges here, and in fact *all* of them are duplicated (remember, closed polygons join the last point back to the first), and 48 edges is exactly the same as with the 16 triangles given in "tri"! OK, I now have no idea why IEND and IADJ are the way they are. :-) TRIGRID seems to like them, though!

This program shows what I explain above, that every edge is covered by two of these polygons:

```
PRO TriangulateTest

; Exploring the output of TRIANGULATE, SPHERE=...
; Dick Jackson, March, 2017

x=RandomU(seed,10)*360-180
y=RandomU(seed,10)*180-90
f=RandomU(seed,10)

Triangulate, x, y, tri, SPHERE=sphere, /DEGREES, FVALUE=f

; Make a polygon mesh from "tri" and the normalized sphere.xyz coordinates
```

```
conn=[Replicate(3, [1, N_Elements(tri)/3]), tri]
o=IDLgrPolygon(Transpose(sphere.xyz), POLYGON=conn, COLOR=[128,128,192])

;XObjView,o ; To see the reasonable polygon mesh from the triangulation

o3 = []
currI = 0L
FOREACH iendI, sphere.iend, polyI DO BEGIN
  ; Make one polygon from next set of values in sphere.iadj
  conn3 = [iendI-currI, sphere.iadj[currI:iendI-1]-1]
  o3 = [o3, IDLgrPolygon(Transpose(sphere.xyz*(1.01+0.01*polyI)), $
       POLYGON=conn3, COLOR=RandomU(seed, 3)*256, STYLE=1, THICK=3)]
  currI = iendI
ENDFOREACH

XObjView, [o, o3] ; To see the mesh and these polygons

END
```

I hope this is helpful!

Cheers,
-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada --- http://www.d-jackson.com
