Subject: Re: 2D interpolation with sparse data
Posted by ben.bighair on Tue, 22 May 2007 17:06:41 GMT
View Forum Message <> Reply to Message

On May 22, 10:38 am, Ken G <kagoldb...@gmail.com> wrote:
> Here's an interesting interpolation problem.
>
> Suppose I have a coarsely sampled 2D dataset--an image. There are
> several ways to fill-in the missing data, including TRIGRID and
> TRI_SURF, etc. What I find though, is that these methods can introduce
> severe artifacts due to the nature of the triangulation.
>
> This example figure here shows the problem clearly:
> http://goldberg.lbl.gov/newsgroup/interpolation_problem.jpg[28k]
>
> My original image has simple, horizontal bands with no vertical
> features. My sparse sampling is collected at striped angles, as you
> can see. I realize that these interpolations aren't 'wrong' per se:
> the way in which they are triangulated strongly affects the final
> result.
>
> Short of re-writing my own triangulation routine, I am wondering if
> there is already a way that I can tell TRIANGULATE to prefer
> triangulation along the x-direction, for example, which in this case
> would solve the problem. Or if there is another built-in routine that
> might work better for me?
>
> I have tried using various Fourier filtering ideas that didn't work
> out as well as I had hoped. I also tried rotating my data-set in
> various ways, triangulating, and then rotating back. So far, those
> ideas haven't worked either.
>

Hello,

I would give IDL's GRIDDATA some scrutiny. For a number of the
sampling methods it offeres you can control its result with either the
ANISOTROPY and SEARCH_ELLIPSE. Once you get the hang of GRIDDATA (and
its companion GRID_INPUT) it can be quite handy.

Cheers,
Ben

Subject: Re: 2D interpolation with sparse data
Posted by cmancone on Wed, 23 May 2007 12:39:54 GMT
View Forum Message <> Reply to Message

On May 22, 10:38 am, Ken G <kagoldb...@gmail.com> wrote:
> Here's an interesting interpolation problem.
>
> Suppose I have a coarsely sampled 2D dataset--an image. There are
> several ways to fill-in the missing data, including TRIGRID and
> TRI_SURF, etc. What I find though, is that these methods can introduce
> severe artifacts due to the nature of the triangulation.
>
> This example figure here shows the problem clearly:
> http://goldberg.lbl.gov/newsgroup/interpolation_problem.jpg[28k]
>
> My original image has simple, horizontal bands with no vertical
> features. My sparse sampling is collected at striped angles, as you
> can see. I realize that these interpolations aren't 'wrong' per se:
> the way in which they are triangulated strongly affects the final
> result.
>
> Short of re-writing my own triangulation routine, I am wondering if
> there is already a way that I can tell TRIANGULATE to prefer
> triangulation along the x-direction, for example, which in this case
> would solve the problem. Or if there is another built-in routine that
> might work better for me?
>
> I have tried using various Fourier filtering ideas that didn't work
> out as well as I had hoped. I also tried rotating my data-set in
> various ways, triangulating, and then rotating back. So far, those
> ideas haven't worked either.
>
> Any ideas?
>
> Thanks,
> Ken G

Maybe this is just me, but I'd just write an interpolation routine.
All you need is simle linear interpolation along one direction, which
is a simple enough problem to solve.

Subject: Re: 2D interpolation with sparse data
Posted by Ken G on Wed, 23 May 2007 14:42:41 GMT
View Forum Message <> Reply to Message

Thank you both for the suggestions!

I did try a strictly-x line-by-line interpolation and that certainly
does work (and quickly) to fill in the missing points. Yet, a problem
might occur when there's a non-x-dependence to the original data. But
I came up with a different idea.

I was originally been using the TRIANGULATE -> TRIGRID path to solve
this interpolation and fill-in the missing data. TRIANGULATE computes
the 'triangles' connectivity based on the simple distance between
points, which means we can trick it to believe that points in x or y
are closer or farther apart. I simply added a constant multiplier to
the y positions in the input arguments to the TRIANGULATE procedure
and it completely changes the triangulation being used. I got the idea
when I looked at the matrix formulation in this article
  http://en.wikipedia.org/wiki/Delaunay_triangulation

So the triangulate call looks like this:
  TRIANGULATE, x, y*scaling_factor, triangles, boundary

and the whole interpolation looks more or less like this (pared down)

w = where(img NE 0)
wx = w mod Nx
wy = w / Nx
scaling = 4L
TRIANGULATE, wx, wy*scaling, triangles, boundary
img2 = TRIGRID(wx,wy,img(w), triangles, [1,1], [0,0,Nx-1,Ny-1], Nx=Nx,
Ny=Ny)

Subject: Re: 2D interpolation with sparse data
Posted by cmancone on Wed, 23 May 2007 18:57:51 GMT
View Forum Message <> Reply to Message

Interesting solution. There's only one thing I'd be worried about if
I were you, and that is: how sensitive is the result to your scaling
factor? Do you have to change the scaling factor depending on how
"bright" your data is? If the scaling factor is low enough, do you
get non-x contributions? If you haven't already, I would make sure
the answers to those questions are to your satisfaction.

On May 23, 10:42 am, Ken G <kagoldb...@gmail.com> wrote:
> Thank you both for the suggestions!
>
> I did try a strictly-x line-by-line interpolation and that certainly
> does work (and quickly) to fill in the missing points. Yet, a problem
> might occur when there's a non-x-dependence to the original data. But
> I came up with a different idea.
>
> I was originally been using the TRIANGULATE -> TRIGRID path to solve
> this interpolation and fill-in the missing data. TRIANGULATE computes
> the 'triangles' connectivity based on the simple distance between
> points, which means we can trick it to believe that points in x or y

> are closer or farther apart. I simply added a constant multiplier to
> the y positions in the input arguments to the TRIANGULATE procedure
> and it completely changes the triangulation being used. I got the idea
> when I looked at the matrix formulation in this article
> http://en.wikipedia.org/wiki/Delaunay_triangulation
>
> So the triangulate call looks like this:
> TRIANGULATE, x, y*scaling_factor, triangles, boundary
>
> and the whole interpolation looks more or less like this (pared down)
>
> w = where(img NE 0)
> wx = w mod Nx
> wy = w / Nx
> scaling = 4L
> TRIANGULATE, wx, wy*scaling, triangles, boundary
> img2 = TRIGRID(wx,wy,img(w), triangles, [1,1], [0,0,Nx-1,Ny-1], Nx=Nx,
> Ny=Ny)

Subject: Re: 2D interpolation with sparse data
Posted by cmancone on Wed, 23 May 2007 18:58:04 GMT
View Forum Message <> Reply to Message

Interesting solution. There's only one thing I'd be worried about if I were you, and that is: how sensitive is the result to your scaling factor? Do you have to change the scaling factor depending on how "bright" your data is? If the scaling factor is low enough, do you get non-x contributions? If you haven't already, I would make sure the answers to those questions are to your satisfaction.

On May 23, 10:42 am, Ken G <kagoldb...@gmail.com> wrote:
> Thank you both for the suggestions!
>
> I did try a strictly-x line-by-line interpolation and that certainly
> does work (and quickly) to fill in the missing points. Yet, a problem
> might occur when there's a non-x-dependence to the original data. But
> I came up with a different idea.
>
> I was originally been using the TRIANGULATE -> TRIGRID path to solve
> this interpolation and fill-in the missing data. TRIANGULATE computes
> the 'triangles' connectivity based on the simple distance between
> points, which means we can trick it to believe that points in x or y
> are closer or farther apart. I simply added a constant multiplier to
> the y positions in the input arguments to the TRIANGULATE procedure
> and it completely changes the triangulation being used. I got the idea
> when I looked at the matrix formulation in this article
> http://en.wikipedia.org/wiki/Delaunay_triangulation

>
> So the triangulate call looks like this:
> TRIANGULATE, x, y*scaling_factor, triangles, boundary
>
> and the whole interpolation looks more or less like this (pared down)
>
> w = where(img NE 0)
> wx = w mod Nx
> wy = w / Nx
> scaling = 4L
> TRIANGULATE, wx, wy*scaling, triangles, boundary
> img2 = TRIGRID(wx,wy,img(w), triangles, [1,1], [0,0,Nx-1,Ny-1], Nx=Nx,
> Ny=Ny)
