Fourier Analysis

James Emery

3/18/2008

Contents

1 Fourier Series

The sequence of trigonometric functions

$$\{\phi_n\} = \{1/\sqrt{T}, \sqrt{2/T}\cos(\omega t), \sqrt{2/T}\sin(\omega t), \sqrt{2/T}\cos(2\omega t), \sqrt{2/T}\sin(2\omega t)....., \\ ....., \sqrt{2/T}\cos(n\omega t), \sqrt{2/T}\sin(n\omega t), ...\}$$

of period

$$T = \frac{2\pi}{\omega},$$

form an orthonormal sequence on any interval of length T.

The inner product of two functions f and g is defined as

$$(f, g) = \int_0^T f(t)g(t)dt.$$

The trigonometric functions are orthonormal in the sense that

$$(\phi_n, \phi_n) = 1,$$

and if $n \neq m$ then

$$(\phi_n, \phi_m) = 0.$$

This orthonormal property of the sequence of trigonometric functions follows from a few facts, two of which are

$$\int_0^T \cos(n\omega t)^2 dt = \int_0^T \sin(n\omega t)^2 dt$$

and

$$\int_0^T (\cos(n\omega t)^2 + \sin(n\omega t)^2) dt = \int_0^T dt = T.$$

So we see that either of the above integrals is equal to $T/2$. Continuing with the facts, we have

$$\int_0^T \cos(n\omega t) \sin(m\omega t) dt = 0.$$

And when n is not equal to m then

$$\int_0^T \cos(n\omega t) \cos(m\omega t) dt = 0,$$

and again

$$\int_0^T \sin(n\omega t) \sin(m\omega t) dt = 0.$$

These properties follow easily by replacing the trigonometric functions by their complex exponential equivalents. The coefficients of a Fourier series

$$f(t) = a_0/2 + a_1 \cos(\omega t) + b_1 \sin(\omega t) + a_2 \cos(2\omega t) + b_2 \sin(2\omega t) + ...,$$

are determined by computing the inner product of f with each of the orthonormal functions. Thus if we write

$$f = \alpha_1 \phi_1 + \alpha_2 \phi_2 + ... + \alpha_n \phi_n + ...,$$

then

$$(f, \phi_n) = \alpha_n (\phi_n, \phi_n) = \alpha_n.$$

Therefore, for the Fourier series, we have

$$a_n = \frac{2}{T} \int_0^T f(t) \cos(n\omega t) dt.$$

and

$$b_n = \frac{2}{T} \int_0^T f(t) \sin(n\omega t) dt.$$

2 Convergence

There exists a continuous function on a closed interval whose Fourier series is nowhere convergent.

However, there are conditions that guarantee that a Fourier series converges. For example: If a function is Legesque integrable, and at a point x, both continuous from the right, and continuous from the left, then the series converges to the mean of the left and right function limits at x. (see Goldberg **The Elements of Real Analysis**.)

3 The L^2 Theory of Fourier Series

See Goldberg **The Elements of Real Analysis**.

4 The Fourier Transform

The Fourier transform of the function f is defined as (Goldberg, **The Fourier Transform**)

$$g(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt.$$

By the Fourier integral theorem

$$f(t) = \int_{-\infty}^{\infty} g(\omega)e^{i\omega t}d\omega.$$

Note. The transform is defined slightly differently by various authors:
Schaum's Mathematical Handbook:

$$g(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt.$$

Then the inverse transform is

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} g(\omega)e^{i\omega t}d\omega.$$

The Fourier Integral and Its Applications by Athanasios Papoulis:

$$g(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(t)e^{i\omega t}dt.$$

Then the inverse transform is

$$f(t) = \int_{-\infty}^{\infty} g(\omega) e^{-i\omega t} d\omega.$$

Quantum Mechanics by Powell and Crasemann:

$$g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{i\omega t} dt.$$

Then the inverse transform is

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega) e^{-i\omega t} d\omega.$$

Digital Signal Processing in Telecommunications by Kishan Shenoi:

$$g(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi\omega t} dt.$$

Then the inverse transform is

$$f(t) = \int_{-\infty}^{\infty} g(\omega) e^{i2\pi\omega t} d\omega.$$

Including the 2π factor in the exponential argument eliminates the multiplier in front of both integrals.

5 The Discrete Fourier Transform

Suppose f has its support in the interval $[0, T]$. Suppose that we divide this interval into N equal pieces, where integer N is even. Let

$$\Delta t = \frac{T}{N}.$$

Define $t_p = p\Delta t$ for $p = 0$ to $N - 1$. Let the fundamental angular frequency be $\Delta\omega = \frac{2\pi}{T}$ and define $\omega_q = q\Delta\omega, q = 0, .., N-1$. A Riemann approximation to g

$$g(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt.$$

is

$$G(\omega) = \frac{1}{2\pi} \sum_{p=0}^{N-1} f(t_p) e^{-i\omega t_p} \Delta t.$$

Define

$$G_q = G(\omega_q),$$

and

$$F_p = f(t_p).$$

The mapping

$$\{F_p\} \to \{G_q\},$$

is called the discrete Fourier Transform. We have

$$t_p \omega_q = \frac{pT}{N} \frac{q2\pi}{T} = \frac{2\pi}{N} pq,$$

so

$$G_q = \frac{T}{2\pi N} \sum_{p=0}^{N-1} F_p (e^{i2\pi/N})^{-pq} = \frac{T}{2\pi N} \sum_{p=0}^{N-1} F_p W^{-pq},$$

where $W = e^{i2\pi/N}$ is the principal Nth root of unity.
The Discrete Fourier Integral Theorem. Let

$$G_n = \frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k e^{-i\omega_n t_k}$$

$$= \frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k W^{-nk},$$

where $W = e^{i2\pi/N}$. Then

$$f(t_j) = F_j = \frac{2\pi}{T} \sum_{n=0}^{N-1} G_n W^{nj}.$$

Proof.

$$\frac{2\pi}{T} \sum_{n=0}^{N-1} G_n W^{nj}$$

$$= \frac{2\pi}{T} \sum_{n=0}^{N-1} \left(\frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k W^{-nk} \right) W^{nj}$$

$$= \frac{1}{N} \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} F_k W^{(j-k)n}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} F_k \sum_{n=0}^{N-1} (W^{j-k})^n$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} F_k \delta_j^k N = F_j.$$

This follows because when $j = k$, then

$$\sum_{n=0}^{N-1} (W^{j-k})^n = N,$$

and when $j \neq k$, then

$$\sum_{n=0}^{N-1} (W^{j-k})^n = (1 - (W^{j-k})^N)/(1 - W^{j-k}) = 0.$$

The latter equation follows from the expression for the sum of a geometric series, and from the fact that W^{j-k} is an Nth root of unity.

6 The Matlab FFT

The Matlab function fft(), computes essentially

$$G_q^{ML} = \sum_{p=0}^{N-1} F_p (e^{i2\pi/N})^{-pq},$$

where $q = 0, ..., N-1$ and $p = 0, ..., N-1$. This is our discrete fourier transform defined above, without the multiplication factor $\frac{T}{2\pi N}$.

$$G_q = \frac{T}{2\pi N} \sum_{p=0}^{N-1} F_p (e^{i2\pi/N})^{-pq}$$

$$= \frac{T}{2\pi N} G_q^{ML}.$$

Because MatLab does not allow 0 indices, it actually computes

$$G_q^{ML} = \sum_{p=1}^{N} F_p (e^{i2\pi/N})^{-(p-1)(q-1)}.$$

where $1 \le q \le N$ and $1 \le p \le N$. So

$$\frac{T}{2\pi N} G_q^{ML}$$

is an approximate value of the continuous Fourier transform

$$g(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt.$$

at sample point ω_q.

The Fourier Approximation Example on p229, of the book "Mastering Matlab 5," runs without change in Octave. The Matlab fft function computes the same fft as is in my paper foran.tex for the discrete Fourier transform, except for the constant multiplier. In this approximation example the same constant multiplier is applied to the Matlab fft output to get the approximation that is graphed. Because the function is real, the magnitude of the transform is symmetric about the t=0 axis, that is, it is an even function. The Nyquist frequency is 1/2 the sampling frequency. So we compute only the lower half frequences to avoid aliasing.

7 The IMSL FFT

The IMSL 1978 Fortran routine fft2c does the same computation as the MatLab function fft described in the previous section:

```
remarks  1.  fft2c computes the fourier transform, x, according
c                 to the following formula;
c
c                   x(k+1) = sum from j = 0 to n-1 of
c                             a(j+1)*cexp((0.0,(2.0*pi*j*k)/n))
c                   for k=0,1,...,n-1 and pi=3.1415...
```

8 Fourier Interpolation

Proposition. If f is real, then

$$g(-\omega) = \overline{g(\omega)}.$$

Corollary The magnitude of the transform of a real function is symmetric.
Proposition. If $\{F_k\}$ is real, then

$$G_q = \overline{G}_{N-q}$$

Proof.

$$\begin{aligned} G_{N-q} &= \frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k W^{-(N-q)k} \\ &= \frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k (W^N)^{-k} W^{qk} \\ &= \frac{T}{2\pi N} \sum_{k=0}^{N-1} F_k W^{qk} \\ &= \overline{G}_q. \end{aligned}$$

Proposition. If f is real and N is even, then the trigonometric sum

$$\tilde{f}(t) = \frac{a_0}{2} + a_1 \cos(\omega_1 t) + b_1 \sin(\omega_1 t) + ...$$

$$+a_{K-1} \cos(\omega_{K-1} t) + b_{K-1} \sin(\omega_{K-1} t)$$

$$+a_K \cos(\omega_K t) + b_K \sin(\omega_K t).$$

interpolates f at each t_p. That is,

$$f(t_p) = F_p = \tilde{f}(t_p).$$

The coefficients in the function $\tilde{f}$ are defined as

$$K = N/2,$$

$$a_0 = \frac{G_0 4\pi}{T},$$

and for $j = 1, .., K-1$,

$$a_j = \frac{\Re(G_j)4\pi}{T},$$

$$b_j = \frac{-\Im(G_j)4\pi}{T}.$$

The last coefficients are

$$a_K = \frac{\Re(G_K)2\pi}{T},$$

and

$$b_K = \frac{-\Im(G_K)2\pi}{T}.$$

Proof. The proof consists in showing that at each t_p, the inverse transform $F(t)$ equals the trigonometric sum.

By the discrete Fourier integral theorem, we have

$$f(t_p) = F_p = \frac{2\pi}{T} \sum_{q=0}^{N-1} G_q W^{pq}$$

$$= \frac{2\pi}{T} \sum_{q=0}^{N-1} G_q W^{pq}$$

$$= \frac{2\pi}{T}[G_0 + (G_1 W^{1p} + G_{N-1} W^{(N-1)p})$$

$$+(G_2 W^{2p} + G_{N-2} W^{(N-2)p})$$

$$+...$$

$$+(G_{K-1} W^{(K-1)p} + G_{K+1} W^{(K+1)p})$$

$$+G_K W^{Kp}$$

$$= \frac{2\pi}{T}\left[G_0 + \sum_{q=1}^{K-1}(G_q W^{pq} + \overline{G_q W^{pq}}) + G_K W^{Kp}\right]$$

$$= \frac{2\pi}{T}\left[G_0 + \sum_{q=1}^{K-1} 2\Re(G_q W^{pq}) + G_K W^{Kp}\right].$$

Let

$$G_q = \alpha_q + i\beta_q.$$

We have

$$\Re(G_q W^{pq}) = \Re[(\alpha_q + i\beta_q)(\cos(\omega_q t_p) + i\sin(\omega_q t_p))].$$

$$= \alpha_q \cos(\omega_q t_p) - \beta_q \sin(\omega_q t_p)$$

$$= \Re(G_q)\cos(\omega_q t_p) - \Im(G_q)\sin(\omega_q t_p).$$

Now each $F_p = f(t_p)$ is real. Then

$$G_0 = \frac{T}{2\pi N}\sum_{k=0}^{N-1} F_k$$

is real. Therefore the equation forces $G_K W^{Kp}$ to be real. So

$$G_K W^{Kp} = \Re(G_K)\cos(\omega_K t_p) - \Im(G_K)\sin(\omega_K t_p).$$

This finishes the proof.
Note: $W^K = -1$. $\tilde{f}(t)$ is not equal to

$$F(t) = \frac{2\pi}{T}\sum_{n=0}^{N-1} G_n e^{i\omega_n t},$$

for all t, because $F(t)$ may be complex. A program for doing this interpolation is given in the next section.

9 A Fourier Interpolation Program

Program **frintrp.ftn** uses the Fast Fourier Transform to compute an interpolating trigonometric sum.

```
c frintrp.ftn modification of fourier.for 3/14/95
      implicit real*8 (a-h,o-z)
      logical iszero
      integer gfile
      complex*16 g(1024)
      dimension iwk(11)
      dimension frq(1025),amod(1025),arg(1025)
      dimension a(1024),b(1024)
      dimension ain(10)
      character*80 cs
      data pi/3.14159265358979d0/
      one=1.
      zero=0.
```

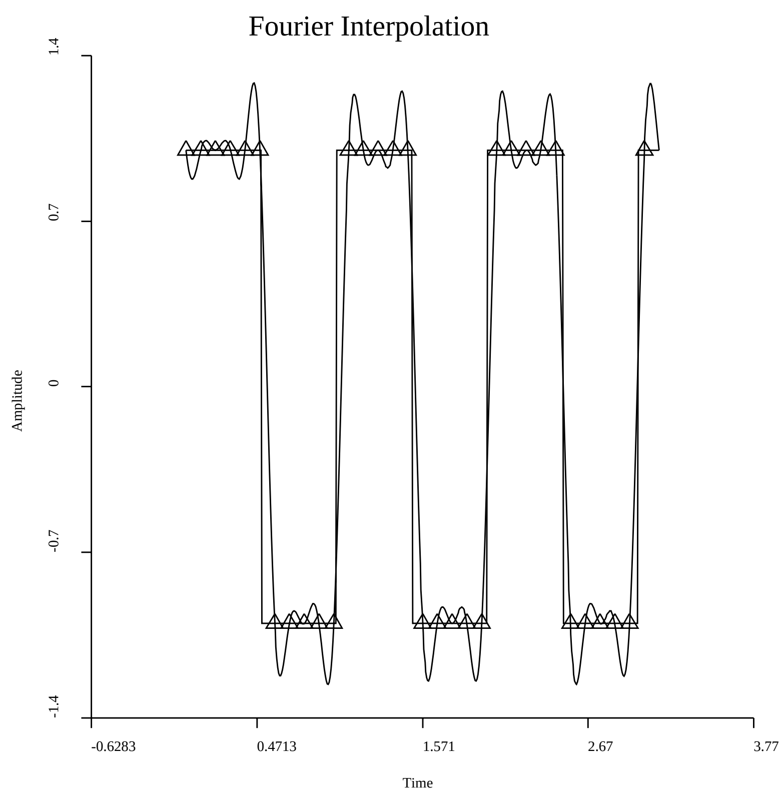

Figure 1: Interpolating a step function with program **frintrp.ftn**.

```
      per=1.
      m=3
      gfile=3
      open(gfile,file='p.gi',status='unknown')
1     continue
      cs='(1)Enter period and number of sample points'
      write(*,*)cs(1:lenstr(cs))
      write(*,2)
2     format(
     +'(2)List transform modulus.'/
     +'(3)List fourier coefficients.'/
     +'(4)Plot function and fourier approximation.'/
     +'(5)Plot transform modulus.'/
     +'(6)Stop.'/
     +)
      call readr(0,ain,nr)
      nm=ain(1)
      if(nm.eq.1)go to 9
      if(nm.eq.2)go to 33
      if(nm.eq.3)go to 33
      if(nm.eq.4)go to 33
      if(nm.eq.5)go to 33
      if(nm.eq.6)stop
      go to 1
9     continue
      write(*,*)'Enter period [3.14159]'
      call readr(0,ain,nr)
      if(nr .eq. 0)then
       per=3.1415926535
      else
       per=ain(1)
      endif
      write(*,*)' Number of sample points = 2^m. Enter m [5]'
      call readr(0,ain,nr)
      if(nr .eq. 0)then
       m=5
      else
       m=ain(1)
      endif
      go to 1
33    continue
      n=2**m
      write(*,*)' Sample Points: '
      do 30 i=1,n
       t=(per/n)*(i-1)
       g(i)=cmplx(f(t),zero)
       write(*,*)i,g(i)
30    continue
      do 40 i=1,n
      g(i)=conjg(g(i))
40    continue
      call fft2c(g,m,iwk)
      write(*,*)' Transform: '
      do 45 i=1,n
       write(*,*)i,g(i)
```

```
45    continue
      gmax=0.
      do 50 i=1,n
      gm=abs(g(i))
      if(gm.gt.gmax)gmax=gm
50    g(i)=conjg(g(i))*per/(2*pi*n)
      del=gmax*1.e-10
      do 52 i=1,n
      am=abs(g(i))
      if(am .lt. del)g(i)=zero
52    continue
      if(nm.eq.2)go to 55
      if(nm.eq.3)go to 79
      if(nm.eq.4)go to 79
55    k=n+1
      if(nm.eq.2)print 60
60    format(' Freq. (cy/sec)   Modulus   Argument (degrees)')
      amodmx=0.
      do 70 i=1,k
      j=i-(n/2+1)
      l=iabs(j)+1
      frq(i)=j/per
      amod(i)=abs(g(l))
      if(amod(i).gt.amodmx)amodmx=amod(i)
      iszero = amod(i) .eq. zero
      arg(i)=0.
      if(.not. iszero)arg(i)=dimag(log(g(l)))*180./pi
      if(j.lt.0)arg(i)=-arg(i)
      if(nm.eq.2)print 75,frq(i),amod(i),arg(i)
75    format(3(1x,g15.8))
70    continue
      if(nm.eq.2)go to 1
      call xvwpor(gfile,-one,one,-one,one)
      xmn=-frq(k)
      xmx=frq(k)
      ymn=0.
      ymx=amodmx
      call xwindo(gfile,xmn,xmx,ymn,ymx)
      call xmove(gfile,frq(1),amod(1))
      do  77 i=2,k
      call xdraw(gfile,frq(i),amod(i))
 77   continue
      go to 1
79    continue
      l=n/2-1
      a0=real(g(1))*2*pi/per
      cmx=abs(a0)
      do 80 j=1,l
      am=real(g(j+1))*4*pi/per
      bm=-dimag(g(j+1))*4*pi/per
      a(j)=am
      b(j)=bm
      if(cmx.lt.am)cmx=am
      if(cmx.lt.bm)cmx=bm
80    continue
```

```
      a(l+1)=real(g(l+2))*2*pi/per
      b(l+1)=0.
      l=l+1
      nz=0
      del=cmx*1.e-10
      do 81 i=1,l
      a1=a(i)
      b1=b(i)
      if(sqrt(a1*a1+b1*b1).lt.del)go to 81
      nz=nz+1
      a(nz)=a1
      b(nz)=b1
      frq(nz)=i/per
      if(abs(a1).lt.del)a(nz)=0.
      if(abs(b1).lt.del)b(nz)=0.
81    continue
      print 82
82    format(' nonzero fourier series coefficients')
      print 84,a0
84    format(' constant= ',g15.8)
      print 90
90    format(' freq. (cy/sec)   cos coeff.     sin coeff.')
      if(nz.eq.0)go to 110
      do 100 j=1,nz
      fr=frq(j)
      print 95,fr,a(j),b(j)
95    format(3(2x,g15.8))
100   continue
110   continue
      if(nm.eq.3)go to 1
      npts=500
      do 135 i=1,npts
      t=(i-1)*per/(npts-1)
      y=f(t)
      if(i.eq.1)ymn=y
      if(i.eq.1)ymx=y
      if(y.lt.ymn)ymn=y
      if(y.gt.ymx)ymx=y
135   continue
      xmn=0.
      xmx=per
      xmarj=(xmx-xmn)*.2
      xmn=xmn-xmarj
      xmx=xmx+xmarj
      ymarj=(ymx-ymn)*.2
      ymn=ymn-ymarj
      ymx=ymx+ymarj
      call xvwpor(gfile,-one,one,-one,one)
      call xwindo(gfile,xmn,xmx,ymn,ymx)
      call xlincl(gfile,1)
      do 140 i=1,npts
       t=(i-1)*per/(npts-1)
       y=f(t)
       if(i .eq. 1)then
        call xmove(gfile,t,y)
```

```
       else
        call xdraw(gfile,t,y)
       endif
140   continue
      call xlincl(gfile,2)
c     dw=2*pi/per
      do 160 i=1,npts
       t=(i-1)*per/(npts-1)
       s=a0
       if(nz .gt. 0)then
        do 150 j=1,nz
         w=2.*pi*frq(j)
         s=s+a(j)*cos(w*t)+b(j)*sin(w*t)
150     continue
       endif
       if(i.eq.1)then
        call xmove(gfile,t,s)
       else
        call xdraw(gfile,t,s)
       endif
160   continue
      call xlincl(gfile,3)
      do 170 i=1,n
       t=(per/n)*(i-1)
       ff=f(t)
       ns=1
       size=.02
       call xdsymbo(gfile,ns,t,ff,size)
170   continue
      go to 1
      end
c+ fft2c fast fourier transform
      subroutine  fft2c (a,m,iwk)
c   imsl routine name   - fft2c
c
c-----------------------------------------------------------------------
c
c   computer            - cdc/single
c
c   latest revision     - january 1, 1978
c
c   purpose             - compute the fast fourier transform of a
c                           complex valued sequence of length equal to
c                           a power two
c
c   usage               - call fft2c (a,m,iwk)
c
c   arguments    a      - complex vector of length n, where n=2**m.
c                           on input a contains the complex valued
c                           sequence to be transformed.
c                           on output a is replaced by the
c                           fourier transform.
c                m      - input exponent to which 2 is raised to
c                           produce the number of data points, n
c                           (i.e. n = 2**m).
```

```
c                iwk    - work vector of length m+1.
c
c   precision/hardware  - single and double/h32
c                       - single/h36,h48,h60
c
c   reqd. imsl routines - none required
c
c   notation            - information on special notation and
c                           conventions is available in the manual
c                           introduction or through imsl routine uhelp
c
c   remarks  1.  fft2c computes the fourier transform, x, according
c                to the following formula;
c
c                  x(k+1) = sum from j = 0 to n-1 of
c                           a(j+1)*cexp((0.0,(2.0*pi*j*k)/n))
c                  for k=0,1,...,n-1 and pi=3.1415...
c
c                note that x overwrites a on output.
c            2.  fft2c can be used to compute
c
c                  x(k+1) = (1/n)*sum from j = 0 to n-1 of
c                           a(j+1)*cexp((0.0,(-2.0*pi*j*k)/n))
c                  for k=0,1,...,n-1 and pi=3.1415...
c
c                by performing the following steps;
c
c                     do 10 i=1,n
c                        a(i) = conjg(a(i))
c                  10 continue
c                     call fft2c (a,m,iwk)
c                     do 20 i=1,n
c                        a(i) = conjg(a(i))/n
c                  20 continue
c
c   copyright           - 1978 by imsl, inc. all rights reserved.
c
c   warranty            - imsl warrants only that imsl testing has been
c                           applied to this code. no other warranty,
c                           expressed or implied, is applicable.
c
c-----------------------------------------------------------------------
c
      implicit real*8 (a-h,o-z)
c                                  specifications for arguments
      integer            m,iwk(*)
      complex*16            a(*)
c                                  specifications for local variables
      integer            i,isp,j,jj,jsp,k,k0,k1,k2,k3,kb,kn,mk,mm,mp,n,
     1                   n4,n8,n2,lm,nn,jk
      real               rad,c1,c2,c3,s1,s2,s3,ck,sk,sq,a0,a1,a2,a3,
     1                   b0,b1,b2,b3,twopi,temp,
     2                   zero,one,z0(2),z1(2),z2(2),z3(2)
      complex            za0,za1,za2,za3,ak2
      equivalence        (za0,z0(1)),(za1,z1(1)),(za2,z2(1)),
```

```
     1                  (za3,z3(1)),(a0,z0(1)),(b0,z0(2)),(a1,z1(1)),
     2                  (b1,z1(2)),(a2,z2(1)),(b2,z2(2)),(a3,z3(1)),
     3                  (b3,z3(2))
      data              sq/.70710678118655d0/,
     1                  sk/.38268343236509d0/,
     2                  ck/.92387953251129d0/,
     3                  twopi/6.2831853071796d0/
      data              zero/0.0d0/,one/1.0d0/
c                                 sq=sqrt2/2,sk=sin(pi/8),ck=cos(pi/8)
c                                 twopi=2*pi
c                                 first executable statement
      mp = m+1
      n = 2**m
      iwk(1) = 1
      mm = (m/2)*2
      kn = n+1
c                                 initialize work vector
      do 5  i=2,mp
         iwk(i) = iwk(i-1)+iwk(i-1)
    5 continue
      rad = twopi/n
      mk = m - 4
      kb = 1
      if (mm .eq. m) go to 15
      k2 = kn
      k0 = iwk(mm+1) + kb
   10 k2 = k2 - 1
      k0 = k0 - 1
      ak2 = a(k2)
      a(k2) = a(k0) - ak2
      a(k0) = a(k0) + ak2
      if (k0 .gt. kb) go to 10
   15 c1 = one
      s1 = zero
      jj = 0
      k = mm - 1
      j = 4
      if (k .ge. 1) go to 30
      go to 70
   20 if (iwk(j) .gt. jj) go to 25
      jj = jj - iwk(j)
      j = j-1
      if (iwk(j) .gt. jj) go to 25
      jj = jj - iwk(j)
      j = j - 1
      k = k + 2
      go to 20
   25 jj = iwk(j) + jj
      j = 4
   30 isp = iwk(k)
      if (jj .eq. 0) go to 40
c                                 reset trigonometric parameters
      c2 = jj * isp * rad
      c1 = cos(c2)
      s1 = sin(c2)
```

```
   35 c2 = c1 * c1 - s1 * s1
      s2 = c1 * (s1 + s1)
      c3 = c2 * c1 - s2 * s1
      s3 = c2 * s1 + s2 * c1
   40 jsp = isp + kb
c                                 determine fourier coefficients
c                                   in groups of 4
      do 50 i=1,isp
         k0 = jsp - i
         k1 = k0 + isp
         k2 = k1 + isp
         k3 = k2 + isp
         za0 = a(k0)
         za1 = a(k1)
         za2 = a(k2)
         za3 = a(k3)
         if (s1 .eq. zero) go to 45
         temp = a1
         a1 = a1 * c1 - b1 * s1
         b1 = temp * s1 + b1 * c1
         temp = a2
         a2 = a2 * c2 - b2 * s2
         b2 = temp * s2 + b2 * c2
         temp = a3
         a3 = a3 * c3 - b3 * s3
         b3 = temp * s3 + b3 * c3
   45    temp = a0 + a2
         a2 = a0 - a2
         a0 = temp
         temp = a1 + a3
         a3 = a1 - a3
         a1 = temp
         temp = b0 + b2
         b2 = b0 - b2
         b0 = temp
         temp = b1 + b3
         b3 = b1 - b3
         b1 = temp
         a(k0) = cmplx(a0+a1,b0+b1)
         a(k1) = cmplx(a0-a1,b0-b1)
         a(k2) = cmplx(a2-b3,b2+a3)
         a(k3) = cmplx(a2+b3,b2-a3)
   50 continue
      if (k .le. 1) go to 55
      k = k - 2
      go to 30
   55 kb = k3 + isp
c                                 check for completion of final
c                                   iteration
      if (kn .le. kb) go to 70
      if (j .ne. 1) go to 60
      k = 3
      j = mk
      go to 20
   60 j = j - 1
```

```
      c2 = c1
      if (j .ne. 2) go to 65
      c1 = c1 * ck + s1 * sk
      s1 = s1 * ck - c2 * sk
      go to 35
   65 c1 = (c1 - s1) * sq
      s1 = (c2 + s1) * sq
      go to 35
   70 continue
c                                  permute the complex vector in
c                                    reverse binary order to normal
c                                    order
      if(m .le. 1) go to 9005
      mp = m+1
      jj = 1
c                                  initialize work vector
      iwk(1) = 1
      do 75  i = 2,mp
         iwk(i) = iwk(i-1) * 2
   75 continue
      n4 = iwk(mp-2)
      if (m .gt. 2) n8 = iwk(mp-3)
      n2 = iwk(mp-1)
      lm = n2
      nn = iwk(mp)+1
      mp = mp-4
c                                  determine indices and switch a
      j = 2
   80 jk = jj + n2
      ak2 = a(j)
      a(j) = a(jk)
      a(jk) = ak2
      j = j+1
      if (jj .gt. n4) go to 85
      jj = jj + n4
      go to 105
   85 jj = jj - n4
      if (jj .gt. n8) go to 90
      jj = jj + n8
      go to 105
   90 jj = jj - n8
      k = mp
   95 if (iwk(k) .ge. jj) go to 100
      jj = jj - iwk(k)
      k = k - 1
      go to 95
  100 jj = iwk(k) + jj
  105 if (jj .le. j) go to 110
      k = nn - j
      jk = nn - jj
      ak2 = a(j)
      a(j) = a(jj)
      a(jj) = ak2
      ak2 = a(k)
      a(k) = a(jk)
```

```
      a(jk) = ak2
  110 j = j + 1
c                                 cycle repeated until limiting number
c                                   of changes is achieved
      if (j .le. lm) go to 80
c
 9005 return
      end
c+ f signal function
c      function f(t)
c      implicit real*8 (a-h,o-z)
c      data pi/3.14159265358979d0/
c      zero=0.
c      one=1.
c     k=cycles per second
c      ak=1.9098609
c      j=4
c      c=0.
c      f=sin(ak*2.*pi*t)
c      f=f+cos(j*2.*pi*t)+c
c      return
c      end
c
c+ f signal function
      function f(t)
      implicit real*8 (a-h,o-z)
      data pi/3.14159265358979d0/
      zero=0.
      one=1.
      if(t .lt. zero)t=-t
      m=t
      tt=t-m
      f=1.
      if(tt .gt. one/2.)f=-1.
      return
      end
c+ readr read a row of floating point numbers
      subroutine readr(nf, a, nr)
      implicit real*8(a-h,o-z)
c   numbers are separated by spaces
c   examples of valid numbers are:
c   12.13 34 45e4 4.78e-6 4e2,5.6D-23,10000.d015
c  nf=file number, 0 for standard input file
c   a=array of returned numbers
c  nr=number of values in returned array,
c     or 0 for empty or blank line,
c     or -1 for end of file on unit nf.
c requires functions val and length
      dimension a(*)
      character*200  b
      character*200  c
      character*1 d
        c=' '
      if(nf.eq.0)then
         read(*,'(a)',end=99)b
```

```
      else
         read(nf,'(a)',end=99)b
      endif
      nr=0
      l=lenstr(b)
      if(l.ge.200)then
        write(*,*)' error in readr subroutine '
        write(*,*)' record is too long '
      endif
      do 1 i=1,l
       d=b(i:i)
       if (d.ne.' ') then
        k=lenstr(c)
        if (k.gt.0)then
           c=c(1:k)//d
        else
           c=d
        endif
       endif
       if( (d.eq.' ').or.(i.eq.l)) then
        if (c.ne.' ') then
          nr=nr+1
           call valsub(c,a(nr),ier)
          c=' '
        endif
       endif
1     continue
      return
99    nr=-1
      return
      end
c
c+ lenstr   nonblank length of string
      function lenstr(s)
c  length of the substring of s obtained by deleting all
c  trailing blanks from s.  thus the length of a string
c  containing only blanks will be 0.
      character   s*(*)
      lenstr=0
      n=len(s)
      do 10 i=n,1,-1
      if(s(i:i) .ne. ' ')then
         lenstr=i
         return
      endif
10    continue
      return
      end
c+ valsub converts string to floating point number (r*8)
      subroutine valsub(s,v,ier)
      implicit real*8(a-h,o-z)
c     examples of valid strings are: 12.13 34 45e4 4.78e-6 4E2
c     the string is checked for valid characters,
c     but the string can still be invalid.
c      s-string
```

```
c      v-returned value
c      ier- 0 normal
c           1 if invalid character found, v returned 0
c
      logical p
      character s*(*),c*50,t*50,ch*15
      character z*1
      data ch/'1234567890+-.eE'/
      v=0.
      ier=1
      l=lenstr(s)
      if(l.eq.0)return
      p=.true.
      do 10 i=1,l
      z=s(i:i)
      if((z.eq.'D').or.(z.eq.'d'))then
          s(i:i)='e'
      endif
      p=p.and.(index(ch,s(i:i)).ne.0)
 10   continue
      if(.not.p)return
      n=index(s,'.')
      if(n.eq.0)then
          n=index(s,'e')
          if(n.eq.0)n=index(s,'E')
          if(n.eq.0)n=index(s,'d')
          if(n.eq.0)n=index(s,'D')
          if(n.eq.0)then
             s=s(1:l)//'.'
          else
            t=s(n:l)
            s=s(1:(n-1))//'.'//t
          endif
          l=l+1
      endif
      write(c,'(a30)')s(1:l)
      read(c,'(g30.23)')v
      ier=0
      return
      end
c+ xszprm write text size parameter (external plot)
      subroutine xszprm(nfile,h)
      implicit real*8(a-h,o-z)
      character s*25,t*26
      call str(h,s)
      l=lenstr(s)
      t(1:1)='z'
      t(2:(l+1))=s(1:l)
      write(nfile,'(a)')t(1:(l+1))
      return
      end
c+ xangpr write text angle parameter (external plot)
      subroutine xangpr(nfile,a)
      implicit real*8(a-h,o-z)
      character s*25,t*26
```

```
      call str(a,s)
      l=lenstr(s)
      t(1:1)='g'
      t(2:(1+l))=s(1:l)
      write(nfile,'(a)')t(1:(l+1))
      return
      end
c+ xdsymbo  symbol to external plot file
      subroutine xdsymbo(nfile,ns,x,y,size)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='s'
      n=2
      call str(x,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(y,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      ans=ns
      call str(ans,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(size,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c+ xlincl line color parameter to external plot file
      subroutine xlincl(nfile,nc)
      implicit real*8(a-h,o-z)
      write(nfile,10)nc
10    format('c',i1)
      return
      end
c+ xdarc  arc into external plot file
      subroutine xdarc(nfile,xc,yc,r,a1,a2,npts)
      implicit real*8(a-h,o-z)
c nfile-unit number for output file
c xc,yc-arc center
c     r-arc radius
c a1,a2-start and end angles
c  npts-number of points in arc
      character s*25,t*80
      t='a'
      n=2
      call str(xc,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(yc,s)
```

```
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(r,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(a1,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(a2,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      an=npts
      call str(an,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c+ xwindo  window parameters to external plot file
      subroutine xwindo(nfile,xmn,xmx,ymn,ymx)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='w'
      n=2
      call str(xmn,s)
      l=lenstr(s)
      t(n:(n+l-1))=s
      n=n+l+1
      call str(xmx,s)
      l=lenstr(s)
      t(n:(n+l-1))=s
      n=n+l+1
      call str(ymn,s)
      l=lenstr(s)
      t(n:(n+l-1))=s
      n=n+l+1
      call str(ymx,s)
      l=lenstr(s)
      t(n:(n+l-1))=s
      write(nfile,'(a)')t(1:(n+l-1))
      call wprm(1,xmn,xmx,ymn,ymx)
      return
      end
c+ xvwpor  viewport parameters to external plot file
      subroutine xvwpor(nfile,xmn,xmx,ymn,ymx)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='v'
      n=2
      call str(xmn,s)
      l=lenstr(s)
```

```
      t(n:80)=s
      n=n+l+1
      call str(xmx,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(ymn,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(ymx,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      call vprm(1,xmn,xmx,ymn,ymx)
      return
      end
c+ xdraw  draw parameters to external plot file
      subroutine xdraw(nfile,x,y)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='d'
      n=2
      call str(x,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(y,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c
c+ xmove  move parameters to external plot file
      subroutine xmove(nfile,x,y)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='m'
      n=2
      call str(x,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(y,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c+ str    floating point number to string
      subroutine str(x,s)
      implicit real*8(a-h,o-z)
      character s*25,c*25,b*25,e*25
       zero=0.
```

```
      if(x.eq.zero)then
         s='0'
         return
      endif
      write(c,'(g11.4)')x
      read(c,'(a25)')b
      l=lenstr(b)
      do 10 i=1,l
      n1=i
      if(b(i:i).ne.' ')go to 20
 10   continue
 20   continue
       if(b(n1:n1).eq.'0')n1=n1+1
       b=b(n1:l)
       l=l+1-n1
       k=index(b,'E')
       if(k.gt.0)e=b(k:l)
       if(k.gt.0)then
         s=b(1:(k-1))
         k1=index(b,'E+0')
         if(k1.gt.0)then
          e='E'//b((k1+3):l)
         else
          k1=index(b,'E+')
          if(k1.gt.0)e='E'//b((k1+2):l)
         endif
         k1=index(b,'E-0')
         if(k1.gt.0)e='E-'//b((k1+3):l)
         l=k-1
       else
          s=b
       endif
       j=index(s,'.')
       n2=l
      if(j.ne.0)then
              do 30 i=1,l
               n2=l+1-i
               if(s(n2:n2).ne.'0')go to 40
30             continue
       endif
40             continue
        s=s(1:n2)
        if(s(n2:n2).eq.'.')then
           s=s(1:(n2-1))
           n2=n2-1
         endif
        if(k.gt.0)s=s(1:n2)//e
      return
      end
```

10 Finite Fourier Analysis

On a set of uniformly spaced points a function space has a finite Fourier basis. See Emery **Finite Fourier Analysis** .

11 Fourier Least Squares

For a general introduction to this topic see: Emery **Least Squares Approximation** (lsq.tex). Suppose we wish to fit a function of the form

$$\sin(\omega t + \phi).$$

We may vary ω over a frequency interval and compute the best fit of the function

$$A\cos(\omega t) + B\sin(\omega t).$$

Then

$$\begin{aligned} &A\cos(\omega t) + B\sin(\omega t) \\ &= \sqrt{A^2+B^2}\left[\frac{A}{\sqrt{A^2+B^2}}\cos(\omega t) + \frac{B}{\sqrt{A^2+B^2}}\sin(\omega t)\right] \\ &= C[\sin\phi\cos\omega t + \cos\phi\sin\omega t] \\ &= C\sin(\omega t + \phi), \end{aligned}$$

where

$$\tan(\phi) = \frac{B}{A}.$$

12 Computing The Period of a Sampled Signal

Suppose we have a periodic sampled signal $\{x_i\}$ consisting of n values. Suppose it has a period $T < n/3$. Suppose the signal has maximum values or peaks at points k, $k+T$, $k+2T$ and so on. Then we can find the period T by locating these peaks and taking T to be the average distance between these peaks. First we shall find the maximum of the signal, which we write as x_m. We let

$$x_1 = .9x_m$$

and

$$x_2 = .95x_m.$$

As we begin searching for a new peak at point j we let

$$z = x_j$$

Then as we examine subsequent points, we store in z the largest value found so far. We store the index of the largest point in k. When a point x_i is found that has fallen below x_1 (.9 times the largest global value), and when we found a value greater than z_2 (.95 times the largest global value), which we have stored in z, then we assume that we are descending from a peak at $z = x_k$. We store this peak point, and reset z and k. Then we look for the next peak. This method is realized in the following $C++$ function:

```
//c+ peaks finds the peaks of a signal
int peaks(
 dvector& x,
 int& n,
 ivector& k,
 int& np){
 //declaration int peaks(dvector&,int&,ivector&,int&);
 int i;
 int kmxpk;
 double x1;
 double x2;
 double xmx;
 double xmxpk;
// External Functions:
 double maxd(double,double);
 //c Input:
 //c t    time array
 //c x    signal value array
 //c n    number of elements in the signal sample
 //c Output:
 //c k   array of peaks, indices where peaks occur
 //c np  number of peaks
 //c     find peaks
```

```
 //c     get absolute maximum of signal
 xmx=x.g(1);
 for(i=1;i<=n ;i++){
  xmx=maxd(xmx,x.g(i));
 }
 x1=.9*xmx;
 x2=.95*xmx;
 np=0;
 kmxpk=1;
 xmxpk=x.g(1);
 for(i=1;i<=n ;i++){
  if((x.g(i) < x1) && (xmxpk  >  x2)){
   np = np + 1;
   k.p(np, kmxpk );
   kmxpk=i;
   xmxpk=x.g(i);
  }
  if(x.g(i)  >  xmxpk){
   xmxpk=x.g(i);
   kmxpk=i;
  }
 }
 return(0);
}
```

13 Finding the Phase and Magnitude of the Fundamental Component of a Sampled Signal

Suppose we have found the period T of a signal as in the previous section. Then the first component of the signal may be written as

$$c_1 \sin(\omega t + \phi),$$

where

$$\omega = \frac{2\pi}{T},$$

and ϕ is the phase angle. The fourier series for the signal is

$$f(t) = a_0/2 + \sum_{n=1}^{\infty} (a_i \cos(n\omega t) + b_i \sin(n\omega t)).$$

Thus (we may assume that c_1 is positive) we have

$$c_1 \sin(\omega t + \phi) = a_1 \cos(\omega t) + b_1 \sin(\omega t)$$

$$= \sqrt{a_1^2 + b_1^2} \left[\frac{a_1}{\sqrt{a_1^2 + b_1^2}} \cos(\omega t) + \frac{b_1}{\sqrt{a_1^2 + b_1^2}} \sin(\omega t) \right]$$

$$= \sqrt{a_1^2 + b_1^2} [\sin(\phi) \cos(\omega t) + \cos(\phi) \sin(\omega t)].$$

Then the phase ϕ is the argument (or angle of the line from the origin to the point) of $(\cos(\phi), \sin(\phi)) = (b_1, a_1)$. That is, in terms of a well known computer function

$$\phi = \mathrm{ATAN2}(a_1, b_1).$$

The magnitude of the first Fourier component is

$$c_1 = \sqrt{a_1^2 + b_1^2}.$$

The fourier coefficients are

$$a_1 = \frac{2}{T} \int_0^T f(t) \cos(\omega t) dt$$

and

$$b_1 = \frac{2}{T} \int_0^T f(t) \sin(\omega t) dt.$$

They may be computed numerically using the trapezoid rule. This is done for the b_1 coefficient in the following C++ function:

```
//c+ sincf sine coefficient of fundamental component of signal
int sincf(
 dvector& f,
 int& n,
 double& p,
 double& b1){
```

```
 //declaration int sincf(dvector&,int&,double&,double&);
 double dt;
 int i;
 double omega;
 double pi;
 double t;
 double v;
 // External Functions:
 // double sin();
 //c Input:
 //c f    sample signal values evenly spaced on interval [0,p]
 //c n    number of sample values
 //c p    period of the signal
 //c Output:
 //c b1   first sine coefficient
 //c Method:   trapazoid rule integration of the integral
 //c     (2/p) \int{0 to p} f(t) \sin(\omega t) dt
 pi=3.14159265358979;
 omega=2.*pi/p;
 dt=p/(n-1);
 b1=0.;
 for(i=1;i<=n ;i++){
  t=(i-1)*dt;
  v=f.g(i)*sin(omega*t);
  if((i  ==  1)  ||  (i  ==  n)){
   v=v/2.;
  }
  b1=b1+v;
 }
 b1=2.*dt*b1/p;
 return(0);
}
```

There is an almost identical function for the cos coefficient.

If we have two signals, we can compute the phase difference between them as the difference of their individual phases.

The technique described here can be used to compute impedance of a load from voltage and current signals.

The FFT could be used for this calculation.

14 Impedance Program

Output of the program **imp.c**:

```
% imp
n= 1024
1 index 146 value            470
2 index 506 value            479
3 index 862 value            414
Period =            358
nn= 358
Current a1 =        -97.4514
Current b1 =       48.781341
current phase angle =      -63.408784 degrees
current magnitude =       108.97887
Voltage a1 =      -214.00182
Voltage b1 =      -382.58334
Voltage phase angle =       -150.7791 degrees
Voltage magnitude =       438.36833
Impedance phase angle =      -87.370312 degrees
Impedance magnitude =       4.0225075
```

Suppose the sampling rate is $20mH$, that is 20×10^6 points per second. Then the distance between sample points is

$$\Delta t = \frac{1}{20 \times 10^6} = 50ns.$$

From the program output, the fundamental period in seconds is

$$T = (50)(358) = 17900ns$$

So the fundamental frequency is

$$f = 1/T = 55.8 \times 10^3 H.$$

To compute the actual impedance magnitude in ohms, we need the the current and voltage scale factors. Note that here zero voltage has value 512. The voltage and current signals have been discretized so that they take values between 0 and 1023.

Listing of the program **imp.c**:

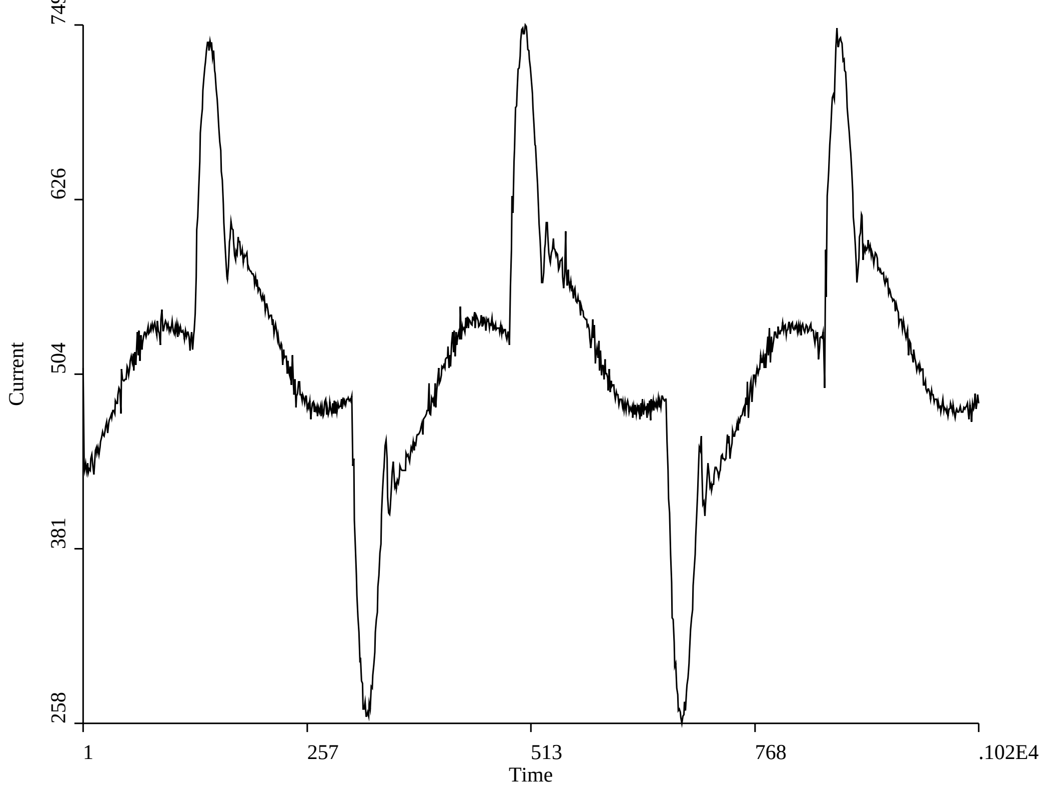

Figure 2: Fundamental Components of Current and Voltage Signals.

```
/* imp.c Impedance from signals from LabWindows data aquisition system*/
/* jim emery 3/21/97 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
/* #include "matvec.h" */
int da[1024][8];
main(){
 FILE *nf1;
 int n;
 double a[10];
 /*t readfv(FILE*,double*,double*,int*);*/

 nf1=fopen("p.dig","r");
 n=-1;
 while(readr(nf1,a)  >  0) {
  n = n + 1;
  da[n][0]=a[0];
  da[n][1]=a[1];
  da[n][2]=a[2];
  da[n][3]=a[3];
  da[n][4]=a[4];
  da[n][5]=a[5];
  da[n][6]=a[6];
  da[n][7]=a[7];
 }
 impedance();
}
 int impedance(){
 /*imp.ftn impedance from signals*/
 FILE *out;
 double a[2000];
 double a1;
 double b1;
 double f[2000];
 int i;
 int j;
 int k[100];
 int n;
 int nn;
 int np;
 double omega;
 double phi;
 double pi;
 double s;
 double t[2000];
 double tp;
 double tt;
 double v[2000];
 double c[2000];
 double cm,vm,cphi,vphi;
 double x,y;
 double maxd(double,double);
 int coscf(double*,int,double,double*);
```

```
int sincf(double*,int,double,double*);
int peaks(double*,int,int*,int*);
out=fopen("p.gi","w");
n=1024;
printf("n= %d\n",n);
for(i=0;i<n;i++){
 t[i]=i;
 c[i]=da[i][0];
 v[i]=da[i][5];
}

fprintf(out,"v-1 1 -1 1\n");
fprintf(out,"w0 1024 0 1024\n");
fprintf(out,"c3\n");
for(i=0;i<1024;i++){
 x=t[i];
 y=c[i];
 if(i == 0){
  fprintf(out,"m%15.8g %15.8g\n",x,y);
 }
 else{
  fprintf(out,"d%15.8g %15.8g\n",x,y);
 }
}
fprintf(out,"c4\n");
for(i=0;i<1024;i++){
 x=t[i];
 y=v[i];
 if(i == 0){
  fprintf(out,"m%15.8g %15.8g\n",x,y);
 }
 else{
  fprintf(out,"d%15.8g %15.8g\n",x,y);
 }
}

peaks(c,n,k,&np);
for(i=1;i<=np;i++){
 j=k[i-1];
 printf("%d index %d value %15.8g\n",i,j,v[j-1]);
}
if(np  >  1){
 s=0.;
 for(i=1;i<=np-1 ;i++){
  s = s + t[k[i]]-t[k[i-1]];
 }
 tp = s/(np-1);
  printf("Period = %15.8g\n",tp);
}
pi=3.14159265358979;
omega=2.*pi/tp;
nn=tp;
printf("nn= %d \n",nn);
coscf(c,nn,tp,&a1);
printf("Current a1 = %15.8g \n",a1);
```

```
 sincf(c,nn,tp,&b1);
 printf("Current b1 = %15.8g \n",b1);
 cphi = atan2(a1,b1)*180./pi;
 printf("current phase angle = %15.8g degrees\n",cphi);
 cm=sqrt(a1*a1+b1*b1);
 printf("current magnitude = %15.8g \n",cm);
 fprintf(out,"c6\n");
 for(i=0;i<1024;i++){
  x=t[i];
  y=512. + a1*cos(omega*x) + b1*sin(omega*x) ;
  if(i == 0){
   fprintf(out,"m%15.8g %15.8g\n",x,y);
  }
  else{
   fprintf(out,"d%15.8g %15.8g\n",x,y);
  }
 }

 coscf(v,nn,tp,&a1);
 printf("Voltage a1 = %15.8g \n",a1);
 sincf(v,nn,tp,&b1);
 printf("Voltage b1 = %15.8g \n",b1);
 vphi = atan2(a1,b1)*180./pi;
 printf("Voltage phase angle = %15.8g degrees\n",vphi);
 vm=sqrt(a1*a1+b1*b1);
 printf("Voltage magnitude = %15.8g \n",vm);
 fprintf(out,"c7\n");
 for(i=0;i<1024;i++){
  x=t[i];
  y=512. + a1*cos(omega*x) + b1*sin(omega*x) ;
  if(i == 0){
   fprintf(out,"m%15.8g %15.8g\n",x,y);
  }
  else{
   fprintf(out,"d%15.8g %15.8g\n",x,y);
  }
 }

 printf("Impedance phase angle = %15.8g degrees\n",vphi-cphi);
 vm=sqrt(a1*a1+b1*b1);
 printf("Impedance magnitude = %15.8g \n",vm/cm);

 return 0;
}
/*c+ readfv  read an array of function pairs*/
int readfv(
 FILE* nf,
 double* x,
 double* y,
 int* n){
 /*//declaration int readfv(FILE*,dvector&,dvector&,int&);*/
 double a[2];
 int ntmp;
 /* External Functions: */
 int readr(FILE*,double*);
```

```
 /*
 //c input:
 //c nf      file number
 //c Output:
 //c x   array of x values
 //c y   array of y values
 //c n   number of values
 //c
 //c Note: uses ireadr, which reads numbers separated
 //c by spaces, and returns -1 at end of file.
 */
 ntmp=0;
 while(readr(nf,a)  >  0) {
  ntmp = ntmp + 1;
  x[ntmp-1]=a[0];
  y[ntmp-1]=a[1];
 }
 *n=ntmp;
 return(0);
}
/*c+ readr read row of numbers*/
int readr(FILE *fn,double *a){
 /*
 //declaration int readr(FILE*,double*);
 // input:
 // fn file pointer
 // output:
 // a-array of numbers read
 // returned value is:
 //  -1, end of file
 //   0, empty line
 //   n, n is number of values read
 //   Remarks:
 //   separate numbers by blanks.
 //   to read from keyboard use: n=readr(stdin,a).
 //   modified for c++, 10/29/96
 */
 extern double atof(const char *s);
 char b[200],c[25],d[2];
 int i,l,nr;
 strcpy(c,"");
 if(fgets(b,200,fn)==NULL){
  nr=-1;
 }
 else{
  nr=0;
  /* fgets puts newline character into string, so subtract 1*/
  l=strlen(b)-1;
  d[0]=' ';
  for(i=0;i<l;i++){
   d[0]=b[i];
   if(d[0] != ' ')strncat(c,d,1);
   if ((d[0]==' ') || (i==l-1)){
    if(strlen(c) != 0){
     nr=nr+1;
```

```
     a[nr-1]=atof(c);
     strcpy(c,"");
    }
   }
  }
 }
 return(nr);
}
/*c+ peaks finds the peaks of a signal*/
int peaks(
 double* x,
 int n,
 int* k,
 int* np){
 /*declaration int peaks(dvector&,int&,ivector&,int&);*/
 int i;
 int kmxpk;
 double x1;
 double x2;
 double xmx;
 double xmxpk;
 int nptmp;
 /*// External Functions:*/
 double maxd(double,double);
 /*
 //c Input:
 //c t    time array
 //c x    signal value array
 //c n    number of elements in the signal sample
 //c Output:
 //c k   array of peaks, indices where peaks occur
 //c np  number of peaks
 //c     find peaks
 //c     get absolute maximum of signal
 */
 xmx=x[0];
 for(i=1;i<=n ;i++){
  xmx=maxd(xmx,x[i-1]);
 }
 x1=.9*xmx;
 x2=.95*xmx;
 nptmp=0;
 kmxpk=1;
 xmxpk=x[0];
 for(i=1;i<=n ;i++){
  if((x[i-1] < x1) && (xmxpk  >  x2)){
   nptmp = nptmp + 1;
   k[nptmp-1] =  kmxpk;
   kmxpk=i;
   xmxpk=x[i-1];
  }
  if(x[i-1]  >  xmxpk){
   xmxpk=x[i-1];
   kmxpk=i;
  }
```

```
 }
 *np=nptmp;
 return(0);
}
/*c+ maxd maximum of two numbers (doubles)*/
double maxd(double x,double y){
 /* //declaration double maxd(double,double);*/
 if( x < y ){
  return(y);
 }
 else{
  return(x);
 }
}
/*c+ coscf cosine coefficient*/
int coscf(
 double* f,
 int n,
 double p,
 double* a1){
 /* //declaration int coscf(dvector&,int&,double&,double&);*/
 double dt;
 int i;
 double omega;
 double pi;
 double t;
 double v;
 double a1p;
 /*
 // External Functions:
 // double cos();
 //c Input:
 //c f    function values evenly spaced on interval [0,p]
 //c n    number of function values
 //c p    period of the function
 //c Output:
 //c a1   first cosine coefficient
 //c     trapazoid rule integration
 //c     (1/2p) \int f(t)*cos(\omega*t) dt
 */
 pi=3.14159265358979;
 omega=2.*pi/p;
 dt=p/(n-1);
 a1p=0.;
 for(i=1;i<=n ;i++){
  t=(i-1)*dt;
  v=f[i-1]*cos(omega*t);
  if((i  ==  1)  ||  (i  ==  n)){
   v=v/2.;
  }
  a1p=a1p+v;
 }
 *a1=2.*dt*a1p/p;
 return(0);
}
```

```
/*c+ sincf sine coefficient*/
int sincf(
 double* f,
 int n,
 double p,
 double* b1){
 /* //declaration int sincf(dvector&,int&,double&,double&);*/
 double dt;
 int i;
 double omega;
 double pi;
 double t;
 double v;
 double b1p;
 /*
 // External Functions:
 // double sin();
 //c Input:
 //c f    function values evenly spaced on interval [0,p]
 //c n    number of function values
 //c p    period of the function
 //c Output:
 //c b1   first sine coefficient
 //c     trapazoid rule integration
 //c     (1/2p) \int f(t)*cos(\omega*t) dt
 */
 pi=3.14159265358979;
 omega=2.*pi/p;
 dt=p/(n-1);
 b1p=0.;
 for(i=1;i<=n ;i++){
  t=(i-1)*dt;
  v=f[i-1]*sin(omega*t);
  if((i  ==  1)  ||  (i  ==  n)){
   v=v/2.;
  }
  b1p=b1p+v;
 }
 *b1=2.*dt*b1p/p;
 return(0);
}
```

15 Applications

Fourier Analysis has been applied in a huge number of areas in science. Some of these areas are: Optics and Diffraction Theory, Signal Processing, X-ray Crystallography and Molecular Structure determination, Approximation Theory, Partial Differential Equations, Tomography, Several Areas in Electrical Engineering, Quantum mechanics, The Theory of Heat Conduction (where it originated), and so on.

16 The Fourier Method of Tomography

A parallel beam of radiation passes through a body and is partially scattered. The intensity of the unscattered radiation is measured on the opposite side of the body. Let a quantity of radiation dq be scattered in distance $d\eta$. This will be proportional to distance and to the amount of original radiation q. Thus the change in radiation intensity is

$$dq = -\lambda q d\eta$$

where λ is a scattering constant. Then

$$\frac{dq}{d\eta} = -\lambda q.$$

Hence

$$q = q_0 e^{-\lambda\eta}.$$

Now suppose the attenuation function varies throughout the body. Let $\lambda = f(x, y, z)$. Then

$$q = q_0 e^{-\int f d\eta}.$$

Then the negative logarithm of the intensity ratios is the integral of f in the direction of the radiation beam:

$$-\ln\frac{q}{q_0} = \int f d\eta.$$

The Projection Theorem.

Let R be an orthogonal transformation. Let

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = R \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Let $f(x_1, x_2, x_3)$ be the attenuation function. The Fourier transform of f is

$$Ff(k_1, k_2, k_3) = \frac{1}{(2\pi)^{(3/2)}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x_1, x_2, x_3) e^{ix\cdot k} dx_1 dx_2 dx_3$$

We shall make a change of variable

$$Ff(k_1, k_2, k_3) = \frac{1}{(2\pi)^{(3/2)}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(R^T x') e^{iR^T x' \cdot k} |J| dx_1' dx_2' dx_3',$$

where $|J|$ is the determinant of the Jacobian of the transformation. Here, because R is orthogonal, $J = 1$. Let $k' = R^T k$. Define

$$g(x') \;\; = \;\; f(R^T x').$$

Then the transform becomes

$$Ff(k) \;\; = \;\; \frac{1}{(2\pi)^{(3/2)}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x') e^{ix' \cdot k'} dx_1' dx_2' dx_3'.$$

The measured intensity at the plane perpendicular to the x_3' axis is

$$p(x_1', x_2') = \int_{-\infty}^{\infty} g(x_1', x_2', x_3') dx_3'$$

Then at a k vector, with a rotation R_k, where $k_3' = 0$, we have

$$Ff(k) \;\; = \;\; \frac{1}{(2\pi)^{(3/2)}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x_1', x_2') e^{i(k_1' x_1' + k_2' x_2')} dx_1' dx_2' = F_2 p(k_1', k_2').$$

This is the two dimensional Fourier transform of the projected intensity. Now given any point k we can find a plane passing through it and we can find a corresponding rotation R_k, so that $k_3' = 0$. Therefore we can compute the three dimensional transform by computing the two dimensional transform of the projected intensities. Then we can compute f by inverting the transform. However, in practice we will have only the transform at a set of finite points, so we must interpolate. The Fourier transform will be computed approximately as the finite Fourier transform, which uses only a finite set of points. However, interpolation is still needed because the proper points for computing the inverse transform will not in general correspond to the measured points. In practice a somewhat different method is used that does the interpolation before the projection intensity transform is calculated. This is the method of back projection.

17 Wavelets and Fourier Series

The theory of wavelets, concerns a set of functions with local support, that form, an orthogonal bases for a function space, and thus give a Fourier expansion of a function in terms of wavelets. In one sense wavelets are portions of infinite waves, or wave packets. Some classes of wavelets are derived as a scaled, and translated mother wavelet.

Wavelets have applications in signal processing, image processing, and data compression. See Meyer for a good introduction to wavelet theory, application, and history. See **Numerical Recipes in Fortran** , 2nd. edition, for wavelet subroutines.

A multiresolution analysis with wavelets, allows functions to be approximated to various scales.

18 Fourier Transforms of Tempered Distributions

See Yosida **Functional Analysis**.

19 A Generalization of Fourier Analysis: Abstract Harmonic Analysis

In the abstract formulation, the Fourier series, and the Fourier transform may be treated similarly. See the references.

20 Spaces of Complete Orthogonal Functions

Fourier analysis can be extended to families of orthogonal functions defined on L_2 spaces, see Horvath. See Widom for families of complete orthogonal functions generated by Sturm-Liouville boundary value problems.

21 Fourier Optics

Fourier transforms can be computed optically, see Goodman as well as other more modern books on Fourier optics.

22 Bibliography for Fourier Analysis

[1]Bachman, George, **Elements of Abstract Harmonic Analysis**, Academic Press, 1964.
[2]Barrow, Gordon M, **Physical Chemistry For The Life Sciences**, McGraw-Hill 1974, (2nd Ed. 1984), Johnson County, 541.3 B279, X-Ray Crystal Structure, Proteins, Fourier Analysis, Van Der Wals.
[3]Briggs William L, Henson Van Emden, **The DFT**, (Discrete Fourier Transform), SIAM, 1995.
[4]Chandrakumar Narayanan, **Modern Techniques In High Resolution FT-NMR**, 1987, Linda Hall, Fourier Optics, Nuclear Magnetic Resonance.
[5]Crowther R A, **Procedures For Three-dimensional Reconstruction Of Spherical Viruses By Fourier Synthesis From Electron Micrographs**, Proceedings Of The Royal Society Of London,, B 261, pp231-261.
[6]Elliott Douglas, Rao K Ramamohan, **Fast Transforms, Algorithms, Analysis, Applications**.
[7]Emery James D, **Finite Fourier Analysis**.
[8]Emery, James D, **Least Squares Approximation**, (lsq.tex).
[9]Emery, James D, **Tomography**, (tomog.tex).
[10]Fourier Joseph, **The Theory Of Heat**, Britanica Great Books.
[11]Fourier Joseph, **Theorie Analytique De La Chaleur**, 1822.
[12]Goldberg R R, **Fourier Transforms**, Cambridge, 1965.
[13]Goldberg R R, **Introduction To Real Analysis**.
[14]Goodman Joseph W, **Introduction To Fourier Optics**, Linda 1968 287 Pages.
[15]Horvath John, **Topological Vector Spaces**.
[16]Jardetzky O, Roberts G, **NMR In Molecular Biology**, Linda Hall,, QP519.9.N83j37, 1987, Outline Of History of Nuclear Magnetic Resonance, Fourier Transform,
[16]Katznelson Yitzhak, **An Introduction to Harmonic Analysis**, 1976, Dover.

[17]Korner T W, **Fourier Analysis**, Cambridge, 1988.
[18]Lighthill M J, **Fourier Analysis and Generalized Functions**, Cambridge, 1964.
[19]Loomis L H, **An Introduction to Abstract Harmonic Analysis**, Dover.
[19b]Meyer Yeves **Wavelets: Algorithms and Applications**, SIAM Press, Philadelphi, 1993. Dover.
[20]Papoulis Athanasios, **The Fourier Integral and Its Applications**, McGraw-Hill, 1962.
[20b]Press, Flannery, Teukolsky, Vetterling **Numerical Recipes in C**, Cambridge U. Press, 2nd edition, 1995.
[21]Ramachandran G N, Srinivasan R, **Fourier Methods In Crystallography**, Linda Hall, Qd945.r35, 1970.
[22]Rudin W, **Fourier Transforms On Groups**.
[23]Sederberg Thomas, **Scheduled Fourier Model Morphing**, Siggraph Proceedings, 1992.
[24]Stark Henry, **Applications Of Optical Fourier Transformations**, Ta1632 .a68 1982 Linda.
[25]Stout George H, Jenson Lyle H, **X-Ray Structure Determination**, Linda Hall, Qd945.s8, 1989, Fast Fourier Transform, Hauptman And Karl Technique.
[26]Strang Gilbert, **Wavelet Transform Versus Fourier Transform**, Bulletin Ams V28 N2 April 93, (also Books On Wavelets Reviewed).
[27]Thompson William J, **Computing For Scientists And Engineers**, C Programs, Complex Variables, Much On Fourier Analysis, 1992,, JOCO.
[28]Truesdell C, **The Tragicomical History of Thermodynamics**, Springer-Verlag, 1980, Critique of Fourier's Work on the Theory of Heat.
[29]Walther Adriaan, **The Ray And Wave Theory Of Lenses**, 1995, Fourier Optics, Linda.
[30]Weiner Norbert, **The Fourier Integral and Certain of Its Applications**, Dover.
[31]Wickerhauser Victor Mladen **Adapted Wavelet Analysis from Theory to Software**, A K Peters, 1994
[32]Widder David, **Laplace Transforms**.
[33]Widom Harold, **Lectures on Integral Equations**, D Van Nostrand, 1969, Eigenvectors of Sturm-Liouville Boundary Value Problems as Complete Orthogonal Functions on $L_2[a, b]$.

[34]Wilson Raymond, **Fourier Series And Optical Transforms Techniques In**, 1995 Linda Fourier Optics.
[35]Yosida Kasaku, **Functional Analysis**, Springer-Verlag, Fourier Transforms of Tempered Distributions.
[36]Zemanian A H, **Distribution Theory and Transform Analysis**.
[37]Zigmund Antoni, **Trigonometrical Series**, Dover, 1955.