Subject: Re: summing mulit-D array along 1 dimension, at an angle to the rows
Posted by Kenneth P. Bowman on Thu, 04 Nov 2010 19:41:42 GMT
View Forum Message <> Reply to Message

In article
<9481d9a7-07ab-4042-aa2f-35ac15d2425d@w38g2000pri.googlegroups.com>,
 rrs <rrstrickler@gmail.com> wrote:

> Suppose you have a 2d or 3d array that contains some information (in
> my case, density) with a relatively complex geometry, and you want to
> collapse this to 1d or 2d by summing (to obtain a column density).
> This is simple if you want to sum along an axis, but I'm having
> problems finding an efficient way to sum along an angle (i.e. if the
> observer is at a 30 degree angle to the x-axis). The best I've been
> able to do is rotate the array by the desired angle, and then perform
> the sum. Example:
> density = transform_volume(TEMPORARY(density), ROTATION=[0,angle,0]);
> column_density = REVERSE(TOTAL(REVERSE(density*voxel_size), 1, /
> CUMULATIVE));
> All the reversing is so that I get a column density from each voxel to
> the observer along the row.
>
> The problem is that I'm working with large arrays, and I need to do
> this for several angles, which makes the whole process both memory-
> intensive and slow.
>
> Am I missing some more sensible way to do this?

I suggest that you interpolate the 2-D or 3-D grid to the line
that you want to average along. Then average the results.

That should be faster than rotating the entire grid.

Assuming your grids are reasonably regular (they can be
stretched, but should be separable), INTERPOLATE is the tool
for this.

Ken Bowman

Subject: Re: summing mulit-D array along 1 dimension, at an angle to the rows
Posted by rrs on Thu, 04 Nov 2010 21:26:18 GMT
View Forum Message <> Reply to Message

If I only needed the column density along a single line, interpolating
as you suggest would work well. However, I need the column density at
each voxel at a specific viewing angle, meaning I really do need to
act on, and get information about, the entire array. Sorry that

wasn't clear in the original post.

TRANSFORM_VOLUME is meant to be fairly efficient, so it may be the
best I can do.

On Nov 4, 12:41 pm, "Kenneth P. Bowman" <k-bow...@null.edu> wrote:
> In article
> < 9481d9a7-07ab-4042-aa2f-35ac15d24...@w38g2000pri.googlegroup s.com >,
>
>
>
> rrs <rrstrick...@gmail.com> wrote:
>> Suppose you have a 2d or 3d array that contains some information (in
>> my case, density) with a relatively complex geometry, and you want to
>> collapse this to 1d or 2d by summing (to obtain a column density).
>> This is simple if you want to sum along an axis, but I'm having
>> problems finding an efficient way to sum along an angle (i.e. if the
>> observer is at a 30 degree angle to the x-axis). The best I've been
>> able to do is rotate the array by the desired angle, and then perform
>> the sum. Example:
>> density = transform_volume(TEMPORARY(density), ROTATION=[0,angle,0]);
>> column_density = REVERSE(TOTAL(REVERSE(density*voxel_size), 1, /
>> CUMULATIVE));
>> All the reversing is so that I get a column density from each voxel to
>> the observer along the row.
>
>> The problem is that I'm working with large arrays, and I need to do
>> this for several angles, which makes the whole process both memory-
>> intensive and slow.
>
>> Am I missing some more sensible way to do this?
>
> I suggest that you interpolate the 2-D or 3-D grid to the line
> that you want to average along. Then average the results.
>
> That should be faster than rotating the entire grid.
>
> Assuming your grids are reasonably regular (they can be
> stretched, but should be separable), INTERPOLATE is the tool
> for this.
>
> Ken Bowman

Subject: Re: summing mulit-D array along 1 dimension, at an angle to the rows
Posted by rogass on Thu, 04 Nov 2010 23:14:42 GMT
View Forum Message <> Reply to Message

On 4 Nov., 22:26, rrs <rrstrick...@gmail.com> wrote:
> If I only needed the column density along a single line, interpolating
> as you suggest would work well. However, I need the column density at
> each voxel at a specific viewing angle, meaning I really do need to
> act on, and get information about, the entire array. Sorry that
> wasn't clear in the original post.
>
> TRANSFORM_VOLUME is meant to be fairly efficient, so it may be the
> best I can do.
>
> On Nov 4, 12:41 pm, "Kenneth P. Bowman" <k-bow...@null.edu> wrote:
>
>
>
>
>
>
>
>
>> In article
>> < 9481d9a7-07ab-4042-aa2f-35ac15d24...@w38g2000pri.googlegroup s.com >,
>
>> rrs <rrstrick...@gmail.com> wrote:
>>> Suppose you have a 2d or 3d array that contains some information (in
>>> my case, density) with a relatively complex geometry, and you want to
>>> collapse this to 1d or 2d by summing (to obtain a column density).
>>> This is simple if you want to sum along an axis, but I'm having
>>> problems finding an efficient way to sum along an angle (i.e. if the
>>> observer is at a 30 degree angle to the x-axis). The best I've been
>>> able to do is rotate the array by the desired angle, and then perform
>>> the sum. Example:
>>> density = transform_volume(TEMPORARY(density), ROTATION=[0,angle,0]);
>>> column_density = REVERSE(TOTAL(REVERSE(density*voxel_size), 1, /
>>> CUMULATIVE));
>>> All the reversing is so that I get a column density from each voxel to
>>> the observer along the row.
>
>>> The problem is that I'm working with large arrays, and I need to do
>>> this for several angles, which makes the whole process both memory-
>>> intensive and slow.
>
>>> Am I missing some more sensible way to do this?
>
>> I suggest that you interpolate the 2-D or 3-D grid to the line
>> that you want to average along. Then average the results.
>
>> That should be faster than rotating the entire grid.
>
>> Assuming your grids are reasonably regular (they can be

>> stretched, but should be separable), INTERPOLATE is the tool
>> for this.
>
>> Ken Bowman

Hi,
maybe you can solve this problem by a projection of the matrix to a vector - orthogonal to your line of sight. This projection is basically the sum into the orthogonal direction of the vector. Both matrix and vector must be normed. So, if you like to know what is the sum of the [1,1] vector is into the [0,1] direction you would have:
sum = transpose([0.5,0.5])##[1,0] =0.5

To project a matrix on a vector you simply do:
transpose(normedmatrix)##normedvector
To do this you choose an vector which fulfills the condition:
total(normedlineofsightvector * normedvector) = 0

Hope it helps

CR
