Subject: Re: Smoothing 3D array with periodic boundaries: what am I missing?
Posted by Jeremy Bailin on Fri, 25 Sep 2009 03:52:00 GMT
View Forum Message <> Reply to Message

On Sep 24, 1:19 pm, Luds <lud...@uvic.ca> wrote:
> I've been trying for a couple days now to write a Gaussian-smoothing
> algorithm to smooth a cube of (scalar) data with periodic boundary
> conditions (this is needed for my task since "structure" in the data
> that straddles an edge of the cube appears on two+ sides of the box).
> I've made it so far, but now can't seem to get around excessive For-
> loop's...
>
> For example, say the box of scalars values runs from (0,1) in x,y, and
> z, and has N^3 points. To smooth at point (x,y,z) in the box I
> generate a 3-D Gaussian with its centroid (mean) at point x,y,z:
>
> Gauss_field = rebin(periodic_gauss_func(X,[[sig],[x]]),N,N,N) * $
>   rebin(reform(periodic_gauss_func(X,[[sig],[y]]),
> 1,N),N,N,N) * $
>   rebin(reform(periodic_gauss_func(X,[[sig],[z]]),
> 1,1,N),N,N,N)
>
> where periodic_gauss_func is a 1-D Gaussian kernel function that wraps
> around the box edge, X=(0,1,...N-1)... sig=sigma. (i.e. this just does
> separate Gaussian smoothing along each direction and combines the
> result).
>
> Then the smoothed field at point (x,y,z) is something like
>
> Smoothed(x,y,z) = TOTAL(TOTAL(scalar_field*Gauss_field,1))
>
> What I can't figure out is an efficient way to do this for all (x,y,z)
> - for a N=1024^3 grid it takes a couple seconds to generate
> Gauss_field. Realistically, I'll have N=1024^3, so For-loops are
> pretty much useless(???), and memory is a bit of an issue too.
>
> Does anyone know of any "canned" routines to do this type of Gaussian
> smoothing? Or of an efficient way to convolve my 3D Gaussian field
> with my scalar field for all (x,y,z)? (I must stress that the Gaussian
> kernel must not be affected by, or truncated at, the box edge)
>
> Many thanks!!
>
> Aaron

Wouldn't the Fourier convolution theorem approach work here? FFT your
data cube, FFT your 3D Gaussian kernel, multiply them, and reverse FFT
them back out? You may need to judiciously use TEMPORARY and/or the /

OVERWRITE keyword if memory is an issue.

-Jeremy.

Subject: Re: Smoothing 3D array with periodic boundaries: what am I missing?
Posted by Luds on Mon, 28 Sep 2009 06:56:33 GMT
View Forum Message <> Reply to Message

On Sep 25, 5:52 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> On Sep 24, 1:19 pm, Luds <lud...@uvic.ca> wrote:
>
>
>
>> I've been trying for a couple days now to write a Gaussian-smoothing
>> algorithm to smooth a cube of (scalar) data with periodic boundary
>> conditions (this is needed for my task since "structure" in the data
>> that straddles an edge of the cube appears on two+ sides of the box).
>> I've made it so far, but now can't seem to get around excessive For-
>> loop's...
>
>> For example, say the box of scalars values runs from (0,1) in x,y, and
>> z, and has N^3 points. To smooth at point (x,y,z) in the box I
>> generate a 3-D Gaussian with its centroid (mean) at point x,y,z:
>
>> Gauss_field = rebin(periodic_gauss_func(X,[[sig],[x]]),N,N,N) * $
>> rebin(reform(periodic_gauss_func(X,[[sig],[y]]),
>> 1,N),N,N,N) * $
>> rebin(reform(periodic_gauss_func(X,[[sig],[z]]),
>> 1,1,N),N,N,N)
>
>> where periodic_gauss_func is a 1-D Gaussian kernel function that wraps
>> around the box edge, X=(0,1,...N-1)... sig=sigma. (i.e. this just does
>> separate Gaussian smoothing along each direction and combines the
>> result).
>
>> Then the smoothed field at point (x,y,z) is something like
>
>> Smoothed(x,y,z) = TOTAL(TOTAL(scalar_field*Gauss_field,1))
>
>> What I can't figure out is an efficient way to do this for all (x,y,z)
>> - for a N=1024^3 grid it takes a couple seconds to generate
>> Gauss_field. Realistically, I'll have N=1024^3, so For-loops are
>> pretty much useless(???), and memory is a bit of an issue too.
>
>> Does anyone know of any "canned" routines to do this type of Gaussian
>> smoothing? Or of an efficient way to convolve my 3D Gaussian field
>> with my scalar field for all (x,y,z)? (I must stress that the Gaussian

>> kernel must not be affected by, or truncated at, the box edge)
>
>> Many thanks!!
>
>> Aaron
>
> Wouldn't the Fourier convolution theorem approach work here? FFT your
> data cube, FFT your 3D Gaussian kernel, multiply them, and reverse FFT
> them back out? You may need to judiciously use TEMPORARY and/or the /
> OVERWRITE keyword if memory is an issue.
>
> -Jeremy.

Yeah, I guess this is the way to go after all.

I had tried this but didn't really trust my smoothed result. E.g. I attempted to smooth a slab of my data cube with smoothed_field=fft(fft(field)*gaussian_filter,1), but only the upper half of the smooth field resembled the original image; the lower half was an inverted backwards copy of the upper half (at least that's what it looked like to my eye). (BTW, it's a Gaussian random field, CDM power-spectrum).

I guess I'll keep messing around with the IDL's fft. I've read on the help pages that the lowest frequencies in the fft should appear something like a spike in the middle of the fft'd image... I see a spike in the corner (0,0) of the image, which means I probably misinterpreting something simple.

Thanks!!

Subject: Re: Smoothing 3D array with periodic boundaries: what am I missing?
Posted by pgrigis on Mon, 28 Sep 2009 14:10:57 GMT
View Forum Message <> Reply to Message

On Sep 28, 2:56 am, Luds <lud...@uvic.ca> wrote:
> On Sep 25, 5:52 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>> On Sep 24, 1:19 pm, Luds <lud...@uvic.ca> wrote:
>
>>> I've been trying for a couple days now to write a Gaussian-smoothing
>>> algorithm to smooth a cube of (scalar) data with periodic boundary
>>> conditions (this is needed for my task since "structure" in the data
>>> that straddles an edge of the cube appears on two+ sides of the box).
>>> I've made it so far, but now can't seem to get around excessive For-
>>> loop's...

>
>>> For example, say the box of scalars values runs from (0,1) in x,y, and
>>> z, and has N^3 points. To smooth at point (x,y,z) in the box I
>>> generate a 3-D Gaussian with its centroid (mean) at point x,y,z:
>
>>> Gauss_field = rebin(periodic_gauss_func(X,[[sig],[x]]),N,N,N) * $
>>> rebin(reform(periodic_gauss_func(X,[[sig],[y]]),
>>> 1,N),N,N,N) * $
>>> rebin(reform(periodic_gauss_func(X,[[sig],[z]]),
>>> 1,1,N),N,N,N)
>
>>> where periodic_gauss_func is a 1-D Gaussian kernel function that wraps
>>> around the box edge, X=(0,1,...N-1)... sig=sigma. (i.e. this just does
>>> separate Gaussian smoothing along each direction and combines the
>>> result).
>
>>> Then the smoothed field at point (x,y,z) is something like
>
>>> Smoothed(x,y,z) = TOTAL(TOTAL(scalar_field*Gauss_field,1))
>
>>> What I can't figure out is an efficient way to do this for all (x,y,z)
>>> - for a N=1024^3 grid it takes a couple seconds to generate
>>> Gauss_field. Realistically, I'll have N=1024^3, so For-loops are
>>> pretty much useless(???), and memory is a bit of an issue too.
>
>>> Does anyone know of any "canned" routines to do this type of Gaussian
>>> smoothing? Or of an efficient way to convolve my 3D Gaussian field
>>> with my scalar field for all (x,y,z)? (I must stress that the Gaussian
>>> kernel must not be affected by, or truncated at, the box edge)
>
>>> Many thanks!!
>
>>> Aaron
>
>> Wouldn't the Fourier convolution theorem approach work here? FFT your
>> data cube, FFT your 3D Gaussian kernel, multiply them, and reverse FFT
>> them back out? You may need to judiciously use TEMPORARY and/or the /
>> OVERWRITE keyword if memory is an issue.
>
>> -Jeremy.
>
> Yeah, I guess this is the way to go after all.
>
> I had tried this but didn't really trust my smoothed result. E.g. I
> attempted to smooth a slab of my data cube with smoothed_field=fft(fft
> (field)*gaussian_filter,1), but only the upper half of the smooth
> field resembled the original image; the lower half was an inverted
> backwards copy of the upper half (at least that's what it looked like

> to my eye). (BTW, it's a Gaussian random field, CDM power-spectrum).
>
> I guess I'll keep messing around with the IDL's fft. I've read on the
> help pages that the lowest frequencies in the fft should appear
> something like a spike in the middle of the fft'd image... I see a
> spike in the corner (0,0) of the image, which means I probably
> misinterpreting something simple.

Don't worry - the ordering of the frequencies in FFT nearly
always is set like that - if that's confusing, a shift of
half the size of the array will set them the way you expect
them to be (with 0 frequency in the middle of the array).

To see the effect - take the 1-dim FFT of a gaussian.
The result is also a gaussian - but you'll need a shift
of half the size of the array to have it properly centered
on the middle of the array.

Ciao,
Paolo

>
> Thanks!!

Subject: Re: Smoothing 3D array with periodic boundaries: what am I missing?
Posted by Luds on Mon, 28 Sep 2009 18:18:05 GMT
View Forum Message <> Reply to Message

On Sep 28, 4:10 pm, Paolo <pgri...@gmail.com> wrote:
> On Sep 28, 2:56 am, Luds <lud...@uvic.ca> wrote:
>
>
>
>> On Sep 25, 5:52 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>> On Sep 24, 1:19 pm, Luds <lud...@uvic.ca> wrote:
>
>>>> I've been trying for a couple days now to write a Gaussian-smoothing
>>>> algorithm to smooth a cube of (scalar) data with periodic boundary
>>>> conditions (this is needed for my task since "structure" in the data
>>>> that straddles an edge of the cube appears on two+ sides of the box).
>>>> I've made it so far, but now can't seem to get around excessive For-
>>>> loop's...
>
>>>> For example, say the box of scalars values runs from (0,1) in x,y, and

>>>> z, and has N^3 points. To smooth at point (x,y,z) in the box I
>>>> generate a 3-D Gaussian with its centroid (mean) at point x,y,z:
>
>>>> Gauss_field = rebin(periodic_gauss_func(X,[[sig],[x]]),N,N,N) * $
>>>>   rebin(reform(periodic_gauss_func(X,[[sig],[y]]),
>>>> 1,N),N,N,N) * $
>>>>   rebin(reform(periodic_gauss_func(X,[[sig],[z]]),
>>>> 1,1,N),N,N,N)
>
>>>> where periodic_gauss_func is a 1-D Gaussian kernel function that wraps
>>>> around the box edge, X=(0,1,...N-1)... sig=sigma. (i.e. this just does
>>>> separate Gaussian smoothing along each direction and combines the
>>>> result).
>
>>>> Then the smoothed field at point (x,y,z) is something like
>
>>>> Smoothed(x,y,z) = TOTAL(TOTAL(scalar_field*Gauss_field,1))
>
>>>> What I can't figure out is an efficient way to do this for all (x,y,z)
>>>> - for a N=1024^3 grid it takes a couple seconds to generate
>>>> Gauss_field. Realistically, I'll have N=1024^3, so For-loops are
>>>> pretty much useless(???), and memory is a bit of an issue too.
>
>>>> Does anyone know of any "canned" routines to do this type of Gaussian
>>>> smoothing? Or of an efficient way to convolve my 3D Gaussian field
>>>> with my scalar field for all (x,y,z)? (I must stress that the Gaussian
>>>> kernel must not be affected by, or truncated at, the box edge)
>
>>>> Many thanks!!
>
>>>> Aaron
>
>>> Wouldn't the Fourier convolution theorem approach work here? FFT your
>>> data cube, FFT your 3D Gaussian kernel, multiply them, and reverse FFT
>>> them back out? You may need to judiciously use TEMPORARY and/or the /
>>> OVERWRITE keyword if memory is an issue.
>
>>> -Jeremy.
>
>> Yeah, I guess this is the way to go after all.
>
>> I had tried this but didn't really trust my smoothed result. E.g. I
>> attempted to smooth a slab of my data cube with smoothed_field=fft(fft
>> (field)*gaussian_filter,1), but only the upper half of the smooth
>> field resembled the original image; the lower half was an inverted
>> backwards copy of the upper half (at least that's what it looked like
>> to my eye). (BTW, it's a Gaussian random field, CDM power-spectrum).
>

>> I guess I'll keep messing around with the IDL's fft. I've read on the
>> help pages that the lowest frequencies in the fft should appear
>> something like a spike in the middle of the fft'd image... I see a
>> spike in the corner (0,0) of the image, which means I probably
>> misinterpreting something simple.
>
> Don't worry - the ordering of the frequencies in FFT nearly
> always is set like that - if that's confusing, a shift of
> half the size of the array will set them the way you expect
> them to be (with 0 frequency in the middle of the array).
>
> To see the effect - take the 1-dim FFT of a gaussian.
> The result is also a gaussian - but you'll need a shift
> of half the size of the array to have it properly centered
> on the middle of the array.
>
> Ciao,
> Paolo
>
>
>
>> Thanks!!
>
>

Thank Paolo. Yeah, I figure out the shift of the fft frequencies and
now everything is a expected.