Subject: Re: position matching
Posted by Paolo Grigis on Tue, 15 May 2007 15:14:33 GMT
View Forum Message <> Reply to Message

cmancone@ufl.edu wrote:
> Yes, I read that article. However, it doesn't quite translate well
> into what I need. It presents two methods, one using arrays, the
> other using a Delaunay triangulation (DT). For my purposes (20,000
> stars) the array method won't work - it requires way too much memory
> (I pondered a similar solution myself).

Ok, I guess I misread your problem... but what if you divide up the first
list in, say, 20 chunks of 1000 stars each and use the array method on
each chunk (such that you get a 1000 by 20'000 array) separately?
You can do that since you want the minimum distance of each star in the
first list from all the stars in the 2nd, right? So it doesn't matter how
many stars in the first list you process each time.

Ciao,
Paolo

That leaves the DT method.
> There's two problems with this. First, I don't just need the closest
> neighbor, I need the closest neighbor within a certain distance.
> Persumably, this is easily solved with a properly placed WHERE or IF
> statement. The bigger problem, however, is that I am matching up two
> separate lists, and I can't have stars on one list matching up stars
> on the same list. The DT doesn't make any distinction between stars
> as far as I can tell. You give it one combined list, and it finds the
> closest stars no matter where they come from, which is a problem.
> I've been trying to figure out just how the DT works, so I can
> determine if it is possible to disentangle the two star lists or not.
> It's a bit confusing though, and I have yet to determine if it will
> work for my purposes.
>
> On May 15, 10:14 am, Paolo Grigis <pgri...@astro.phys.ethz.ch> wrote:
>> Seems to be a recurring theme... here's a nice article:
>>
>> http://www.dfanning.com/code_tips/slowloops.html
>>
>> Ciao,
>> Paolo
>>
>> cmanc...@ufl.edu wrote:
>>> Hi everyone,
>>> A common task I have to do is take two lists of stars with x & y
>>> positions and match up the closest stars within a certain radius (so

>>> that each star has at most one match, that one being the best match).
>>> A long time ago I wrote some code to do this that gets the job done,
>>> but probably not in the fastest way. It just uses a for loop over one
>>> of the lists and uses a where to search for the closest star to each
>>> star on the other list. Most of the time this is more than adequate,
>>> but anytime my star lists get around 10000-20000 stars each (which
>>> happens on a not-so irregular basis) the program turns into quite a
>>> beast and takes its sweet time (i.e. a minute or two). Granted, this
>>> isn't exactly research-stopping time delays, but I'm sure that with a
>>> well thought-out algorithm, the execution time could be pulled down to
>>> a handful of seconds. The problem is, I have yet to come up with a
>>> well thought-out algorithm. I'm sure I'm not the only one who has run
>>> into this, so I was hoping there might be someone else out there that
>>> has dealt with the same thing, and knows a better way.
>>> -Conor
>
>