Subject: Re: Imposing inverse fft to be real
Posted by Craig Markwardt on Tue, 22 Jan 2013 01:22:14 GMT
View Forum Message <> Reply to Message

On Monday, January 21, 2013 6:41:19 PM UTC-5, bblay wrote:
> The issue: I am creating complex arrays with the same amplitude but randomizing the phases, then using inverse FFT.
>
>
>
> This is easy, however I would like to impose that they are conjugate symmetric, i.e. Hermitian. The original array ("dn" and its fft "org" as defined below) are real valued and hence will return mostly (i.e. withing floating point precision) real valued FFT results. However, the randomized phase array ("rr", as defined below) should have the conjugate symmetry imposed on it in order to insure that the resulting inverse FFT will have negligible imaginary components. This seems like it would be straight forward....
>
>
>
> dn=readfits('array.fits.gz')
>
>
>
> org=fft(dn,/double,/center) ; take the fft, center frequencies around origin
>
>
>
> imbx=imaginary(org)
>
> rebx=real_part(org)
>
>
>
> phi=atan(imbx,rebx,/phase)
>
> phaserand=dblarr(512,512,512)
>
> amp=abs(org)
>
>
>
> i_index=phi(sort(randomu(seed,[512,512,512]))) ; sorted random phases....
>
> phaserand[*,*,*]=i_index ; sorted randomized phases
>
> rr=complex(amp*cos(phaserand),amp*sin(phaserand),/double)
>
>

>
> for i=0,255 do begin   ;impose conjugate symmetry for real result
>
> for j=0,255 do begin
>
> for k=0,255 do begin
>
>
>
> rr[511-i,511-j,511-k]=conj(rr[i,j,k])
>
>
>
> endfor
>
> endfor
>
> endfor
>
>
>
> inverse_rr=fft(rr,/inverse,/double,/center)
>
>
>
> I've tried this using the center option, shifting the arrays...etc etc. But no matter what I try, the inverse of rr has non-negligible imaginary values :(
>
>
>
> Any ideas on this? Thanks! :)

I kind of think you are trying to bite off more than you can chew. First try with an array of 8 or 16 values instead of 512x512 so you can understand how the FFT arranges its output values.

For 1D FFT, conjugate symmetry does not hold for the 0th value and the N/2th value. These are the DC component and the Nyquist frequency respectively. I suspect this is where your problem lies. How this works in 2D is a slightly more complicated question :-).

Craig