
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.
