Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 15:15:22 GMT
View Forum Message <> Reply to Message

On Jul 30, 10:01 am, Gray <graylikethec...@gmail.com> wrote:
> Hi all,
>
> For quite a while I've been using JD Smith's match_2d routine to match
> xy coords between lists. However, this and all the other matching
> codes I've seen out there suffer from a variation of the uniqueness of
> matches problem.
>
> Codes like SRCOR in the NASA IDL library let you specify a one-to-one
> match, i.e. enforcing that each element in list 2 only be matched to
> one element in list 1; using match_2d's match_distance keyword one
> could implement the same effect oneself. However, while that excludes
> multiple matches to the same element, it's all done after the fact,
> after the original match was determined.
>
> What I'm looking for is an algorithm that matches 2 lists, identifies
> multiple-matches, and then looks for additional matches within the
> search radius for elements which would become unmatched after
> enforcing a one-to-one relationship. What I mean is, say element 0 in
> list 2 is matched to both element 3 and element 5 in list 1, and that
> the distance between 2_0 and 1_3 is smaller than the distance between
> 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
> there is element 2_1 which is also within the search radius of 1_5?
> Then, 1_5 should be re-matched with 2_1.
>
> My best idea thus far is to run match_2d once, identify multiple-
> matches, keep the matches with minimum distance using match_distance,
> then iterate with the remaining elements until match_2d returns no
> matches. Can anyone come up with a better solution?

Hmmm... what about starting with first point (a) in list 1, finding
the nearest
point (b) to (a) in list 2, removing (b) from list 2 and repeat for
all points
in list 1? [this assumes list 1 and list 2 have the same number of
elements N,
which is a necessary condition for a one-to-one matching].

With some smart partitioning of list 1 it will take ~log(N) to find
the nearest
point, so we are looking at ~ N log(N) operations...

Ciao,
Paolo

>
> --Gray

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 15:23:51 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
> On Jul 30, 10:01 am, Gray <graylikethec...@gmail.com> wrote:
>
>
>
>
>
>
>> Hi all,
>
>> For quite a while I've been using JD Smith's match_2d routine to match
>> xy coords between lists. However, this and all the other matching
>> codes I've seen out there suffer from a variation of the uniqueness of
>> matches problem.
>
>> Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>> match, i.e. enforcing that each element in list 2 only be matched to
>> one element in list 1; using match_2d's match_distance keyword one
>> could implement the same effect oneself. However, while that excludes
>> multiple matches to the same element, it's all done after the fact,
>> after the original match was determined.
>
>> What I'm looking for is an algorithm that matches 2 lists, identifies
>> multiple-matches, and then looks for additional matches within the
>> search radius for elements which would become unmatched after
>> enforcing a one-to-one relationship. What I mean is, say element 0 in
>> list 2 is matched to both element 3 and element 5 in list 1, and that
>> the distance between 2_0 and 1_3 is smaller than the distance between
>> 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>> there is element 2_1 which is also within the search radius of 1_5?
>> Then, 1_5 should be re-matched with 2_1.
>
>> My best idea thus far is to run match_2d once, identify multiple-
>> matches, keep the matches with minimum distance using match_distance,
>> then iterate with the remaining elements until match_2d returns no
>> matches. Can anyone come up with a better solution?
>
> Hmmm... what about starting with first point (a) in list 1, finding

> the nearest
> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
> all points
> in list 1? [this assumes list 1 and list 2 have the same number of
> elements N,
> which is a necessary condition for a one-to-one matching].
>
> With some smart partitioning of list 1 it will take ~log(N) to find
> the nearest
> point, so we are looking at ~ N log(N) operations...
>
> Ciao,
> Paolo
>
>
>
>
>
>> --Gray

I'm fine with having there be points which don't match at all w/in the search radius, I'm just looking to force any matches that exist to be recognized.

The straight FOR-loop method is certainly serviceable, but I had hoped there was a more efficient way to do it... but it's certainly possible (or even likely) that anything fancier I try to do is LESS efficient.

--Gray

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 15:25:15 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>
>
>
>
>> On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>> Hi all,
>
>>> For quite a while I've been using JD Smith's match_2d routine to match
>>> xy coords between lists. However, this and all the other matching

>>> codes I've seen out there suffer from a variation of the uniqueness of
>>> matches problem.
>
>>> Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>> match, i.e. enforcing that each element in list 2 only be matched to
>>> one element in list 1; using match_2d's match_distance keyword one
>>> could implement the same effect oneself. However, while that excludes
>>> multiple matches to the same element, it's all done after the fact,
>>> after the original match was determined.
>
>>> What I'm looking for is an algorithm that matches 2 lists, identifies
>>> multiple-matches, and then looks for additional matches within the
>>> search radius for elements which would become unmatched after
>>> enforcing a one-to-one relationship. What I mean is, say element 0 in
>>> list 2 is matched to both element 3 and element 5 in list 1, and that
>>> the distance between 2_0 and 1_3 is smaller than the distance between
>>> 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>> there is element 2_1 which is also within the search radius of 1_5?
>>> Then, 1_5 should be re-matched with 2_1.
>
>>> My best idea thus far is to run match_2d once, identify multiple-
>>> matches, keep the matches with minimum distance using match_distance,
>>> then iterate with the remaining elements until match_2d returns no
>>> matches. Can anyone come up with a better solution?
>
>> Hmmm... what about starting with first point (a) in list 1, finding
>> the nearest
>> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>> all points
>> in list 1? [this assumes list 1 and list 2 have the same number of
>> elements N,
>> which is a necessary condition for a one-to-one matching].
>
>> With some smart partitioning of list 1 it will take ~log(N) to find
>> the nearest
>> point, so we are looking at ~ N log(N) operations...
>
>> Ciao,
>> Paolo
>
>>> --Gray
>
> I'm fine with having there be points which don't match at all w/in the
> search radius, I'm just looking to force any matches that exist to be
> recognized.
>
> The straight FOR-loop method is certainly serviceable, but I had hoped
> there was a more efficient way to do it... but it's certainly possible

> (or even likely) that anything fancier I try to do is LESS efficient.
>
> --Gray

Though I have trouble believing that FOR is the way to go when I have
~50k elements in each list.

Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 15:41:04 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>> On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> Hi all,
>
>>>> For quite a while I've been using JD Smith's match_2d routine to match
>>>> xy coords between lists. However, this and all the other matching
>>>> codes I've seen out there suffer from a variation of the uniqueness of
>>>> matches problem.
>
>>>> Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> match, i.e. enforcing that each element in list 2 only be matched to
>>>> one element in list 1; using match_2d's match_distance keyword one
>>>> could implement the same effect oneself. However, while that excludes
>>>> multiple matches to the same element, it's all done after the fact,
>>>> after the original match was determined.
>
>>>> What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> multiple-matches, and then looks for additional matches within the
>>>> search radius for elements which would become unmatched after
>>>> enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> the distance between 2_0 and 1_3 is smaller than the distance between
>>>> 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> there is element 2_1 which is also within the search radius of 1_5?
>>>> Then, 1_5 should be re-matched with 2_1.
>
>>>> My best idea thus far is to run match_2d once, identify multiple-
>>>> matches, keep the matches with minimum distance using match_distance,
>>>> then iterate with the remaining elements until match_2d returns no

>>>> matches. Can anyone come up with a better solution?
>
>>> Hmmm... what about starting with first point (a) in list 1, finding
>>> the nearest
>>> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>> all points
>>> in list 1? [this assumes list 1 and list 2 have the same number of
>>> elements N,
>>> which is a necessary condition for a one-to-one matching].
>
>>> With some smart partitioning of list 1 it will take ~log(N) to find
>>> the nearest
>>> point, so we are looking at ~ N log(N) operations...
>
>>> Ciao,
>>> Paolo
>
>>>> --Gray
>
>> I'm fine with having there be points which don't match at all w/in the
>> search radius, I'm just looking to force any matches that exist to be
>> recognized.
>
>> The straight FOR-loop method is certainly serviceable, but I had hoped
>> there was a more efficient way to do it... but it's certainly possible
>> (or even likely) that anything fancier I try to do is LESS efficient.
>
>> --Gray
>
> Though I have trouble believing that FOR is the way to go when I have
> ~50k elements in each list.

An empty for loop running 50k loops only take a few milliseconds - so
the interpretation price for looping is really not an issue here.

The issue is how efficiently you do the matching inside the loop...
that's the key issue and that's where you will see a huge difference
between a O(n) and a O(log(n)) method.

Ciao,
Paolo

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 15:41:33 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:25 am, Gray <graylikethec...@gmail.com> wrote:

> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>
>
>> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>> On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> Hi all,
>
>>>> For quite a while I've been using JD Smith's match_2d routine to match
>>>> xy coords between lists. However, this and all the other matching
>>>> codes I've seen out there suffer from a variation of the uniqueness of
>>>> matches problem.
>
>>>> Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> match, i.e. enforcing that each element in list 2 only be matched to
>>>> one element in list 1; using match_2d's match_distance keyword one
>>>> could implement the same effect oneself. However, while that excludes
>>>> multiple matches to the same element, it's all done after the fact,
>>>> after the original match was determined.
>
>>>> What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> multiple-matches, and then looks for additional matches within the
>>>> search radius for elements which would become unmatched after
>>>> enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> the distance between 2_0 and 1_3 is smaller than the distance between
>>>> 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> there is element 2_1 which is also within the search radius of 1_5?
>>>> Then, 1_5 should be re-matched with 2_1.
>
>>>> My best idea thus far is to run match_2d once, identify multiple-
>>>> matches, keep the matches with minimum distance using match_distance,
>>>> then iterate with the remaining elements until match_2d returns no
>>>> matches. Can anyone come up with a better solution?
>
>>> Hmmm... what about starting with first point (a) in list 1, finding
>>> the nearest
>>> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>> all points
>>> in list 1? [this assumes list 1 and list 2 have the same number of
>>> elements N,
>>> which is a necessary condition for a one-to-one matching].
>
>>> With some smart partitioning of list 1 it will take ~log(N) to find

>>> the nearest
>>> point, so we are looking at ~ N log(N) operations...
>
>>> Ciao,
>>> Paolo
>
>>>> --Gray
>
>> I'm fine with having there be points which don't match at all w/in the
>> search radius, I'm just looking to force any matches that exist to be
>> recognized.
>
>> The straight FOR-loop method is certainly serviceable, but I had hoped
>> there was a more efficient way to do it... but it's certainly possible
>> (or even likely) that anything fancier I try to do is LESS efficient.
>
>> --Gray
>
> Though I have trouble believing that FOR is the way to go when I have
> ~50k elements in each list.

AND... there's no guarantee that the first match you find for a given
element in list 2 is the best one.

Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 15:59:35 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > Hi all,
>
>>>> > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > xy coords between lists. However, this and all the other matching
>>>> > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > matches problem.
>
>>>> > Codes like SRCOR in the NASA IDL library let you specify a one-to-one

>>>> > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > one element in list 1; using match_2d's match_distance keyword one
>>>> > could implement the same effect oneself. However, while that excludes
>>>> > multiple matches to the same element, it's all done after the fact,
>>>> > after the original match was determined.
>
>>>> > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > multiple-matches, and then looks for additional matches within the
>>>> > search radius for elements which would become unmatched after
>>>> > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > there is element 2_1 which is also within the search radius of 1_5?
>>>> > Then, 1_5 should be re-matched with 2_1.
>
>>>> > My best idea thus far is to run match_2d once, identify multiple-
>>>> > matches, keep the matches with minimum distance using match_distance,
>>>> > then iterate with the remaining elements until match_2d returns no
>>>> > matches. Can anyone come up with a better solution?
>
>>>> Hmmm... what about starting with first point (a) in list 1, finding
>>>> the nearest
>>>> point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> all points
>>>> in list 1? [this assumes list 1 and list 2 have the same number of
>>>> elements N,
>>>> which is a necessary condition for a one-to-one matching].
>
>>>> With some smart partitioning of list 1 it will take ~log(N) to find
>>>> the nearest
>>>> point, so we are looking at ~ N log(N) operations...
>
>>>> Ciao,
>>>> Paolo
>
>>>> > --Gray
>
>>> I'm fine with having there be points which don't match at all w/in the
>>> search radius, I'm just looking to force any matches that exist to be
>>> recognized.
>
>>> The straight FOR-loop method is certainly serviceable, but I had hoped
>>> there was a more efficient way to do it... but it's certainly possible
>>> (or even likely) that anything fancier I try to do is LESS efficient.
>
>>> --Gray
>

>> Though I have trouble believing that FOR is the way to go when I have
>> ~50k elements in each list.
>
> AND... there's no guarantee that the first match you find for a given
> element in list 2 is the best one.

what is the "best" match you would like to obtain?

Ciao,
Paolo

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 16:06:35 GMT
View Forum Message <> Reply to Message

On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
> On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>
>
>> On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > Hi all,
>
>>>> > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > xy coords between lists. However, this and all the other matching
>>>> > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > matches problem.
>
>>>> > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > could implement the same effect oneself. However, while that excludes
>>>> > > multiple matches to the same element, it's all done after the fact,
>>>> > > after the original match was determined.
>
>>>> > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > multiple-matches, and then looks for additional matches within the
>>>> > > search radius for elements which would become unmatched after

>>>> > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > Then, 1_5 should be re-matched with 2_1.
>
>>>> > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > matches, keep the matches with minimum distance using match_distance,
>>>> > > then iterate with the remaining elements until match_2d returns no
>>>> > > matches. Can anyone come up with a better solution?
>
>>>> > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > the nearest
>>>> > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > all points
>>>> > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > elements N,
>>>> > which is a necessary condition for a one-to-one matching].
>
>>>> > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > the nearest
>>>> > point, so we are looking at ~ N log(N) operations...
>
>>>> > Ciao,
>>>> > Paolo
>
>>>> > > --Gray
>
>>>> I'm fine with having there be points which don't match at all w/in the
>>>> search radius, I'm just looking to force any matches that exist to be
>>>> recognized.
>
>>>> The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> there was a more efficient way to do it... but it's certainly possible
>>>> (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> --Gray
>
>>> Though I have trouble believing that FOR is the way to go when I have
>>> ~50k elements in each list.
>
>> AND... there's no guarantee that the first match you find for a given
>> element in list 2 is the best one.
>
> what is the "best" match you would like to obtain?
>
> Ciao,

> Paolo

Smallest distance between two points.

Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 16:12:24 GMT
View Forum Message <> Reply to Message

On Jul 30, 12:06 pm, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
>
>
>
>> On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>> On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > Hi all,
>
>>>> > > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > > xy coords between lists. However, this and all the other matching
>>>> > > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > > matches problem.
>
>>>> > > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > > could implement the same effect oneself. However, while that excludes
>>>> > > > multiple matches to the same element, it's all done after the fact,
>>>> > > > after the original match was determined.
>
>>>> > > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > > multiple-matches, and then looks for additional matches within the
>>>> > > > search radius for elements which would become unmatched after
>>>> > > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > > Then, 1_5 should be re-matched with 2_1.
>

>>>> > > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > > matches, keep the matches with minimum distance using match_distance,
>>>> > > > then iterate with the remaining elements until match_2d returns no
>>>> > > > matches. Can anyone come up with a better solution?
>
>>>> > > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > > the nearest
>>>> > > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > > all points
>>>> > > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > > elements N,
>>>> > > which is a necessary condition for a one-to-one matching].
>
>>>> > > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > > the nearest
>>>> > > point, so we are looking at ~ N log(N) operations...
>
>>>> > > Ciao,
>>>> > > Paolo
>
>>>> > > > --Gray
>
>>>> > I'm fine with having there be points which don't match at all w/in the
>>>> > search radius, I'm just looking to force any matches that exist to be
>>>> > recognized.
>
>>>> > The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> > there was a more efficient way to do it... but it's certainly possible
>>>> > (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> > --Gray
>
>>>> Though I have trouble believing that FOR is the way to go when I have
>>>> ~50k elements in each list.
>
>>> AND... there's no guarantee that the first match you find for a given
>>> element in list 2 is the best one.
>
>> what is the "best" match you would like to obtain?
>
>> Ciao,
>> Paolo
>
> Smallest distance between two points.

In the sense that the sum of all distances between matched points of
list (1) and (2) is minimal?

Ciao,
Paolo

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 17:09:30 GMT
View Forum Message <> Reply to Message

On Jul 30, 12:12 pm, Paolo <pgri...@gmail.com> wrote:
> On Jul 30, 12:06 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>
>
>> On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
>
>>> On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > > On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > > Hi all,
>
>>>> > > > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > > > xy coords between lists. However, this and all the other matching
>>>> > > > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > > > matches problem.
>
>>>> > > > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > > > could implement the same effect oneself. However, while that excludes
>>>> > > > > multiple matches to the same element, it's all done after the fact,
>>>> > > > > after the original match was determined.
>
>>>> > > > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > > > multiple-matches, and then looks for additional matches within the
>>>> > > > > search radius for elements which would become unmatched after
>>>> > > > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if

>>>> > > > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > > > Then, 1_5 should be re-matched with 2_1.
>
>>>> > > > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > > > matches, keep the matches with minimum distance using match_distance,
>>>> > > > > then iterate with the remaining elements until match_2d returns no
>>>> > > > > matches. Can anyone come up with a better solution?
>
>>>> > > > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > > > the nearest
>>>> > > > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > > > all points
>>>> > > > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > > > elements N,
>>>> > > > which is a necessary condition for a one-to-one matching].
>
>>>> > > > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > > > the nearest
>>>> > > > point, so we are looking at ~ N log(N) operations...
>
>>>> > > > Ciao,
>>>> > > > Paolo
>
>>>> > > > > --Gray
>
>>>> > > I'm fine with having there be points which don't match at all w/in the
>>>> > > search radius, I'm just looking to force any matches that exist to be
>>>> > > recognized.
>
>>>> > > The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> > > there was a more efficient way to do it... but it's certainly possible
>>>> > > (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> > > --Gray
>
>>>> > Though I have trouble believing that FOR is the way to go when I have
>>>> > ~50k elements in each list.
>
>>>> AND... there's no guarantee that the first match you find for a given
>>>> element in list 2 is the best one.
>
>>> what is the "best" match you would like to obtain?
>
>>> Ciao,
>>> Paolo
>
>> Smallest distance between two points.
>

> In the sense that the sum of all distances between matched points of
> list (1) and (2) is minimal?
>
> Ciao,
> Paolo

Hmmm... not exactly. In the sense that for any point in either list, it is matched to the closest point within the search radius which is not matched to a closer point. So, for example, if my matching radius is 1.5, and my 2 lists are:

1,1 1,2 3,5 6,6
and
1,2.1 0,1.5 5,6 2,2

Then, the optimal match would be to match 2_1 with 1_2, 2_2 with 1_1 (even though 2_2 is closer to 1_2 than 1_1, 1_2 is closer to 2_1), 2_3 with 1_4, and neither 1_3 or 2_4 are matched because they do not have an unmatched star w/in the search radius. In match_2d and srcor, 2_2 wouldn't be matched with anything, because the first pass would match 2_2 with 1_2, but 2_1 would have priority (because it is closer to 1_2) and 2_2 would become unmatched.

Subject: Re: yet another 2d matching question
Posted by Gray on Fri, 30 Jul 2010 17:21:00 GMT
View Forum Message <> Reply to Message

On Jul 30, 1:09 pm, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 12:12 pm, Paolo <pgri...@gmail.com> wrote:
>
>
>
>
>
>> On Jul 30, 12:06 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>>> On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > > > On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:

>
>>>> > > > > > Hi all,
>
>>>> > > > > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > > > > xy coords between lists. However, this and all the other matching
>>>> > > > > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > > > > matches problem.
>
>>>> > > > > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > > > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > > > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > > > > could implement the same effect oneself. However, while that excludes
>>>> > > > > > multiple matches to the same element, it's all done after the fact,
>>>> > > > > > after the original match was determined.
>
>>>> > > > > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > > > > multiple-matches, and then looks for additional matches within the
>>>> > > > > > search radius for elements which would become unmatched after
>>>> > > > > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > > > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > > > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > > > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > > > > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > > > > Then, 1_5 should be re-matched with 2_1.
>
>>>> > > > > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > > > > matches, keep the matches with minimum distance using match_distance,
>>>> > > > > > then iterate with the remaining elements until match_2d returns no
>>>> > > > > > matches. Can anyone come up with a better solution?
>
>>>> > > > > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > > > > the nearest
>>>> > > > > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > > > > all points
>>>> > > > > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > > > > elements N,
>>>> > > > > which is a necessary condition for a one-to-one matching].
>
>>>> > > > > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > > > > the nearest
>>>> > > > > point, so we are looking at ~ N log(N) operations...
>
>>>> > > > > Ciao,
>>>> > > > > Paolo
>
>>>> > > > > > --Gray
>
>>>> > > I'm fine with having there be points which don't match at all w/in the

>>>> > > > search radius, I'm just looking to force any matches that exist to be
>>>> > > > recognized.
>
>>>> > > > The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> > > > there was a more efficient way to do it... but it's certainly possible
>>>> > > > (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> > > > --Gray
>
>>>> > > Though I have trouble believing that FOR is the way to go when I have
>>>> > > ~50k elements in each list.
>
>>>> > AND... there's no guarantee that the first match you find for a given
>>>> > element in list 2 is the best one.
>
>>>> what is the "best" match you would like to obtain?
>
>>>> Ciao,
>>>> Paolo
>
>>> Smallest distance between two points.
>
>> In the sense that the sum of all distances between matched points of
>> list (1) and (2) is minimal?
>
>> Ciao,
>> Paolo
>
> Hmmm... not exactly. In the sense that for any point in either list,
> it is matched to the closest point within the search radius which is
> not matched to a closer point. So, for example, if my matching radius
> is 1.5, and my 2 lists are:
>
> 1,1 1,2   3,5   6,6
> and
> 1,2.1   0,1.5   5,6   2,2
>
> Then, the optimal match would be to match 2_1 with 1_2, 2_2 with 1_1
> (even though 2_2 is closer to 1_2 than 1_1, 1_2 is closer to 2_1), 2_3
> with 1_4, and neither 1_3 or 2_4 are matched because they do not have
> an unmatched star w/in the search radius. In match_2d and srcor, 2_2
> wouldn't be matched with anything, because the first pass would match
> 2_2 with 1_2, but 2_1 would have priority (because it is closer to
> 1_2) and 2_2 would become unmatched.

Sorry, typo. My example makes more sense if 2_1 = 0,1.6

Subject: Re: yet another 2d matching question
Posted by pgrigis on Fri, 30 Jul 2010 18:13:19 GMT
View Forum Message <> Reply to Message

On Jul 30, 1:21 pm, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 1:09 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>> On Jul 30, 12:12 pm, Paolo <pgri...@gmail.com> wrote:
>
>>> On Jul 30, 12:06 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > On Jul 30, 11:41 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > On Jul 30, 11:25 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > On Jul 30, 11:23 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > > On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > > > > On Jul 30, 10:01 am, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> > > > > > > Hi all,
>
>>>> > > > > > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > > > > > xy coords between lists. However, this and all the other matching
>>>> > > > > > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > > > > > matches problem.
>
>>>> > > > > > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > > > > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > > > > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > > > > > could implement the same effect oneself. However, while that excludes
>>>> > > > > > > multiple matches to the same element, it's all done after the fact,
>>>> > > > > > > after the original match was determined.
>
>>>> > > > > > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > > > > > multiple-matches, and then looks for additional matches within the
>>>> > > > > > > search radius for elements which would become unmatched after
>>>> > > > > > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > > > > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > > > > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > > > > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > > > > > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > > > > > Then, 1_5 should be re-matched with 2_1.
>

>>>> > > > > > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > > > > > matches, keep the matches with minimum distance using match_distance,
>>>> > > > > > > then iterate with the remaining elements until match_2d returns no
>>>> > > > > > > matches. Can anyone come up with a better solution?
>
>>>> > > > > > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > > > > > the nearest
>>>> > > > > > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > > > > > all points
>>>> > > > > > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > > > > > elements N,
>>>> > > > > > which is a necessary condition for a one-to-one matching].
>
>>>> > > > > > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > > > > > the nearest
>>>> > > > > > point, so we are looking at ~ N log(N) operations...
>
>>>> > > > > > Ciao,
>>>> > > > > > Paolo
>
>>>> > > > > > > --Gray
>
>>>> > > > > I'm fine with having there be points which don't match at all w/in the
>>>> > > > > search radius, I'm just looking to force any matches that exist to be
>>>> > > > > recognized.
>
>>>> > > > > The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> > > > > there was a more efficient way to do it... but it's certainly possible
>>>> > > > > (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> > > > > --Gray
>
>>>> > > > Though I have trouble believing that FOR is the way to go when I have
>>>> > > > ~50k elements in each list.
>
>>>> > > AND... there's no guarantee that the first match you find for a given
>>>> > > element in list 2 is the best one.
>
>>>> > what is the "best" match you would like to obtain?
>
>>>> > Ciao,
>>>> > Paolo
>
>>>> Smallest distance between two points.
>
>>> In the sense that the sum of all distances between matched points of
>>> list (1) and (2) is minimal?
>

>>> Ciao,
>>> Paolo
>
>> Hmmm... not exactly. In the sense that for any point in either list,
>> it is matched to the closest point within the search radius which is
>> not matched to a closer point. So, for example, if my matching radius
>> is 1.5, and my 2 lists are:
>
>> 1,1 1,2 3,5 6,6
>> and
>> 1,2.1 0,1.5 5,6 2,2
>
>> Then, the optimal match would be to match 2_1 with 1_2, 2_2 with 1_1
>> (even though 2_2 is closer to 1_2 than 1_1, 1_2 is closer to 2_1), 2_3
>> with 1_4, and neither 1_3 or 2_4 are matched because they do not have
>> an unmatched star w/in the search radius. In match_2d and srcor, 2_2
>> wouldn't be matched with anything, because the first pass would match
>> 2_2 with 1_2, but 2_1 would have priority (because it is closer to
>> 1_2) and 2_2 would become unmatched.
>
> Sorry, typo. My example makes more sense if 2_1 = 0,1.6

Let me argue that the algorithm you are describing for matching
points does not deliver very satisfactory results.

In fact it is much easier to think about this as a 1-dim
problem (and ignoring for now the fact that you reject some matches
if they are too far apart).

Data:
List 1: [1,5 ,8,9]
List 2: [0,2.5,3,6]

Now the algorithm would be to travel along a list
from first to last elements and assign the closest
unmatched points.

Let's start with building matches from list 1:
1 <-> 0
5 <-> 6
8 <-> 3
9 <-> 2.5

(you get this numbers by starting from 1, looking for closest number
which is 0, assigning 1 <-> 0 match and removing the matched points
from the list, then looking for the nearest element to 5 etc.)

On the other hand if you start building matches from list 2:

0 <-> 1
2.5 <-> 5
3 <-> 8
6 <-> 9

These solutions are different from each other.

Moreover, if the arrays are reordered internally,
another different solution would be found.

You would probably want a way of finding matches that
does not depend on the internal order of the 2 lists,
or on which list you start with.

Ciao,
Paolo

Subject: Re: yet another 2d matching question
Posted by JDS on Fri, 30 Jul 2010 22:23:27 GMT
View Forum Message <> Reply to Message

Paulo spotted the issue. What determines whether a given point in the search list "is not matched to a closer point"? Your 1-to-1 match will be sensitive to the input ordering of the target list. The intention of match_radius is to specify the maximum separation beneath which all matches are "equally good". For example, the statistical uncertainty in the position itself. Multiple matches would then imply either is an equally good match. If you still wanted to do this (for example if you are conducting a match for which sub-match_distance separations are still meaningful), it will have to be a pre- or post-processing step, since all matches are performed in parallel (which is what gives MATCH_2D its speed).

JD

Subject: Re: yet another 2d matching question
Posted by Gray on Sat, 31 Jul 2010 11:42:05 GMT
View Forum Message <> Reply to Message

On Jul 30, 2:13 pm, Paolo <pgri...@gmail.com> wrote:
> On Jul 30, 1:21 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>
>

>
>
>> On Jul 30, 1:09 pm, Gray <graylikethec...@gmail.com> wrote:
>
>>> On Jul 30, 12:12 pm, Paolo <pgri...@gmail.com> wrote:
>
>>>> On Jul 30, 12:06 pm, Gray <graylikethec...@gmail.com> wrote:
>
>>>> > On Jul 30, 11:59 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > On Jul 30, 11:41 am, Gray <graylikethec...@gmail.com> wrote:
>
>>>> > > > On Jul 30, 11:25 am, Gray <graylikethec...@gmail.com> wrote:
>
>>>> > > > > On Jul 30, 11:23 am, Gray <graylikethec...@gmail.com> wrote:
>
>>>> > > > > > On Jul 30, 11:15 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> > > > > > > On Jul 30, 10:01 am, Gray <graylikethec...@gmail.com> wrote:
>
>>>> > > > > > > > Hi all,
>
>>>> > > > > > > > For quite a while I've been using JD Smith's match_2d routine to match
>>>> > > > > > > > xy coords between lists. However, this and all the other matching
>>>> > > > > > > > codes I've seen out there suffer from a variation of the uniqueness of
>>>> > > > > > > > matches problem.
>
>>>> > > > > > > > Codes like SRCOR in the NASA IDL library let you specify a one-to-one
>>>> > > > > > > > match, i.e. enforcing that each element in list 2 only be matched to
>>>> > > > > > > > one element in list 1; using match_2d's match_distance keyword one
>>>> > > > > > > > could implement the same effect oneself. However, while that excludes
>>>> > > > > > > > multiple matches to the same element, it's all done after the fact,
>>>> > > > > > > > after the original match was determined.
>
>>>> > > > > > > > What I'm looking for is an algorithm that matches 2 lists, identifies
>>>> > > > > > > > multiple-matches, and then looks for additional matches within the
>>>> > > > > > > > search radius for elements which would become unmatched after
>>>> > > > > > > > enforcing a one-to-one relationship. What I mean is, say element 0 in
>>>> > > > > > > > list 2 is matched to both element 3 and element 5 in list 1, and that
>>>> > > > > > > > the distance between 2_0 and 1_3 is smaller than the distance between
>>>> > > > > > > > 2_0 and 1_5. In that case, 1_5 would become unmatched; but what if
>>>> > > > > > > > there is element 2_1 which is also within the search radius of 1_5?
>>>> > > > > > > > Then, 1_5 should be re-matched with 2_1.
>
>>>> > > > > > > > My best idea thus far is to run match_2d once, identify multiple-
>>>> > > > > > > > matches, keep the matches with minimum distance using match_distance,
>>>> > > > > > > > then iterate with the remaining elements until match_2d returns no
>>>> > > > > > > > matches. Can anyone come up with a better solution?

>
>>>> > > > > > > Hmmm... what about starting with first point (a) in list 1, finding
>>>> > > > > > > the nearest
>>>> > > > > > > point (b) to (a) in list 2, removing (b) from list 2 and repeat for
>>>> > > > > > > all points
>>>> > > > > > > in list 1? [this assumes list 1 and list 2 have the same number of
>>>> > > > > > > elements N,
>>>> > > > > > > which is a necessary condition for a one-to-one matching].
>
>>>> > > > > > > With some smart partitioning of list 1 it will take ~log(N) to find
>>>> > > > > > > the nearest
>>>> > > > > > > point, so we are looking at ~ N log(N) operations...
>
>>>> > > > > > > Ciao,
>>>> > > > > > > Paolo
>
>>>> > > > > > > > --Gray
>
>>>> > > > > > I'm fine with having there be points which don't match at all w/in the
>>>> > > > > > search radius, I'm just looking to force any matches that exist to be
>>>> > > > > > recognized.
>
>>>> > > > > > The straight FOR-loop method is certainly serviceable, but I had hoped
>>>> > > > > > there was a more efficient way to do it... but it's certainly possible
>>>> > > > > > (or even likely) that anything fancier I try to do is LESS efficient.
>
>>>> > > > > > --Gray
>
>>>> > > > > Though I have trouble believing that FOR is the way to go when I have
>>>> > > > > ~50k elements in each list.
>
>>>> > > > AND... there's no guarantee that the first match you find for a given
>>>> > > > element in list 2 is the best one.
>
>>>> > > what is the "best" match you would like to obtain?
>
>>>> > > Ciao,
>>>> > > Paolo
>
>>>> > Smallest distance between two points.
>
>>>> In the sense that the sum of all distances between matched points of
>>>> list (1) and (2) is minimal?
>
>>>> Ciao,
>>>> Paolo
>
>>> Hmmm... not exactly. In the sense that for any point in either list,

>>> it is matched to the closest point within the search radius which is
>>> not matched to a closer point. So, for example, if my matching radius
>>> is 1.5, and my 2 lists are:
>
>>> 1,1 1,2 3,5 6,6
>>> and
>>> 1,2.1 0,1.5 5,6 2,2
>
>>> Then, the optimal match would be to match 2_1 with 1_2, 2_2 with 1_1
>>> (even though 2_2 is closer to 1_2 than 1_1, 1_2 is closer to 2_1), 2_3
>>> with 1_4, and neither 1_3 or 2_4 are matched because they do not have
>>> an unmatched star w/in the search radius. In match_2d and srcor, 2_2
>>> wouldn't be matched with anything, because the first pass would match
>>> 2_2 with 1_2, but 2_1 would have priority (because it is closer to
>>> 1_2) and 2_2 would become unmatched.
>
>> Sorry, typo. My example makes more sense if 2_1 = 0,1.6
>
> Let me argue that the algorithm you are describing for matching
> points does not deliver very satisfactory results.
>
> In fact it is much easier to think about this as a 1-dim
> problem (and ignoring for now the fact that you reject some matches
> if they are too far apart).
>
> Data:
> List 1: [1,5 ,8,9]
> List 2: [0,2.5,3,6]
>
> Now the algorithm would be to travel along a list
> from first to last elements and assign the closest
> unmatched points.
>
> Let's start with building matches from list 1:
> 1 <-> 0
> 5 <-> 6
> 8 <-> 3
> 9 <-> 2.5
>
> (you get this numbers by starting from 1, looking for closest number
> which is 0, assigning 1 <-> 0 match and removing the matched points
> from the list, then looking for the nearest element to 5 etc.)
>
> On the other hand if you start building matches from list 2:
> 0 <-> 1
> 2.5 <-> 5
> 3 <-> 8
> 6 <-> 9

>
> These solutions are different from each other.
>
> Moreover, if the arrays are reordered internally,
> another different solution would be found.
>
> You would probably want a way of finding matches that
> does not depend on the internal order of the 2 lists,
> or on which list you start with.
>
> Ciao,
> Paolo

The FOR-loop indeed has the problem of internal ordering, which is essentially what I was trying to say.

I did get the "iterated match_2d" algorithm working.

Subject: Re: yet another 2d matching question
Posted by Gray on Sat, 31 Jul 2010 11:47:14 GMT
View Forum Message <> Reply to Message

On Jul 30, 6:23 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
> Paulo spotted the issue. What determines whether a given point in the
> search list "is not matched to a closer point"? Your 1-to-1 match
> will be sensitive to the input ordering of the target list. The
> intention of match_radius is to specify the maximum separation beneath
> which all matches are "equally good". For example, the statistical
> uncertainty in the position itself. Multiple matches would then imply
> either is an equally good match. If you still wanted to do this (for
> example if you are conducting a match for which sub-match_distance
> separations are still meaningful), it will have to be a pre- or post-
> processing step, since all matches are performed in parallel (which is
> what gives MATCH_2D its speed).
>
> JD

Hmm... if all matches are equally good within the match_distance, then how does match_2d prioritize matches when there is more than one source in list b within the match radius of list a? This could happen when, for example, the positional accuracy of the sources in each list is low, but there is a possible shift (translation+rotation+etc.) between the members of the two lists which necessitates a larger match radius.

Subject: Re: yet another 2d matching question
Posted by Gray on Sat, 31 Jul 2010 11:48:12 GMT
View Forum Message <> Reply to Message

On Jul 31, 7:47 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 6:23 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
>
>> Paulo spotted the issue. What determines whether a given point in the
>> search list "is not matched to a closer point"? Your 1-to-1 match
>> will be sensitive to the input ordering of the target list. The
>> intention of match_radius is to specify the maximum separation beneath
>> which all matches are "equally good". For example, the statistical
>> uncertainty in the position itself. Multiple matches would then imply
>> either is an equally good match. If you still wanted to do this (for
>> example if you are conducting a match for which sub-match_distance
>> separations are still meaningful), it will have to be a pre- or post-
>> processing step, since all matches are performed in parallel (which is
>> what gives MATCH_2D its speed).
>
>> JD
>
> Hmm... if all matches are equally good within the match_distance, then
> how does match_2d prioritize matches when there is more than one
> source in list b within the match radius of list a? This could happen
> when, for example, the positional accuracy of the sources in each list
> is low, but there is a possible shift (translation+rotation+etc.)
> between the members of the two lists which necessitates a larger match
> radius.

I mean the positional accuracy is HIGH. The UNCERTAINTY in the
positions is LOW.

Subject: Re: yet another 2d matching question
Posted by JDS on Tue, 03 Aug 2010 20:53:03 GMT
View Forum Message <> Reply to Message

On Jul 31, 7:47 am, Gray <grayliketheco...@gmail.com> wrote:
> On Jul 30, 6:23 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
>
>> Paulo spotted the issue. What determines whether a given point in the
>> search list "is not matched to a closer point"? Your 1-to-1 match
>> will be sensitive to the input ordering of the target list. The
>> intention of match_radius is to specify the maximum separation beneath
>> which all matches are "equally good". For example, the statistical
>> uncertainty in the position itself. Multiple matches would then imply
>> either is an equally good match. If you still wanted to do this (for
>> example if you are conducting a match for which sub-match_distance

>> separations are still meaningful), it will have to be a pre- or post-
>> processing step, since all matches are performed in parallel (which is
>> what gives MATCH_2D its speed).
>
>> JD
>
> Hmm... if all matches are equally good within the match_distance, then
> how does match_2d prioritize matches when there is more than one
> source in list b within the match radius of list a? This could happen
> when, for example, the positional accuracy of the sources in each list
> is low, but there is a possible shift (translation+rotation+etc.)
> between the members of the two lists which necessitates a larger match
> radius.

This is only true if your match_distance represents some positional uncertainty; i.e. it's not meaningful to say a given star is 50 milliarseconds closer when the precision with which you know your search list is 2 arcseconds. It does simply return the closest point within match_distance if there are multiple matches. You could certainly alter this to return *all* matches within match_radius, then use post-processing to enforce a 1-to-1 matching. IDL 8's new LIST type would make this much easier than before (when I've used REVERSE_INDEX style arrays for the same purpose).

JD