Subject: Secrets FFTs revealed!! 2D example included
Posted by Peter Brooker on Fri, 05 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
I have just been through a learning curve on FFTs. Much thanks to Alan

Barnett for putting me on the right track. I think I have them figured

out and now want to write a reference that captures my present level
of

understanding. Realize that I have learned only as much of the FFT

theory as needed. My motivation is that I am going to be applying the

FFT functions to modeling the effect of a finite lens size on the image.

(The finite lens size will chop off the higher order frequencies).

Perhaps somebody else will want to expand/improve this reference. These

are only my best guesses to how everything works. Perhaps this is

something for the IDL FAQ.
<p>This reference is organized as follows:

PART#1:Relate complex expansion to real Fourier series

PART#2:IDL form of complex expansion

PART#3:Specific example: f(t) = sin (4*pi*t)

PART#4:Specific example: T(x,y) =
<p>PART#1: Relate complex expansion to real Fourier series.

Assume you have a function f(t) that is periodic in t with a period
T.

Then there exists coefficients a_n & b_n such that
<p>f(t) = a_o + sum_n(a_n*cos(2*pi*n*t/T)+b_n*sin(2*pi*n*t/T)) , n=1,2,...
<p>This is just the Fourier series of function with period X. Nothing
new

here. See eq #4, section 10.3, Advanced Engineering Mathematics,

Kreyszig. This expansion though assumes -T/2 < t < T/2
<p>Now consider an alternate form of the Fourier series expansion.
<p>f(t)=sum_n(A_n*exp(j*2*pi*n*t/T)), n=0,1,2,...
<p>In order for me to be comfortable with this expansion I need to see
how

this expansion relates to the expansion above. In particular, how do
the

complex An relate to the real a_n and b_n?
<p>Consider the following:
<p>II= sum_n(A_n*exp(j*2*pi*n*t/T)), n=0,1,2,...   j*j
= -1
<p>   =A_o+sum_n(A_n*(cos(2*pi*n*t/T)+j*sin(2*pi*n*t/T)),
n=1,2,...
<p>Let A_n=(aa_n+j*bb_n)
<p>II= A_o+sum_n((aa_n+j*bb_n)*(cos(2*pi*n*t/T)+j*sin(2*pi*n*t/T))
<p> = A_o + sum_n(aa_n*cos() - bb_n*sin() + j*(bb_n*cos()+aa_n*sin())
<p>Real(II) = Real(A_o) + sum_n(aa_n*cos(2*pi*n*t/T)-bb_n*sin(2*pi*n*t/T))
<p>Comparing to the first expansion we see that

<p>Real(A_o)=a_o,   aa_n=a_n,   -bb_n=b_n
<p>To me, this proves existence of the complex expansion. Knowing one,
you

can figure out the other. Part #1 is complete.
<p>Part #2: IDL form of complex expansion
<p>Let f(t) be a periodic function with period T defined on an interval

[0,T].
<p>Then there exist complex A_n such that
<p>f(t)= sum_n(A_n*exp(j*2*pi*n*t/T)), n=0,1,2,...   j*j = -1
<p>Divide the interval into N sections. t~t_i = i*T/N

Then,

f(t_i) = sum_n(A_n*exp(j*2*pi*n*t_i/T))

            =
sum_n(A_n*exp(j*2*pi*n*i*(T/N)/T))

            = sum_n(
A_n*exp(j*2*pi*n*i/N)
) , n=0,1,...
<p>This is exactly what is found in the IDL manual under the section for

FFT. The only difference is that t has been replaced by u and A_n has

been replaced by F(u). Note that the period T has dropped out. Also
note

that t has been replaced by t_i = i*T/N. In order for this to
happen,

the interval over which t is defined must be from [0,T]. This is

different from the definition of t being defined over the interval

[-T/2,T/2]. Perhaps this is why b_n = -bb_n.
<p>********UNFORTUNATELY IT IS WRONG***************
<p>What is wrong is the values of n in the sum. IDL does not use the values

of n=0,1,2,... IDL actually uses n= -N/2+1, -N/2+2, ...-1,0,1,...,N/2

The reason for doing this must have to do with FFT theory. Note also

that the number of values of n is N.
<p>It gets more complicated. From the manual we have
<p>F(u) = 1/N*sum_x(f(x)*exp(-j*2pi*ux/N)) , x=0,1,...N-1
<p>First thing to realize is that F(u) is really F_n. Where n is an

integer. This comes from the fact that f(x) is periodic in x.
<p>The manual also mentions that the "frequencies" are

Fo, 1/(NT),2/(NT),...,1/2T,-(N-2)/(2NT),...,-1/NT
<p>After trial and error I have determined that the value of the ns range

for -N/2 to N/2. Futhermore, the F_n are stored in the order associated

with the following values of n
<p>0,1,2,...,N/2,-(N/2-1),-(N/2-1),...,-1 <== this is bizarre!!
<p>Let N=8. Then N/2=4
<p>The F_n would be stored in an array. The array of n values associated

with this array would be:
<p>[0,1,2,3,4,-3,-2,-1]
<p>Part #3: Specific Example
<p>Consider the interval t = [0,1]. This choice of interval implies T=1.

Let f(t) = sin (4*pi*t)

```
<p>f(t_i)=sin(2pi*2*i/N), i=0,1,...N
<p>f(t_i)=sum_n(A_n*exp(-j*2pi*n*i/N)) , n=-N/2,...-1,0,1,...N/2
<br>        = A_nN/2... +
A_n2*(cos(2pi*(-2)*i/N)+j*sin(2pi*(-2)*i/N))+
<br>              
             
  
+ A_o+A_n1*exp()+A_1*exp()+
<br>              
             
     
A_2*(cos(2pi*(2)*i/N)+j*sin(2pi*(2)*i/N))
<br>+ A_3*exp()+...
<br>        =... + A_n2*cos(2pi*2*i/N)+A_2*cos(2pi*2*i/N)
+
<br>              
  
+A_n2*j*(-1)*sin(2pi*2*i/N)+A_2*j*sin(2pi*2*i/N)) + ....
<br>        = ... + (A_n2+A_2)*cos(2pi*2*i/N)+j*(
-A_n2 + A_2)*sin(2pi*2*i/N)
<br>+ ...
<p>where A_n2 stands for A_n where n= -2
<p>Equating the series to sin(2pi*2*i/n) we conclude
<p>A_n = 0 for all n except  n = -2 or n = 2.
<p>A_n2+A_2=0
<br>j*(-A_n2 + A_2) = 1
<p>Let A_n2=(a_n2+j*b_n2) and A_2=(a_2 + j*b_2)
<p>The above equations imply
<p>(a_n2 + a_2)   + j*(  b_n2+b_2) = 0  &amp;
<br>j*[( -a_n2 + a_2) + j*( -b_n2 + b_2)] = 1
<br>==> a_n2 + a_2=0, b_n2+b_2 =0 ==> a_n2= - a_n2, b_n2 = -b_n2
<p>==> 2*a_n2=0 ==> a_n2=a_2 = 0
<br>==> j*j*(2*b_2)=1 ==> 2*b_2 = -1/2, b_2 = 1/2
<p>A_n2 = 0 + j*(1/2)
<p>A_2  =  0 + j*(-1/2)
<p>We now have calculated the solutions.
<p>The following code calculates this and displays the correct answers.
It
<br>shows how to plot A_n vs n correctly.
<p>;idl_program fft_sine.pro
<br>TT=1
<br>Npts=100
<br>t=findgen(Npts)/(Npts-1)*TT
<br>f_t=sin(4.*!pi*t/TT)
<br>!p.multi=[0,2,3]
<br>plot,t,f_t, title='f(t) vs t'
<br>A_n=fft(f_t,-1) ; complex fourier coefficients
<p>plot,float(A_n),yrange=[-.5,.5],title='float(A_n)'
```

```
<br>plot,imaginary(A_n),yrange=[-.5,.5],title='imaginary(A_n)'
<p>a=findgen(Npts/2+1)
<br>b=-reverse(findgen(Npts/2-1)+1)
<br>c=[a,b] ; c=[-N/2+1,-N/2+2, ...,-1,0,1,...,N/2]
<br>print,c
<br>sub=sort(c)
<br> plot,c(sub),float(A_n(sub)),yrange=[-.5,.5],title='float(A_n ) vs n'
<br>plot,c(sub),imaginary(A_n(sub)),yrange=[-.5,.5], $
<br>          title='imaginary(A_n) vs
n'
<br>plot,c(sub),imaginary(A_n(sub)),xrange=[-5,5],$
<br>          title='imaginary(An) vs
n' ; finer x scale
<br>end
<br> <b><font size=+2></font></b>
<p><b><font size=+2>Part #4 Specific Example:</font></b>
<br><b><font size=+2>T(x,y)= sin(6pi*x) + cos(4pi*y)</font></b>
<p>Hopefully the program below is somewhat self explantory.
<br>It calculates the C=FFT(T,-1)
<br> 
<p>;idl_program fft_2D.pro
<br>!p.multi=0
<br>Tx=1
<br>Nx=100
<br>x=findgen(Nx)/(Nx-1)*Tx
<br>Ty=1
<br>Ny=100
<br>y=findgen(Ny)/(Ny-1)*Ty
<br>T=fltarr(Nx,Ny)
<br>for i=0,Nx-1 do begin
<br>   for j=0,Ny-1 do begin
<br>      T(i,j)= sin(2.*!pi*3.*i/Nx) + cos(2.*!pi*2*j/Ny)
<br>      endfor
<br>   endfor
<br>;
<br>!p.multi=[0,2,2]
<br>shade_surf,T,x,y,xtitle='x',ytitle='y',title='T(x,y)'
<br>;
<br>;
<br>C=fft(T,-1) ; complex fourier coefficients
<br>surface,float(C)
<br>aaa=where(float(c) gt .4)
<br>surface,imaginary(C)
<br>;
<br>a=findgen(Nx/2+1)
<br>b=-reverse(findgen(Nx/2-1)+1)
<br>ns=[a,b] ; this is the array of n's associated with C(n). n goes with
x
```

```
<br>;print,ns ; n goes from 0,...,Nx/2, -(Nx/2-1),...,-1
<br>subn=sort(ns) ;  n goes with x
<br>n_sort=ns(subn)
<br>;
<br>a=findgen(Ny/2+1)
<br>b=-reverse(findgen(Ny/2-1)+1)
<br>ms=[a,b] ; this is the array of m's associated with C(n,m)
<br>print,ms ; m goes from 0,...,Ny/2, -(Ny/2-1),...,-1
<br>subm=sort(ms) ; m goes with y
<br>m_sort=ms(subm)
<br>;
<br>sub_n_p3=where(ns eq 3)
<br>sub_n_n3=where(ns eq -3)
<br>sub_n_0=where(ns eq 0)
<br>;
<br>sub_m_p2=where(ms eq 2)
<br>sub_m_n2=where(ms eq -2)
<br>sub_m_0=where(ms eq 0)
<br>;
<br> print,'C(3,0),c(-3,0)=',C(sub_n_p3,sub_m_0),C(sub_n_n3,sub_m _0)
<br>;
<br> print,'C(0,2),c(0,-2)=',C(sub_n_0,sub_m_p2),C(sub_n_0,sub_m_ n2)
<br>;
<br>; now we need to define CC(n,m) to have normal scaling in n &amp; m.
<br>;
<br>CC=C*0.
<br>;
<br>for n=0,Nx-1 do begin
<br>   for m=0,Ny-1 do begin
<br>      CC(subn(n),subm(m))= C(n,m)
<br>      endfor
<br>   endfor
<br>;
<br>surface,ABS(CC),n_sort,m_sort
<br>;
<br>end</html>
```

Subject: Re: Secrets FFTs revealed!!
Posted by Paul van Delst on Fri, 05 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Peter Brooker wrote:

```
>
  <snip>
>
>  Divide the interval into N sections. t~t_i = i*T/N
>  Then,
```

> f(t_i) = sum_n(A_n*exp(j*2*pi*n*t_i/T))
> = sum_n(A_n*exp(j*2*pi*n*i*(T/N)/T))
> = sum_n(A_n*exp(j*2*pi*n*i/N)) , n=0,1,...
>
> This is exactly what is found in the IDL manual under the section for
> FFT. The only difference is that t has been replaced by u and A_n has
> been replaced by F(u). Note that the period T has dropped out. Also note
> that t has been replaced by t_i = i*T/N. In order for this to happen,
> the interval over which t is defined must be from [0,T]. This is
> different from the definition of t being defined over the interval
> [-T/2,T/2]. Perhaps this is why b_n = -bb_n.
>
> ********UNFORTUNATELY IT IS WRONG***************
>
> What is wrong is the values of n in the sum. IDL does not use the values
> of n=0,1,2,... IDL actually uses n= -N/2+1, -N/2+2, ...-1,0,1,...,N/2
> The reason for doing this must have to do with FFT theory. Note also
> that the number of values of n is N.

Great job on the ref but I have always found it hard to read ASCII
equations. :o)

The input to the FFT should include BOTH the positive and negative
frequencies. So if you have a function of N+1 points, then you want to
supply the FFT with 2N points (or if you have a function of N/2 + 1
points you supply it with N points. The "+1"th point is the Nyquist
point.

Say you have the following real function (e.g. an interferogram):

```
        ZPD        Nyquist point
         |            |
         |     +ve     |
         |<---optical--->|
         |    delays    |
         V             |
                      |
         |            |
         |            |
         ||    |       V
         |||  |||  ||
         ||||||||||||||||
         |||  |||  ||
         ||    |
         |
         |
```

where ZPD == zero path difference. If you want to FFT this function to a spectrum, the input to the FFT should be:

```
        ZPD         Nyquist point
         |             |
         |     +ve     |     -ve
         |<---optical--->|<---optical--->
         |    delays   |    delays
         V              |
                       |
         |             |
         |             |
         ||    |          V          |    |
         |||  |||  ||         ||  |||  ||
         ||||||||||||||||||||||||||||||||
         |||  |||  ||         ||  |||  ||
         ||    |                  |    |
         |
         |
```

where the function values at negative optical delays are simply the reflected positive ones. Note that neither the Nyquist point nor the ZPD point is reflected - the reflection occurs *around* the Nyquist point. The ZPD (or zero frequency point) is unique and the Nyquist point is ambiguous, i.e. it contains both +ve and -ve frequency info.

The IDL FFT documentation mentions the storage of the negative frequencies. It's just that given a real function (e.g. something measured), one simply repeats the +ve frequency values into -ve frequencies.

So, I don't think the documentation is wrong but the most I would be willing to bet is a beer or two (due to my lack of understanding, not RSI's).

Here a section of the header to my fft_to_interferogram.pro mentioning the reflection. Check out the input/output array sizes in the example section:

```
; PROCEDURE:
;       The input spectrum is reflected about it highest frequency
;       (largest wavenumber). The spectrum is FFT'd and the
;       resultant interval is scaled by the input spectrum wavenumber
;       interval.
;
;       The interferogram is then shifted so that the ZPD occurs at the
;       centre (like a measured IFG) and the redundant most negative
;       Nyquist point is placed at the end of the interferogram array.
```

```
;
; EXAMPLE:
;       Given a spectrum:
;
;         IDL> HELP, spc, v
;         SPC             FLOAT     = Array[12445]
;         V               FLOAT     = Array[12445]
;
;       The Fourier transform can be found by typing:
;
;         IDL> PRINT, fft_to_interferogram( spc, v, ifg, opd )
;                1
;
;       resulting in an interferogram and optical delay grid:
;
;         IDL> HELP, ifg, opd
;         IFG             DOUBLE    = Array[24889]
;         OPD             DOUBLE    = Array[24889]
;
```

and here's the code snippet that actually does the reflection, FFT'ing. Take note of the bounds of the REVERSE'd protion of "spectrum" - no zero frequency (point 0) and no Nyquist frequency (point n_spectrum_pts-1) reflection:

```
 ;--------------------------------------------------------- -------------------
;               -- Fourier transform the input spectrum --
 ;--------------------------------------------------------- -------------------

; --------------------
; Reflect the spectrum
; --------------------

  spectrum_to_fft = TEMPORARY( [ spectrum, REVERSE( spectrum[ 1:
n_spectrum_pts - 2 ] ) ] )


; ----------------
; FFT the spectrum
; ----------------

  interferogram = FFT( TEMPORARY( spectrum_to_fft ), /DOUBLE, /INVERSE )
```

cheers,

paulv
--

Paul van Delst Ph: (301) 763-8000 x7274
CIMSS @ NOAA/NCEP Fax: (301) 763-8545
Rm.202, 5200 Auth Rd. Email: pvandelst@ncep.noaa.gov
Camp Springs MD 20746