Subject: match_2d
Posted by vino on Mon, 20 Apr 2009 13:40:19 GMT
View Forum Message <> Reply to Message

Hello everyone,

I am using match_2d from J.D.Smith's library (http://
tir.astro.utoledo.edu/idl/match_2d.pro) to match between two
catalogues. Below is what i am doing :

```
IDL> print,star_radec1
     242.759     -29.4162
     252.666     -31.6364
     250.523     -30.5292
     244.782     -20.2181
IDL> print,star_radec2
     252.666     -31.6364
     250.523     -30.5292
     267.782     -22.9120
IDL>  match=match_2d(star_radec1(0,*),star_radec1(1,*),star_radec2
(0,*),star_radec2(1,*),.02,MATCH_DISTANCE=md)
IDL> print,match
        -1        -1        -1        -1
IDL> print,md
  5.98943e-22      3.62562  2.23579e-17  2.62037e-22
```

Eventhough there are 3 objects matching, why is it that match_2d is
not finding them?? I shall be very grateful for any pointers...

Thanks and regards,

Vino

Subject: Re: match_2d
Posted by vino on Mon, 27 Apr 2009 10:53:56 GMT
View Forum Message <> Reply to Message

Hi Jeremy!

Thank you for ur reply...

Unfortunately as you correctly guessed my array is too big even when i
split it into chunks.. :(

eg:

RA1 FLOAT = Array[267679]
DEC1 FLOAT = Array[267679]
RA2 FLOAT = Array[235476]
DEC2 FLOAT = Array[235476]

So i guess any form of brute force array method is going to turn up memory issues!! :(
Since all my stellar photometry data will be finally loaded into a data base, i just have to make sure i gather all the information for a particular object before further analysis...It is a round about way but saves a lot of initial processing time....

Thank you so much for your help...

Regards

Vino

On Apr 24, 6:50 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 23, 8:10 am, vino <astrocr...@gmail.com> wrote:
>
>
>
>> Hi Jeremy!!
>
>> Thank you very much for helping me out....It works very well with my
>> data set...
>> For me to be able to use this routine is going to save me about a
>> couple of weeks of runtime in my program!!
>
>> I have looked at WITHINSPHRAD but in that case, i still need to have
>> a loop which is what i was trying to avoid!!
>
>> Thanks to J.D.Smith for giving us a boon with routines like this!! (i
>> will someday learn how to use histogram)..
>
>> Regards,
>
>> Vino
>
>> On Apr 22, 11:39 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>>> Aha... I've looked at it in gory detail, and it turns out that the

>>>> routine implicitly assumes that the minimum value of both x2 and y2
>>>> are 0. So you can get it to work if you do the following:
>
>>> Aha! Thanks for the catch. That's what you get when you evaluate an
>>> algorithm on artificial random coordinates ranging uniformly from
>>> [0,1].
>
>>> I've updated MATCH_2D at the address mentioned to handle this issue
>>> explicitly, and also catch cases of matching points which fall just
>>> slightly outside the bounding box of the search set. I've also added
>>> a much-needed warning regarding using this Euclidean matching
>>> algorithm for points on the sphere (e.g. star positions, lat/lon,
>>> etc.):
>
>>> ; WARNING:
>>> ;
>>> ;   Distance is evaluated in a strict Euclidean sense. For
>>> ;   points on a sphere, the distance between two given
>>> ;   coordinates is *not* the Euclidean distance. As an extreme
>>> ;   example, consider two points very near the N. pole, but on
>>> ;   opposite sides (one due E, one due W). For small patches,
>>> ;   this Euclidean assumption is approximately valid, and the
>>> ;   method works. See NOTES above for a tip regarding obtaining
>>> ;   a (more) uniform match criterion on the sphere.
>>> ;;
>
>>> Give this version a try. By the way, the value of MATCH_DISTANCE for
>>> points which did *not* match is not meaningful.
>
>>> JD
>
> That, of course, is a challenge. ;-) Try this version, which will
> allow you to do many-to-many matches:
>
>   http://www.physics.mcmaster.ca/~bailinj/idl/withinsphrad_vec .pro
>
> It uses the "throw lots of memory at the problem" paradigm (it
> internally uses several N1 x N2 arrays simultaneously), so you may
> find that it runs out of memory fairly quickly. If it's a problem, you
> can always try chunking up your coordinates and doing a FOR loop
> through the chunks - it should at least be faster than looping through
> each coordinate.
>
> I'm pretty sure there's a HIST_ND-based algorithm of doing this
> similar to MATCH_2D but taking spherical trig into account, but I
> don't have the patience to figure it out.
>
> -Jeremy.

Subject: Re: match_2d
Posted by JDS on Mon, 27 Apr 2009 19:06:37 GMT
View Forum Message <> Reply to Message

> I'm pretty sure there's a HIST_ND-based algorithm of doing this
> similar to MATCH_2D but taking spherical trig into account, but I
> don't have the patience to figure it out.

That would be challenging for the whole sphere, since histogram can only evaluate monotonic coordinate fields. You can always first remap your coordinates using some projection which puts the ill-behaved parts (nominally, the poles) far away, and preserves distance locally. For example, if you have a small field (a degree or so) near the pole, this would be a nice way of solving the converging longitude lines issues. But generally? Sounds tough.

JD

Subject: Re: match_2d
Posted by Jeremy Bailin on Wed, 29 Apr 2009 02:44:09 GMT
View Forum Message <> Reply to Message

On Apr 27, 3:06 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>> I'm pretty sure there's a HIST_ND-based algorithm of doing this
>> similar toMATCH_2Dbut taking spherical trig into account, but I
>> don't have the patience to figure it out.
>
> That would be challenging for the whole sphere, since histogram can
> only evaluate monotonic coordinate fields. You can always first remap
> your coordinates using some projection which puts the ill-behaved
> parts (nominally, the poles) far away, and preserves distance
> locally. For example, if you have a small field (a degree or so) near
> the pole, this would be a nice way of solving the converging longitude
> lines issues. But generally? Sounds tough.
>
> JD

How about if it was done in 3D? Instead of 2D angular coordinates, use the 3D coordinates of the relevant points on the surface of a unit sphere, and then use HIST_ND to determine which 3D bin the points are in and build the algorithm analogously to MATCH_2D?

The main problem I see is that, for small bin sizes (ie. small desired angular separations), there's a lot of wasted memory storing the histogram in locations that don't lie on the surface of the sphere and therefore are necessarily zero. But maybe there's a way of enumerating the bins that do contain part of the surface - if so, then you could

use that enumeration to map the 3D positions into a simple number that
you can run HISTOGRAM on.

-Jeremy.

Subject: Re: match_2d
Posted by JDS on Wed, 29 Apr 2009 20:29:43 GMT
View Forum Message <> Reply to Message

On Apr 28, 10:44 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 27, 3:06 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>> I'm pretty sure there's a HIST_ND-based algorithm of doing this
>>> similar toMATCH_2Dbut taking spherical trig into account, but I
>>> don't have the patience to figure it out.
>
>> That would be challenging for the whole sphere, since histogram can
>> only evaluate monotonic coordinate fields. You can always first remap
>> your coordinates using some projection which puts the ill-behaved
>> parts (nominally, the poles) far away, and preserves distance
>> locally. For example, if you have a small field (a degree or so) near
>> the pole, this would be a nice way of solving the converging longitude
>> lines issues. But generally? Sounds tough.
>
>> JD
>
> How about if it was done in 3D? Instead of 2D angular coordinates, use
> the 3D coordinates of the relevant points on the surface of a unit
> sphere, and then use HIST_ND to determine which 3D bin the points are
> in and build the algorithm analogously to MATCH_2D?
>
> The main problem I see is that, for small bin sizes (ie. small desired
> angular separations), there's a lot of wasted memory storing the
> histogram in locations that don't lie on the surface of the sphere and
> therefore are necessarily zero. But maybe there's a way of enumerating
> the bins that do contain part of the surface - if so, then you could
> use that enumeration to map the 3D positions into a simple number that
> you can run HISTOGRAM on.

I thought of that and rejected it for the reason you mention. The
vast majority of memory would be devoted to empty volume, and as the
resolution grew, the fraction of wasted memory would grow as well.
The mapping you describe to do away with the empty space is equivalent
to spherical projection, for which there is no unique mapping for the
whole sphere. One possibility would be to project iteratively,
forming low distortion projections over the sphere to push the poles
off out of the way, matching against a subset of the data, rotate the

projection, repeat. Some heuristic for deciding which projection, how large, and where to center it, would be needed.

At some point, it would become simpler to use pattern matching via Delauney triangulation or other patterns formed from the target list.

JD

Subject: Re: match_2d
Posted by Jeremy Bailin on Wed, 29 Apr 2009 23:30:02 GMT
View Forum Message <> Reply to Message

On Apr 29, 4:29 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
> On Apr 28, 10:44 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Apr 27, 3:06 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>>> I'm pretty sure there's a HIST_ND-based algorithm of doing this
>>>> similar toMATCH_2Dbut taking spherical trig into account, but I
>>>> don't have the patience to figure it out.
>
>>> That would be challenging for the whole sphere, since histogram can
>>> only evaluate monotonic coordinate fields. You can always first remap
>>> your coordinates using some projection which puts the ill-behaved
>>> parts (nominally, the poles) far away, and preserves distance
>>> locally. For example, if you have a small field (a degree or so) near
>>> the pole, this would be a nice way of solving the converging longitude
>>> lines issues. But generally? Sounds tough.
>
>>> JD
>
>> How about if it was done in 3D? Instead of 2D angular coordinates, use
>> the 3D coordinates of the relevant points on the surface of a unit
>> sphere, and then use HIST_ND to determine which 3D bin the points are
>> in and build the algorithm analogously to MATCH_2D?
>
>> The main problem I see is that, for small bin sizes (ie. small desired
>> angular separations), there's a lot of wasted memory storing the
>> histogram in locations that don't lie on the surface of the sphere and
>> therefore are necessarily zero. But maybe there's a way of enumerating
>> the bins that do contain part of the surface - if so, then you could
>> use that enumeration to map the 3D positions into a simple number that
>> you can run HISTOGRAM on.
>
> I thought of that and rejected it for the reason you mention. The

> vast majority of memory would be devoted to empty volume, and as the
> resolution grew, the fraction of wasted memory would grow as well.
> The mapping you describe to do away with the empty space is equivalent
> to spherical projection, for which there is no unique mapping for the
> whole sphere. One possibility would be to project iteratively,
> forming low distortion projections over the sphere to push the poles
> off out of the way, matching against a subset of the data, rotate the
> projection, repeat. Some heuristic for deciding which projection, how
> large, and where to center it, would be needed.
>
> At some point, it would become simpler to use pattern matching via
> Delauney triangulation or other patterns formed from the target list.
>
> JD

One idea to at least limit the amount of wasted memory is to use a larger grid spacing than absolutely required - the match_2d algorithm needs grid spacings that are no smaller than 2x the desired separation, but I think it should work fine if the spacing is larger... the drawback would be that you need to calculate the correct angular distance for a larger number of particles than strictly necessary, but that would be a worthwhile tradeoff at some point.

But I think that there should be an enumeration mapping solution. There certainly exists an enumeration for any grid size... I can generate one by placing points randomly on the surface of the sphere, calculating the 3D histogram, and then getting a list of which cells contain points - the enumeration is then simply be the ordinal of the cell within the list. But that's a stupid solution in this case, because the entire point is to avoid calculating the full 3D histogram. Still, the fact that an enumeration is possible makes me think that it should be possible to generate it from first principles rather than empirically. :-)=

-Jeremy.

Subject: Re: match_2d
Posted by Jeremy Bailin on Thu, 30 Apr 2009 00:00:30 GMT
View Forum Message <> Reply to Message

On Apr 29, 7:30 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 29, 4:29 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>
>
>> On Apr 28, 10:44 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>

>>> On Apr 27, 3:06 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>>> > I'm pretty sure there's a HIST_ND-based algorithm of doing this
>>>> > similar toMATCH_2Dbut taking spherical trig into account, but I
>>>> > don't have the patience to figure it out.
>
>>>> That would be challenging for the whole sphere, since histogram can
>>>> only evaluate monotonic coordinate fields. You can always first remap
>>>> your coordinates using some projection which puts the ill-behaved
>>>> parts (nominally, the poles) far away, and preserves distance
>>>> locally. For example, if you have a small field (a degree or so) near
>>>> the pole, this would be a nice way of solving the converging longitude
>>>> lines issues. But generally? Sounds tough.
>
>>>> JD
>
>>> How about if it was done in 3D? Instead of 2D angular coordinates, use
>>> the 3D coordinates of the relevant points on the surface of a unit
>>> sphere, and then use HIST_ND to determine which 3D bin the points are
>>> in and build the algorithm analogously to MATCH_2D?
>
>>> The main problem I see is that, for small bin sizes (ie. small desired
>>> angular separations), there's a lot of wasted memory storing the
>>> histogram in locations that don't lie on the surface of the sphere and
>>> therefore are necessarily zero. But maybe there's a way of enumerating
>>> the bins that do contain part of the surface - if so, then you could
>>> use that enumeration to map the 3D positions into a simple number that
>>> you can run HISTOGRAM on.
>
>> I thought of that and rejected it for the reason you mention. The
>> vast majority of memory would be devoted to empty volume, and as the
>> resolution grew, the fraction of wasted memory would grow as well.
>> The mapping you describe to do away with the empty space is equivalent
>> to spherical projection, for which there is no unique mapping for the
>> whole sphere. One possibility would be to project iteratively,
>> forming low distortion projections over the sphere to push the poles
>> off out of the way, matching against a subset of the data, rotate the
>> projection, repeat. Some heuristic for deciding which projection, how
>> large, and where to center it, would be needed.
>
>> At some point, it would become simpler to use pattern matching via
>> Delauney triangulation or other patterns formed from the target list.
>
>> JD
>
> One idea to at least limit the amount of wasted memory is to use a
> larger grid spacing than absolutely required - the match_2d algorithm
> needs grid spacings that are no smaller than 2x the desired

> separation, but I think it should work fine if the spacing is
> larger... the drawback would be that you need to calculate the correct
> angular distance for a larger number of particles than strictly
> necessary, but that would be a worthwhile tradeoff at some point.
>
> But I think that there should be an enumeration mapping solution.
> There certainly exists an enumeration for any grid size... I can
> generate one by placing points randomly on the surface of the sphere,
> calculating the 3D histogram, and then getting a list of which cells
> contain points - the enumeration is then simply be the ordinal of the
> cell within the list. But that's a stupid solution in this case,
> because the entire point is to avoid calculating the full 3D
> histogram. Still, the fact that an enumeration is possible makes me
> think that it should be possible to generate it from first principles
> rather than empirically. :-)=
>
> -Jeremy.

Now that I think about it, you can use the random points directly to
get the grid cells without going via the histogram - just calculate
the bin each one would fall into, and UNIQ them. You just need to
generate enough points that you're virtually guaranteed to get a hit
in each bin. You could guarantee it by also including every neighbour
of any grid cell that contains a point - that way even a cell that
only has a sliver of surface pass through it and therefore does not
contain a point will get into the list.

Hmmm... okay, I'm going to code that up and see if it works.

-Jeremy.

Subject: Re: match_2d
Posted by Jeremy Bailin on Sat, 09 May 2009 05:02:40 GMT
View Forum Message <> Reply to Message

On Apr 29, 8:00 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 29, 7:30 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Apr 29, 4:29 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>> On Apr 28, 10:44 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>>> On Apr 27, 3:06 pm, JDS <jdtsmith.nos...@yahoo.com> wrote:
>
>>>> > > I'm pretty sure there's a HIST_ND-based algorithm of doing this

>>>> > > similar toMATCH_2Dbut taking spherical trig into account, but I
>>>> > > don't have the patience to figure it out.
>
>>>> > That would be challenging for the whole sphere, since histogram can
>>>> > only evaluate monotonic coordinate fields. You can always first remap
>>>> > your coordinates using some projection which puts the ill-behaved
>>>> > parts (nominally, the poles) far away, and preserves distance
>>>> > locally. For example, if you have a small field (a degree or so) near
>>>> > the pole, this would be a nice way of solving the converging longitude
>>>> > lines issues. But generally? Sounds tough.
>
>>>> > JD
>
>>>> How about if it was done in 3D? Instead of 2D angular coordinates, use
>>>> the 3D coordinates of the relevant points on the surface of a unit
>>>> sphere, and then use HIST_ND to determine which 3D bin the points are
>>>> in and build the algorithm analogously toMATCH_2D?
>
>>>> The main problem I see is that, for small bin sizes (ie. small desired
>>>> angular separations), there's a lot of wasted memory storing the
>>>> histogram in locations that don't lie on the surface of the sphere and
>>>> therefore are necessarily zero. But maybe there's a way of enumerating
>>>> the bins that do contain part of the surface - if so, then you could
>>>> use that enumeration to map the 3D positions into a simple number that
>>>> you can run HISTOGRAM on.
>
>>> I thought of that and rejected it for the reason you mention. The
>>> vast majority of memory would be devoted to empty volume, and as the
>>> resolution grew, the fraction of wasted memory would grow as well.
>>> The mapping you describe to do away with the empty space is equivalent
>>> to spherical projection, for which there is no unique mapping for the
>>> whole sphere. One possibility would be to project iteratively,
>>> forming low distortion projections over the sphere to push the poles
>>> off out of the way, matching against a subset of the data, rotate the
>>> projection, repeat. Some heuristic for deciding which projection, how
>>> large, and where to center it, would be needed.
>
>>> At some point, it would become simpler to use pattern matching via
>>> Delauney triangulation or other patterns formed from the target list.
>
>>> JD
>
>> One idea to at least limit the amount of wasted memory is to use a
>> larger grid spacing than absolutely required - thematch_2dalgorithm
>> needs grid spacings that are no smaller than 2x the desired
>> separation, but I think it should work fine if the spacing is
>> larger... the drawback would be that you need to calculate the correct
>> angular distance for a larger number of particles than strictly

>> necessary, but that would be a worthwhile tradeoff at some point.
>
>> But I think that there should be an enumeration mapping solution.
>> There certainly exists an enumeration for any grid size... I can
>> generate one by placing points randomly on the surface of the sphere,
>> calculating the 3D histogram, and then getting a list of which cells
>> contain points - the enumeration is then simply be the ordinal of the
>> cell within the list. But that's a stupid solution in this case,
>> because the entire point is to avoid calculating the full 3D
>> histogram. Still, the fact that an enumeration is possible makes me
>> think that it should be possible to generate it from first principles
>> rather than empirically. :-)=
>
>> -Jeremy.
>
> Now that I think about it, you can use the random points directly to
> get the grid cells without going via the histogram - just calculate
> the bin each one would fall into, and UNIQ them. You just need to
> generate enough points that you're virtually guaranteed to get a hit
> in each bin. You could guarantee it by also including every neighbour
> of any grid cell that contains a point - that way even a cell that
> only has a sliver of surface pass through it and therefore does not
> contain a point will get into the list.
>
> Hmmm... okay, I'm going to code that up and see if it works.
>
> -Jeremy.

Here is my best version so far:

http://www.physics.mcmaster.ca/~bailinj/idl/withinsphrad_vec 3d.pro

It borrows heavily from JD's MATCH_2D, but does it in 3D on the surface of the sphere. It sidesteps all of the "wasted space" and "enumerating the surface of the sphere" issues by enumerating cells that contain points in either the input or target coordinate lists - any other cells on the sphere are irrelevant, since all we need to know about them is that they don't contain any points. So I just make a list of all cells that contain a point in either coordinate list and use the index into that list as my mapping. The histograms then become very efficient! And it doesn't waste extra space generating the enumeration, since we need to calculate those cell locations anyway.

In my simple tests, this does a factor of 100 better than my WITHINSPHRAD_VEC, and about a factor of 10 better than a for loop using the scalar WITHINSPHRAD, but ymmv depending on the number of input and target coordinates, as well as the angular matching radius.

-Jeremy.

Subject: Re: match_2d
Posted by vino on Thu, 06 May 2010 11:27:07 GMT
View Forum Message <> Reply to Message

Can you tell us what your error/problem when using this??
One thing i do is to use reform to make sure x1,y1,x2 and y2 is a 1D
array.

Regards,
Vino

On May 6, 9:29 am, Dave Poreh <d.po...@gmail.com> wrote:
> Hello everyone,
> I have a problem that I can't fix.
> I am using match_2d from J.D.Smith's library to match between two
> catalogues. Below is what i am doing:
> IDL> print, laser
> 13.51132 78.08015 2.244529
> 13.5041 78.07921 2.416421
> 13.49686 78.07828 2.420123
> 13.48957 78.07736 2.668057
> 13.48225 78.07644 2.652986
> 13.4749 78.07554 2.604654
> 13.46749 78.07465 2.373039
> 13.46009 78.07375 2.699916
> ……
> IDL> print, sat
> 13.50834 78.08333 5.965842
> 13.50001 78.08333 3.459386
> 13.50001 78.075 3.092191
> 13.49168 78.075 3.444704
> 13.48334 78.075 3.364458
> 13.47501 78.075 3.108061
> 13.46668 78.075 3.121016
> 13.45834 78.075 3.017546
> 13.45001 78.075 3.15583
> ……
> What I want to do is find the closest data from array laser to each of
> array sat and then average them (of course for some points of sat I do
> not have close laser data). At the end I want to plot sat data and
> averaged laser to comparison.
> x1=laser[0,*]
> y1=laser[1,*]
> x2=sat[0,*]

> y2=sat[1,*]
>
> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)
> Any help highly appreciated
> Cheers
> Dave

Subject: Re: match_2d
Posted by d.poreh on Thu, 06 May 2010 12:25:43 GMT
View Forum Message <> Reply to Message

On May 6, 4:27 am, vino <astrocr...@gmail.com> wrote:
> Can you tell us what your error/problem when using this??
> One thing i do is to use reform to make sure x1,y1,x2 and y2 is a 1D
> array.
>
> Regards,
> Vino
>
> On May 6, 9:29 am, Dave Poreh <d.po...@gmail.com> wrote:
>
>
>
>> Hello everyone,
>> I have a problem that I can't fix.
>> I am using match_2d from J.D.Smith's library to match between two
>> catalogues. Below is what i am doing:
>> IDL> print, laser
>> 13.51132 78.08015 2.244529
>> 13.5041 78.07921 2.416421
>> 13.49686 78.07828 2.420123
>> 13.48957 78.07736 2.668057
>> 13.48225 78.07644 2.652986
>> 13.4749 78.07554 2.604654
>> 13.46749 78.07465 2.373039
>> 13.46009 78.07375 2.699916
>> ……
>> IDL> print, sat
>> 13.50834 78.08333 5.965842
>> 13.50001 78.08333 3.459386
>> 13.50001 78.075 3.092191
>> 13.49168 78.075 3.444704
>> 13.48334 78.075 3.364458
>> 13.47501 78.075 3.108061
>> 13.46668 78.075 3.121016
>> 13.45834 78.075 3.017546
>> 13.45001 78.075 3.15583

>> ……
>> What I want to do is find the closest data from array laser to each of
>> array sat and then average them (of course for some points of sat I do
>> not have close laser data). At the end I want to plot sat data and
>> averaged laser to comparison.
>> x1=laser[0,*]
>> y1=laser[1,*]
>> x2=sat[0,*]
>> y2=sat[1,*]
>
>> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)
>> Any help highly appreciated
>> Cheers
>> Dave

Actually there is no error, but I can't understand the procedure. At
the end output is a 1D array (match). What I want is for each pixel of
SAT array find the closest data from laser array (but I don't know how
to do that).
Cheers
Dave

Subject: Re: match_2d
Posted by wlandsman on Thu, 06 May 2010 18:11:23 GMT
View Forum Message <> Reply to Message

> Actually there is no error, but I can't understand the procedure. At
> the end output is a 1D array (match). What I want is for each pixel of
> SAT array find the closest data from laser array (but I don't know how
> to do that).
> Cheers
> Dave

In your example

match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)

'match' gives the indices of x2,y2 which are the closest match to
x1,y1. So the closest point to x1[0],y1[0] is
x2[match[0]],y2[match[0]].

--Wayne

Subject: Re: match_2d
Posted by d.poreh on Fri, 07 May 2010 12:06:13 GMT

View Forum Message <> Reply to Message

On May 6, 11:11 am, wlandsman <wlands...@gmail.com> wrote:
>> Actually there is no error, but I can't understand the procedure. At
>> the end output is a 1D array (match). What I want is for each pixel of
>> SAT array find the closest data from laser array (but I don't know how
>> to do that).
>> Cheers
>> Dave
>
> In your example
>
> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)
>
> 'match' gives the indices of x2,y2 which are the closest match to
> x1,y1. So the closest point to x1[0],y1[0] is
> x2[match[0]],y2[match[0]].
>
> --Wayne

Thank you. Another question: how could I find the other points of x2-y2 that surrounded in the search area (in this example 0.008) because maybe there is more than 1 point.
Cheers
Dave

Subject: Re: match_2d
Posted by Jeremy Bailin on Fri, 07 May 2010 12:30:04 GMT
View Forum Message <> Reply to Message

On May 7, 8:06 am, Dave Poreh <d.po...@gmail.com> wrote:
> On May 6, 11:11 am, wlandsman <wlands...@gmail.com> wrote:
>
>
>
>
>
>>> Actually there is no error, but I can't understand the procedure. At
>>> the end output is a 1D array (match). What I want is for each pixel of
>>> SAT array find the closest data from laser array (but I don't know how
>>> to do that).
>>> Cheers
>>> Dave
>
>> In your example
>
>> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)

>
>> 'match' gives the indices of x2,y2 which are the closest match to
>> x1,y1. So the closest point to x1[0],y1[0] is
>> x2[match[0]],y2[match[0]].
>
>> --Wayne
>
> Thank you. Another question: how could I find the other points of x2-
> y2 that surrounded in the search area (in this example 0.008) because
> maybe there is more than 1 point.
> Cheers
> Dave

You use a routine that I coincidentally wrote yesterday:
matchall_2d.pro, which is based heavily on JD's code (see below - I
haven't yet put it up in JBIU but I'll do it soon). Note that if your
coordinates are things like lat, lon and cover a larger area, 2D
distances won't be correct and you should use
withinsphrad_vec3d.pro... which is also apparently not yet in JBIU.
Jeez, I'm slacking. If you need it, ask me and I'll send it... and it
will hopefully be online soon too.

```
;+
; NAME:
;    MATCHALL_2D
;
; PURPOSE:
;    Determines which of a set of 2D coordinates are a given distance
from
;    each of a vector of points. Based on JD's MATCH_2D and my
WITHINSPHRAD_VEC3D
;    (in fact, it's basically WITHINSPHRAD_VEC3D tuned back down to a
;    Euclidean surface).
;
; CATEGORY:
;    Astro
;
; CALLING SEQUENCE:
;    Result = MATCHALL_2D(X1, Y1, X2, Y2, Distance, Nwithin)
;
; INPUTS:
;    X1:     Vector of X coordinates.
;
;    Y1:     Vector of Y coordinates.
;
;    X2:     Vector of X coordinates.
;
```

```
;    Y2:     Vector of Y coordinates.
;
;    Distance:  Maximum distance.
;
; OUTPUTS:
;    The function returns the list of indices of X2, Y2 that lie
within
;    Sphrad of each point X1,Y1. The format of the returned array is
;    similar to the REVERSE_INDICES array from HISTOGRAM: the indices
;    into X2,Y2 that are close enough to element i of X1,Y1 are
;    contained in Result[Result[i]:Result[i+1]-1] (note, however, that
;    these indices are not guaranteed to be sorted). If there are no
matches,
;    then Result[i] eq Result[i+1].
;
; OPTIONAL OUTPUTS:
;    Nwithin: A vector containing the number of matches for each of
X1,Y1.
;
; EXAMPLE:
;    Note that the routine is similar to finding
;      WHERE( (X2-X1[i])^2 + (Y2-Y1[i])^2 LE Distance^2, Nwithin)
;    for each element of X1 and Y1, but is much more efficient.
;
;    Shows which random points are within 0.1 of various coordinates:
;     FIXME
;
;    seed=43
;    nrandcoords = 5000l
;    xrand = 2. * RANDOMU(seed, nrandcoords) - 1.
;    yrand = 2. * RANDOMU(seed, nrandcoords) - 1.
;    xcoords = [0.25, 0.5, 0.75]
;    ycoords = [0.75, 0.5, 0.25]
;    ncoords = N_ELEMENTS(xcoords)
;    matches = MATCHALL_2D(xcoords, ycoords, xrand, yrand, 0.1,
nmatches)
;    PLOT, /ISO, PSYM=3, xrand, yrand
;    OPLOT, PSYM=1, COLOR=FSC_COLOR('blue'), xcoords, ycoords
;    OPLOT, PSYM=3, COLOR=FSC_COLOR('red'), xrand[matches[ncoords
+1:*]], $
;      yrand[matches[ncoords+1:*]]
;
; MODIFICATION HISTORY:
;    Written by:    Jeremy Bailin
;    10 June 2008   Public release in JBIU as WITHINSPHRAD
;    24 April 2009  Vectorized as WITHINSPHRAD_VEC
;    25 April 2009  Polished to improve memory use
;    9 May 2009     Radical efficiency re-write as WITHINSPHRAD_VEC3D
```

```
borrowing
;                heavily from JD Smith's MATCH_2D
;    13 May 2009    Removed * from LHS index in final remapping for
speed
;    6 May 2010     Changed to MATCHALL_2D and just using Euclidean 2D
coordinates
;                (add a bunch of stuff back in from MATCH_2D and
take out a bunch
;                of angle stuff)
;-
function matchall_2d, x1, y1, x2, y2, distance, nwithin

if n_elements(x2) ne n_elements(y2) then $
  message, 'X2 and Y2 must have the same number of elements.'
if n_elements(x1) ne n_elements(y1) then $
  message, 'X1 and Y1 must have the same number of elements.'
if n_elements(distance) ne 1 then $
  message, 'Distance must contain one element.'

n1 = n_elements(x1)
n2 = n_elements(x2)

gridlen = 2.*distance
mx=[max(x2,min=mnx2),max(y2,min=mny2)]
mn=[mnx2,mny2]
mn-=1.5*gridlen
mx+=1.5*gridlen

h =  hist_nd([1#x2,1#y2],gridlen,reverse_indices=ri,min=mn,max=mx )
d = size(h,/dimen)

; bin locations of 1 in the 2 grid
xoff = 0. > (x1-mn[0])/gridlen[0] < (d[0]-1.)
yoff = 0. > (y1-mn[1])/(n_elements(gridlen) gt 1?gridlen[1]:gridlen) <
(d[1]-1.)
xbin = floor(xoff) & ybin=floor(yoff)
bin = xbin + d[0]*ybin   ; 1D index

; search 4 bins for closets match - check which quadrant
xoff = 1 - 2*((xoff-xbin) lt 0.5)
yoff = 1 - 2*((yoff-ybin) lt 0.5)

rad2 = distance^2

; loop through all neighbouring cells in correct order
for xi=0,1 do begin
  for yi=0,1 do begin
    b = 0l > (bin + xi*xoff + yi*yoff*d[0]) < (d[0]*d[1]-1)
```

```
    ; dual histogram method, loop by count in search bins (see JD's
code)
    h2 = histogram(h[b], omin=om, reverse_indices=ri2)

    ; loop through repeat counts
    for k=long(om eq 0), n_elements(h2)-1 do if h2[k] gt 0 then begin
      these_bins = ri2[ri2[k]:ri2[k+1]-1]

      if k+om eq 1 then begin ; single point
        these_points = ri[ri[b[these_bins]]]
      endif else begin
        targ=[h2[k],k+om]
        these_points = ri[ri[rebin(b[these_bins],targ,/sample)]+ $
          rebin(lindgen(1,k+om),targ,/sample)]
        these_bins = rebin(temporary(these_bins),targ,/sample)
      endelse

      ; figure out which ones are really within
      within = where( (x2[these_points]-x1[these_bins])^2 +
(y2[these_points] - $
        y1[these_bins])^2 le rad2, nwithin)

      if nwithin gt 0 then begin
        ; have there been any pairs yet?
        if n_elements(plausible) eq 0 then begin
          plausible = [[these_bins[within]],[these_points[within]]]
        endif else begin
          ; concatenation is inefficient, but we do it at most 4 x N1
times
          plausible = [plausible,[[these_bins[within]],
[these_points[within]]]]
        endelse
      endif

    endif
  endfor
endfor

if n_elements(plausible) eq 0 then begin
  nwithin=replicate(0l,n1)
  return, replicate(-1,n1+1)
endif else begin
  ; use histogram to generate a reverse_indices array that contains
  ; the relevant entries, and then map into the appropriate elements
  ; in 2
  nwithin = histogram(plausible[*,0], min=0, max=n1-1,
reverse_indices=npri)
```

```
  npri[n1+1] = plausible[npri[n1+1:*],1]
  return, npri
endelse

end
```

Subject: Re: match_2d
Posted by d.poreh on Fri, 07 May 2010 12:55:06 GMT
View Forum Message <> Reply to Message

On May 7, 5:30 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> On May 7, 8:06 am, Dave Poreh <d.po...@gmail.com> wrote:
>
>
>
>
>
>> On May 6, 11:11 am, wlandsman <wlands...@gmail.com> wrote:
>
>>>> Actually there is no error, but I can't understand the procedure. At
>>>> the end output is a 1D array (match). What I want is for each pixel of
>>>> SAT array find the closest data from laser array (but I don't know how
>>>> to do that).
>>>> Cheers
>>>> Dave
>
>>> In your example
>
>>> match=match_2d(x1,y1,x2,y2,0.008,MATCH_DISTANCE=md)
>
>>> 'match' gives the indices of x2,y2 which are the closest match to
>>> x1,y1. So the closest point to x1[0],y1[0] is
>>> x2[match[0]],y2[match[0]].
>
>>> --Wayne
>
>> Thank you. Another question: how could I find the other points of x2-
>> y2 that surrounded in the search area (in this example 0.008) because
>> maybe there is more than 1 point.
>> Cheers
>> Dave
>
> You use a routine that I coincidentally wrote yesterday:
> matchall_2d.pro, which is based heavily on JD's code (see below - I
> haven't yet put it up in JBIU but I'll do it soon). Note that if your
> coordinates are things like lat, lon and cover a larger area, 2D
> distances won't be correct and you should use

> withinsphrad_vec3d.pro... which is also apparently not yet in JBIU.
> Jeez, I'm slacking. If you need it, ask me and I'll send it... and it
> will hopefully be online soon too.
>
> ;+
> ; NAME:
> ;   MATCHALL_2D
> ;
> ; PURPOSE:
> ;   Determines which of a set of 2D coordinates are a given distance
> from
> ;   each of a vector of points. Based on JD's MATCH_2D and my
> WITHINSPHRAD_VEC3D
> ;   (in fact, it's basically WITHINSPHRAD_VEC3D tuned back down to a
> ;   Euclidean surface).
> ;
> ; CATEGORY:
> ;   Astro
> ;
> ; CALLING SEQUENCE:
> ;   Result = MATCHALL_2D(X1, Y1, X2, Y2, Distance, Nwithin)
> ;
> ; INPUTS:
> ;   X1:   Vector of X coordinates.
> ;
> ;   Y1:   Vector of Y coordinates.
> ;
> ;   X2:   Vector of X coordinates.
> ;
> ;   Y2:   Vector of Y coordinates.
> ;
> ;   Distance: Maximum distance.
> ;
> ; OUTPUTS:
> ;   The function returns the list of indices of X2, Y2 that lie
> within
> ;   Sphrad of each point X1,Y1. The format of the returned array is
> ;   similar to the REVERSE_INDICES array from HISTOGRAM: the indices
> ;   into X2,Y2 that are close enough to element i of X1,Y1 are
> ;   contained in Result[Result[i]:Result[i+1]-1] (note, however, that
> ;   these indices are not guaranteed to be sorted). If there are no
> matches,
> ;   then Result[i] eq Result[i+1].
> ;
> ; OPTIONAL OUTPUTS:
> ;   Nwithin: A vector containing the number of matches for each of
> X1,Y1.
> ;

```
>  ; EXAMPLE:
>  ;    Note that the routine is similar to finding
>  ;      WHERE( (X2-X1[i])^2 + (Y2-Y1[i])^2 LE Distance^2, Nwithin)
>  ;    for each element of X1 and Y1, but is much more efficient.
>  ;
>  ;    Shows which random points are within 0.1 of various coordinates:
>  ;     FIXME
>  ;
>  ;    seed=43
>  ;    nrandcoords = 5000l
>  ;    xrand = 2. * RANDOMU(seed, nrandcoords) - 1.
>  ;    yrand = 2. * RANDOMU(seed, nrandcoords) - 1.
>  ;    xcoords = [0.25, 0.5, 0.75]
>  ;    ycoords = [0.75, 0.5, 0.25]
>  ;    ncoords = N_ELEMENTS(xcoords)
>  ;    matches = MATCHALL_2D(xcoords, ycoords, xrand, yrand, 0.1,
>  nmatches)
>  ;    PLOT, /ISO, PSYM=3, xrand, yrand
>  ;    OPLOT, PSYM=1, COLOR=FSC_COLOR('blue'), xcoords, ycoords
>  ;    OPLOT, PSYM=3, COLOR=FSC_COLOR('red'), xrand[matches[ncoords
>  +1:*]], $
>  ;      yrand[matches[ncoords+1:*]]
>  ;
>  ; MODIFICATION HISTORY:
>  ;    Written by:    Jeremy Bailin
>  ;    10 June 2008   Public release in JBIU as WITHINSPHRAD
>  ;    24 April 2009  Vectorized as WITHINSPHRAD_VEC
>  ;    25 April 2009  Polished to improve memory use
>  ;    9 May 2009     Radical efficiency re-write as WITHINSPHRAD_VEC3D
>  borrowing
>  ;                   heavily from JD Smith's MATCH_2D
>  ;    13 May 2009    Removed * from LHS index in final remapping for
>  speed
>  ;    6 May 2010     Changed to MATCHALL_2D and just using Euclidean 2D
>  coordinates
>  ;                   (add a bunch of stuff back in from MATCH_2D and
>  take out a bunch
>  ;                   of angle stuff)
>  ;-
>  function matchall_2d, x1, y1, x2, y2, distance, nwithin
>
>  if n_elements(x2) ne n_elements(y2) then $
>    message, 'X2 and Y2 must have the same number of elements.'
>  if n_elements(x1) ne n_elements(y1) then $
>    message, 'X1 and Y1 must have the same number of elements.'
>  if n_elements(distance) ne 1 then $
>    message, 'Distance must contain one element.'
>
```

```
>  n1 = n_elements(x1)
>  n2 = n_elements(x2)
>
>  gridlen = 2.*distance
>  mx=[max(x2,min=mnx2),max(y2,min=mny2)]
>  mn=[mnx2,mny2]
>  mn-=1.5*gridlen
>  mx+=1.5*gridlen
>
>  h =  hist_nd([1#x2,1#y2],gridlen,reverse_indices=ri,min=mn,max=mx )
>  d = size(h,/dimen)
>
>  ; bin locations of 1 in the 2 grid
>  xoff = 0. > (x1-mn[0])/gridlen[0] < (d[0]-1.)
>  yoff = 0. > (y1-mn[1])/(n_elements(gridlen) gt 1?gridlen[1]:gridlen) <
>  (d[1]-1.)
>  xbin = floor(xoff) & ybin=floor(yoff)
>  bin = xbin + d[0]*ybin   ; 1D index
>
>  ; search 4 bins for closets match - check which quadrant
>  xoff = 1 - 2*((xoff-xbin) lt 0.5)
>  yoff = 1 - 2*((yoff-ybin) lt 0.5)
>
>  rad2 = distance^2
>
>  ; loop through all neighbouring cells in correct order
>  for xi=0,1 do begin
>    for yi=0,1 do begin
>      b = 0l > (bin + xi*xoff + yi*yoff*d[0]) < (d[0]*d[1]-1)
>
>      ; dual histogram method, loop by count in search bins (see JD's
>  code)
>      h2 = histogram(h[b], omin=om, reverse_indices=ri2)
>
>      ; loop through repeat counts
>      for k=long(om eq 0), n_elements(h2)-1 do if h2[k] gt 0 then begin
>        these_bins = ri2[ri2[k]:ri2[k+1]-1]
>
>        if k+om eq 1 then begin ; single point
>          these_points = ri[ri[b[these_bins]]]
>        endif else begin
>          targ=[h2[k],k+om]
>          these_points = ri[ri[rebin(b[these_bins],targ,/sample)]+ $
>            rebin(lindgen(1,k+om),targ,/sample)]
>          these_bins = rebin(temporary(these_bins),targ,/sample)
>        endelse
>
>        ; figure out which ones are really within
```

```
>       within = where( (x2[these_points]-x1[these_bins])^2 +
>  (y2[these_points] - $
>         y1[these_bins])^2 le rad2, nwithin)
>
>       if nwithin gt 0 then begin
>         ; have there been any pairs yet?
>         if n_elements(plausible) eq 0 then begin
>           plausible = [[these_bins[within]],[these_points[within]]]
>         endif else begin
>           ; concatenation is inefficient, but we do it at most 4 x N1
>  times
>           plausible = [plausible,[[these_bins[within]],
>  [these_points[within]]]]
>         endelse
>       endif
>
>     endif
>   endfor
>  endfor
>
>  if n_elements(plausible) eq 0 then begin
>    nwithin=replicate(0l,n1)
>    return, replicate(-1,n1+1)
>  endif else begin
>    ; use histogram to generate a reverse_indices array that contains
>    ; the relevant entries, and then map into the appropriate elements
>    ; in 2
>    nwithin = histogram(plausible[*,0], min=0, max=n1-1,
>  reverse_indices=npri)
>    npri[n1+1] = plausible[npri[n1+1:*],1]
>    return, npri
>  endelse
>
>  end
```

Works like a charm! Yes please send me (withinsphrad_vec3d.pro) and thank you again.
Cheers
Dave
