Subject: FFT secrets revealed!! Update 6/5/2000
Posted by Peter Brooker on Mon, 05 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
Below is a repeat of my orginal post to the group in which I tried to explain
FFT use

from the point of view of somebody who has little experience of FFT
algorithms but

does understand the concept of Fourier series as well as Fourier transforms.
This update

includes a large modification to part 1 that develops the complex expansion
from the

series expansion. In it I show how all the imaginary components

of the complex FFT expression cancel to zero. Modifications were also
made to part2.
<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) =sin(6pi*x) + cos(4pi*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>The coefficients are given by:
<p>a_o = 2/T*integral(f(t)*dt)
<p>a_n =2/T*integral(f(t)*cos(2*pi*n*t/T)*dt)
<p>b_n =2/T*integral(f(t)*sin(2*pi*n*t/T)*dt)
<p>The limits of the integration are from -T/2 to T/2.
<p>This is just the Fourier series of function with period T. Nothing
new

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

Kreyszig. This expansion though assumes -T/2 < t < T/2
<p>Let's now define the complex coefficients:
<p>c_o = a_o

c_n = 1/2*(a_n + j*b_n)
<p>Then for n = 1,2,...
<p>c_n = 1/2*(a_n + j*b_n)

      = 2/T*1/2*(integral(f(t)*cos(2*pi*n*t/T)*dt
) +

j*2/T*integral(f(t)*sin(2*pi*n*t/T)*dt))

     = 1/T*integral(f(t)*[cos(2*pi*n*t/T) + j*sin(2*pi*n*t/T)]*dt)

     = 1/T*integral(f(t)*exp(j*2*pi*n*t/T)*dt
)
<p>Since exp(0)=1 we can write:
<p>c_n = 1/T*integral(f(t)*exp(j*2*pi*n*t/T)*dt) , n= 0,1,2,...
<p>Note also that [(c_n)^*] = 1/T*integral(f(t)*exp(-j*2*pi*n*t/T)*dt
)

= 1/T*integral(f(t)*exp(j*2*pi*(-n)*t/T)*dt)

=c_(-n)
<p>Here [(c_n)^*] denotes the complex conjugate of the element inside (
)

(Sorry for the clumsy notoation!!!)
<p>Now here is something completely different ...
<p>Given the definition of c_n above:
<p>f(t)=sum_n(c_n*exp(-j*2*pi*n*t/T)), n= ...-2,-1,0,1,2,...
<p>Proof:
<p>Define AA and BB by:
<p>sum_n(c_n*exp(-j*2*pi*n*t/T)) = AA + c_o + BB,
<p>where AA = sum_n(c_n*exp(-j*2*pi*n*t/T)), n=...,-2,-1

             BB = sum_n(
c_n*exp(-j*2*pi*n*t/T)), n=1,2,....
<p>Let's work on AA some more:
<p>AA = sum_n(c_n*exp(-j*2*pi*n*t/T)), n=...,-2,-1

        =sum_n(c_(-n)* exp(-j*2*(-n)*t/T)),
n = 1,2,....
<p>Now c_(-n) = [(c_n)^*] and exp(-j*2*(-n)*t/T) = [(exp(-j*2*pi*n*t/T)
)^*]
<p>therefore,
<p>c_(-n)* exp(-j*2*(-n)*t/T) = [(c_n)^*] * [(exp(-j*2*pi*n*t/T))^*]

= [(c_n* exp(-j*2*pi*n*t/T))^*]
<p>Using this we have;
<p>AA =sum_n([c_n*exp(-j*2*n*t/T)]^*), n=1,2,...

          =[(sum_n(
c_n*exp(-j*2*n*t/T)
)   )]^*
<p>Comapring this expression to sum BB we see that AA = [(BB)^*]
<p>Let's write the original sum again with this information:
<p>sum_n(c_n*exp(-j*2*pi*n*t/T)) = AA + c_o + BB = [(BB)^*] + c_o +BB

          = c_o + 2*Re(BB)

<p>Now BB= sum_n(c_n*exp(-j*u)), n=1,2,.... and u = 2*pi*n*t/T

= sum_n(1/2*(a_n+j*b_n)*(cos(u) - j*sin(u))

= 1/2*sum_n(a_n*cos(u) + j*(-j)*b_n*sin(u) +j*(b_n*cos(u) -a_n*sin(u)
)

= 1/2*sum_n(a_n*cos(u) + b_n*sin(u) +j*(b_n*cos(u) -a_n*sin(u))
<p>From this we see that:
<p>2*Re(BB) = 2*1/2*sum_n(a_n*cos(u) + b_n*sin(u))

= sum_n(a_n*cos(u) + b_n*sin(u))
<p>Now we have:
<p>sum_n(c_n*exp(-j*2*pi*n*t/T)) = c_o + 2*Re(BB)

= a_o + sum_n(a_n*cos(u) + b_n*sin(u)), n= 1,2,...
<p>This is exactly the definition for f(t). I have therefore proved that
<p>f(t) = sum_n(c_n*exp(-j*2*pi*n*t/T)) , n= ...,-2,-1,0,1,2,...

<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=...,-1,0,1,...   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=...,-1,0,1,...
<p>What about the ns?
<p>IDL truncates the sum from -N/2+1, -N/2 + 2, ..., -1,0,1,...,N/2
<p>This is how IDL performs the FFT. This 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].
<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

        = A_nN/2... +
A_n2*(cos(2pi*(-2)*i/N)+j*sin(2pi*(-2)*i/N))+

+ A_o+A_n1*exp()+A_1*exp()+

A_2*(cos(2pi*(2)*i/N)+j*sin(2pi*(2)*i/N))

+ A_3*exp()+...

        =... + A_n2*cos(2pi*2*i/N)+A_2*cos(2pi*2*i/N)
+

+A_n2*j*(-1)*sin(2pi*2*i/N)+A_2*j*sin(2pi*2*i/N)) + ...

        = ... + (A_n2+A_2)*cos(2pi*2*i/N)+j*(
-A_n2 + A_2)*sin(2pi*2*i/N)

+ ...
<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

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 &

j*[(-a_n2 + a_2) + j*(-b_n2 + b_2)] = 1

==> 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

==> 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

shows how to plot A_n vs n correctly.
<p>;idl_program fft_sine.pro

TT=1

Npts=100

t=findgen(Npts)/(Npts-1)*TT

f_t=sin(4.*!pi*t/TT)

!p.multi=[0,2,3]

plot,t,f_t, title='f(t) vs t'

A_n=fft(f_t,-1) ; complex fourier coefficients
<p>plot,float(A_n),yrange=[-.5,.5],title='float(A_n)'

plot,imaginary(A_n),yrange=[-.5,.5],title='imaginary(A_n)'
<p>a=findgen(Npts/2+1)

b=-reverse(findgen(Npts/2-1)+1)

c=[a,b] ; c=[-N/2+1,-N/2+2, ...,-1,0,1,...,N/2]

print,c

sub=sort(c)

 plot,c(sub),float(A_n(sub)),yrange=[-.5,.5],title='float(A_n) vs n'

plot,c(sub),imaginary(A_n(sub)),yrange=[-.5,.5], $

          title='imaginary(A_n) vs
n'

plot,c(sub),imaginary(A_n(sub)),xrange=[-5,5],$

          title='imaginary(An) vs
n' ; finer x scale

end

<p>Part #4 Specific Example:2D

T(x,y)= sin(6pi*x) + cos(4pi*y)
<p>Hopefully the program below is somewhat self explantory.

It calculates the C=FFT(T,-1)

<p>;idl_program fft_2D.pro

!p.multi=0

Tx=1

Nx=100

x=findgen(Nx)/(Nx-1)*Tx

Ty=1

Ny=100

y=findgen(Ny)/(Ny-1)*Ty

T=fltarr(Nx,Ny)

for i=0,Nx-1 do begin

   for j=0,Ny-1 do begin

      T(i,j)= sin(2.*!pi*3.*i/Nx) + cos(2.*!pi*2*j/Ny)

      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>;
```


surface,ABS(CC),n_sort,m_sort

;

end</html>