Subject: Re: match_2d
Posted by Jeremy Bailin on Tue, 21 Apr 2009 15:47:01 GMT
View Forum Message <> Reply to Message

On Apr 20, 9:40 am, vino <astrocr...@gmail.com> wrote:
> 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

It looks like MATCH_2D requires that its inputs be flat vectors, but
instead you're feeding it [1,N] arrays.

-Jeremy.

Subject: Re: match_2d
Posted by vino on Tue, 21 Apr 2009 16:23:11 GMT
View Forum Message <> Reply to Message

Hi Jerely Ballin,

I even tried reforming them before input like this :

DL> match=match_2d(reform(star_radec1(0,*)),reform(star_radec1
(1,*)),reform(star_radec2
(0,*)),reform(star_radec2(1,*)),.02,MATCH_DISTANCE=md)

Even this doesnt seem to work.... :(

Eventhough in this eg, we need to do a one-to-one match, in my
original data, it has to be one-to-many as well...Will the routine
work for that as well??

Thanks and regards,

Vino

On Apr 21, 4:47 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Apr 20, 9:40 am, vino <astrocr...@gmail.com> wrote:
>
>
>
>> 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
>
> It looks like MATCH_2D requires that its inputs be flat vectors, but
> instead you're feeding it [1,N] arrays.
>
> -Jeremy.

Subject: Re: match_2d
Posted by Jeremy Bailin on Wed, 22 Apr 2009 17:22:15 GMT
View Forum Message <> Reply to Message

On Apr 21, 12:23 pm, vino <astrocr...@gmail.com> wrote:
> Hi Jerely Ballin,
>
> I even tried reforming them before input like this :
>
> DL> match=match_2d(reform(star_radec1(0,*)),reform(star_radec1
> (1,*)),reform(star_radec2
> (0,*)),reform(star_radec2(1,*)),.02,MATCH_DISTANCE=md)
>
> Even this doesnt seem to work.... :(
>
> Eventhough in this eg, we need to do a one-to-one match, in my
> original data, it has to be one-to-many as well...Will the routine
> work for that as well??
>
> Thanks and regards,
>
> Vino
>
> On Apr 21, 4:47 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>> On Apr 20, 9:40 am, vino <astrocr...@gmail.com> wrote:
>
>>> 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
>
>> It looks like MATCH_2D requires that its inputs be flat vectors, but
>> instead you're feeding it [1,N] arrays.
>
>> -Jeremy.

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:

```
minra = min(star_radec2[0,*])
mindec = min(star_radec2[1,*])
match = match_2d(star_radec1[0,*]-minra, star_radec1[1,*]-mindec,
star_radec2[0,*]-minra, $
  star_radec2[1,*]-mindec, 0.02, match_distance=md)

IDL> print, match
        -1          0          1         -1
IDL> print, md
  1.53332e-19      0.00000      0.00000  5.90697e-22
```

Also, it looks like it will only return 1 match per input coordinate.
If you need "all matches within a given distance", take a look at
WITHINSPHRAD (http://web.astroconst.org/jbiu/jbiu-doc/astro/
withinsphrad.html) from JBIU (http://web.astroconst.org/jbiu).

-Jeremy.

Subject: Re: match_2d
Posted by JDS on Wed, 22 Apr 2009 22:39:08 GMT
View Forum Message <> Reply to Message

>
> 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

Subject: Re: match_2d
Posted by vino on Thu, 23 Apr 2009 12:10:57 GMT
View Forum Message <> Reply to Message

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

Subject: Re: match_2d
Posted by Jeremy Bailin on Fri, 24 Apr 2009 17:50:38 GMT
View Forum Message <> Reply to Message

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.
