Chapter 11: Fourier Series, Discrete Fourier Transform and Fast Fourier Transform

In general, curve fitting through a set of data points can be done by a linear combination of polynomial functions, with based functions $1, x, x^2, ..., x^m$. In this chapter, however, trigonometric functions such as $1, \cos(x), \cos(2x), ...\cos(nx), \sin(x), \sin(2x), ..., \sin(nx)$ will be used as based functions. In the former, the unknown coefficients of based functions can be found by solving the associated linear simultaneous equations (where the number of unknown coefficients will be matched with the same number of equations, provided by a set of given data points). In the later, however, the unknown coefficients can be efficiently solved (by exploiting special properties of trigonometric functions) without requiring to solve the expensive simultaneous linear equations.

11.1 Background

The following relationships can be readily established, and will be used in subsequent sections for derivation of useful formulas for the unknown Fourier coefficients, in both time and frequency domains.

$$\int_0^T \sin(kw_0 t)dt = \int_0^T \cos(kw_0 t)dt = 0 \quad \text{.....(11.1)}$$

$$\int_0^T \sin^2(kw_0 t)dt = \int_0^T \cos^2(kw_0 t)dt = \frac{T}{2} \quad \text{.....(11.2)}$$

$$\int_0^T \cos(kw_0 t)\sin(gw_0 t)dt = 0 \quad \text{.....(11.3)}$$

$$\int_0^T \sin(kw_0 t)\sin(gw_0 t)dt = 0 \quad \text{.....(11.4)}$$

$$\int_0^T \cos(kw_0 t)\cos(gw_0 t)dt = 0 \quad \text{.....(11.5)}$$

In Eqs (11.1 – 11.5), one has

$$w_0 = 2\pi f \quad \text{.....(11.6)}$$

$$f = \frac{1}{T} \quad \text{.....(11.7)}$$

where f and T represents the frequency (in cycles/time) and period (in seconds) respectively.

A periodic function f(t) with a period T should satisfy the following equation

$$f(t+T) = f(t)$$

Proof of Eq.(11.1)

Let $A = \int_0^T \sin(kw_0 t)dt = -\left(\frac{1}{kw_0}\right)\left[\cos(kw_0 t)\right]_0^T$

$A = \left(\frac{-1}{kw_0}\right)\left[\cos(kw_0 T) - \cos(0)\right] = \left(\frac{-1}{kw_0}\right)\left[\cos(k2\pi) - 1\right] = 0$

Proof of Eq. (11.2)

Let $B = \int_0^T \sin^2(kw_0 t)dt$

Since $\cos(2\alpha) = \cos(\alpha + \alpha) = \cos^2(\alpha) - \sin^2(\alpha) = 1 - 2\sin^2(\alpha)$

Hence $\sin^2(\alpha) = \frac{1 - \cos(2\alpha)}{2}$

Thus: $B = \int_o^T \left[\frac{1}{2} - \frac{1}{2}\cos(2kw_0 t)\right]dt = \left[\left(\frac{1}{2}\right)t - \left(\frac{1}{2}\right)\left(\frac{1}{2kw_0}\right)\sin(2kw_0 t)\right]_0^T$

$B = \left[\frac{T}{2} - \frac{1}{4kw_0}\sin(2kw_0 T)\right] - [0] = \frac{T}{2} - \left(\frac{1}{4kw_0}\right)\sin(2k * 2\pi) = \frac{T}{2}$

Proof of Eq. (11.3)

Let $C = \int_0^T \sin(gw_0 t)\cos(kw_0 t)dt$ ……..…….(11.8)

Recalled:

$\sin(\alpha + \beta) = \sin(\alpha)\cos(\beta) + \sin(\beta)\cos(\alpha)$

Hence:

$C = \int_0^T \left[\sin\left[(g + k)w_0 t\right] - \sin(kw_0 t)\cos(gw_0 t)\right]dt$

$C = \int_0^T \sin\left[(g + h)w_0 t\right]dt - \int_0^T \sin(kw_0 t)\cos(gw_0 t)dt$

C=0 $(see Eq.11.1) - \int_0^T \sin(kw_0 t)\cos(gw_0 t)dt$ …….(11.9)

Adding Eqs.(11.8,11.9), side by side one obtains

$$2C = \int_0^T \left[\sin(gw_0t)\cos(kw_0t) - \sin(kw_0t)\cos(gw_0t)\right]dt$$

$$2C = \int_0^T \sin\left[(gw_0t) - (kw_0t)\right]dt = \int_0^T \sin\left[(g-k)w_0t\right]dt$$

2C = 0, since the right side of the above equation is zero (see Eq.11.1). Thus,

$$C = \int_o^T \sin(gw_0t)\cos(kw_0t)dt = 0$$

Proof of Eq.(11.4)

Let $D = \int_0^T \sin(kw_0t)\sin(gw_0t)dt$ …….(11.10)

Since $\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$

or $\sin(\alpha)\sin(\beta) = \cos(\alpha)\cos(\beta) - \cos(\alpha+\beta)$

Thus,

$$D = \int_0^T \left\{\cos(kw_0t)\cos(gw_0t) - \cos(kw_0t + gw_0t)\right\}dt$$

$$D = \int_0^T \cos(kw_0t)\cos(gw_0t)dt - \int_0^T \cos\left[(k+g)w_0t\right]dt$$

$$D = \int_0^T \cos(kw_0t)\cos(gw_0t)dt - 0 \quad \text{(see Eq.(11.1))}\text{……..……..…...(11.11)}$$

Adding Eqs. (11.10, 11.11) side by side, one obtains:

$$2D = \int_0^T \left\{\cos(kw_0t)\cos(gw_0t) + \sin(kw_0t)\sin(gw_0t)\right\}dt$$

$$2D = \int_0^T \cos\left[kw_0t - gw_0t\right]dt = \int_0^T \cos\left[(k-g)w_0t\right]dt$$

2D = 0, since the right side of the above equation is zero (see Eq.11.1). Thus,

$$D \equiv \int_0^T \sin(kw_0t)\sin(gw_0t)dt = 0$$

Proof of Eq.(11.5)

Eq(11.5) can be proved in a same fashion as the proof for Eq(11.4)

11.2 Fourier Series, and Discrete Fourier Transforms (DFT).

For a function with period T, a continuous Fourier series can be expressed as

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kw_0 t) + b_k \sin(kw_0 t) \quad \text{.....(11.12)}$$

The unknown Fourier coefficients a_0, a_k and b_k can be computed as

$$a_0 = \left(\frac{1}{T}\right)\int_o^T f(t)dt \quad \text{.....(11.13)}$$

Thus, a_0 can be interpreted as the "average" function value between the period interval [0,T].

$$a_k = \left(\frac{2}{T}\right)\int_0^T f(t)\cos(kw_0 t)dt \equiv a_{-k} \quad \text{.....(11.14)}$$

$$b_k = \left(\frac{2}{T}\right)\int_0^T f(t)\sin(kw_0 t)dt \equiv -b_{-k} \quad \text{.....(11.15)}$$

(A) Derivation of formulas for $a_{0,} a_k$ and b_k

Integrating both sides of Eq.(11.12) with respect to time, one gets

$$\int_0^T f(t)dt = \int_0^T a_0 dt + \int_0^T \sum_{k=1}^{\infty} a_k \cos(kw_0 t)dt + \int_0^T \sum_{k=1}^{\infty} b_k \sin(kw_0 t)dt \quad \text{.....(11.16)}$$

The second and third terms on the right hand side of the above equations are both zeros, due to earlier results stated in Eq.(11.1)

Thus:

$$\int_0^T f(t)dt = \left[a_0 t\right]_0^T = a_0 T$$

Hence:

$$a_0 = \left(\frac{1}{T}\right)\int_0^T f(t)dt \quad \text{.....(11.13, repeated)}$$

Now, if both sides of Eq.(11.12) are multiplied by $\sin(mw_0 t)$ and then integrated with respect to time, one obtains:

$$\int_0^T f(t) * \sin(mw_0 t)dt = \int_0^T a_0 \sin(mw_0 t)dt + \int_0^T \sum_{k=1}^{\infty} a_k \cos(kw_0 t)\sin(mw_0 t)dt$$

$$+ \int_0^T \sum_{k=1}^{\infty} b_k \sin(kw_0 t)\sin(mw_0 t)dt \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.17)$$

Due to Eqs. (11.1, 11.3), the first and second terms on the right hand side (RHS) of Eq(11.17) are zero.

Due to Eq. (11.4), the third RHS term of Eq.(11.17) is also zero, with the exception when k=m, which will become (by referring to Eq.11.2):

$$\int_0^T f(t)\sin(kw_0 t)dt = 0 + 0 + \int_0^T b_k \sin^2(kw_0 t)dt = b_k * \frac{T}{2}$$

Thus:

$$b_k = \left(\frac{2}{T}\right)\int_0^T f(t)\sin(kw_0 t)dt \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.15\text{, repeated})$$

Similar derivation can be used to obtain a_k, as shown in Eq.(11.14)

(B) A Fortran Program for finding Fourier Coefficients a_0, a_k, b_k

Based upon the derived formulas for $a_0, a_k,$ and b_k (shown in Eqs. 11.13-11.15, respectively). A FORTRAN computer program has been developed (refer to Table 11.1 for a complete source code listing) and tested for several class examinations in the past several years. Major descriptions of the Fourier program can be summarized as

(a) Input Descriptions (See Example 11.2)

The following input information are required in the input data file:

. Period = 2*3.1416 (assumed); nterms=8 (assumed, for a_k and b_k)

. nsegments = 3 (to determine the given periodic function)

. integration limits for all segments = $-\pi, \frac{-\pi}{2}, \frac{\pi}{2}, \pi$

. descriptions of given periodic function in each segment, defined in subroutine_f

function = $\frac{-\pi}{2}$; for the 1^{st} segment.

function = -t ; for the 2^{nd} segment.

function = $\frac{-\pi}{2}$; for the 3^{rd} segment.

(b) Output Descriptions: (See Example 11.2)

The numerical values of the unknown Fourier Coefficients $a_0, a_1, a_2, ..., a_k, b_1, b_2 ...., b_k$ will be printed.

(c) Users' Internet Access for computer simulations of Fourier Coefficients

can be found at the following website
www.lions.odu.edu/~amoha006/numerical_methods.

Table 11.1 FORTRAN Listings of Fourier Coefficient Program

```
program ce305           ! Updated Version = August 7, 2008
      implicit real*8(a-h,o-z)
      dimension alimit_int(10), ff(10), ak(10), bk(10) ! for Fourier series
c
      write(6,*) '==================================='
      write(6,*) 'Name: Duc T. NGUYEN; TODAY Date: 08/07/2008'
      write(6,*) 'Course: Numerical Methods'
      write(6,*) '==================================='
c.....Fourier series, with N (max N = 3) segments for integration
      pai=3.14159
      period=2.0*pai
      angfreq=2.0*pai/period
      ntrapezoid=1234
      nterms_ak=8               ! maximum = 10
      nterms=nterms_ak
c-------------------------------------------------------------------------------------------------------
c.....test (Fall'2008 semester)
c.....user's input to define: # segments, and integration limits
      nsegments=3
      alimit_int(1)=-pai
      alimit_int(2)=-pai/2.d0
      alimit_int(3)= pai/2.d0
      alimit_int(nsegments+1)=+pai
c-------------------------------------------------------------------------------------------------------
c
      write(6,*) 'nsegments,period,angfreq,nterms for FOURIER coeff. ='
      write(6,*)  nsegments,period,angfreq,nterms
      write(6,*) '(alimit_int(i),i=1,nsegments+1)'
      write(6,*)  (alimit_int(i),i=1,nsegments+1)
c
      iaoakbk=0          ! for computing a0
      call area_under_curve(nsegments, pai, period, angfreq,
     $                      ntrapezoid,
     $                      nterms_ak, alimit_int, ff, a0, ak, bk,
     $                      area,iaoakbk)
c
      iaoakbk=1          ! for computing ak
      call area_under_curve(nsegments, pai, period, angfreq,
     $                      ntrapezoid,
     $                      nterms_ak, alimit_int, ff, a0, ak, bk,
     $                      area,iaoakbk)
c
      iaoakbk=2          ! for computing bk
      call area_under_curve(nsegments, pai, period, angfreq,
```

```
   $                    ntrapezoid,
   $                    nterms_ak, alimit_int, ff, a0, ak, bk,
   $                    area,iaoakbk)
c
 999   stop
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine area_under_curve(nsegments, pai, period, angfreq,
   $                    ntrapezoid,
   $                    nterms_ak, alimit_int, ff, a0, ak, bk,
   $                    area,iaoakbk)
      implicit real*8(a-h,o-z)
      dimension ff(10)
      dimension alimit_int(10), ak(10), bk(10) ! for Fourier series
c
      nfourier_series=nterms_ak
      if (iaoakbk .eq. 0) nfourier_series=1
c
      do 1 k=1, nfourier_series
c
      area=0.d0               ! initialized value
c
        do 2 i=1,nsegments
       a=alimit_int(i)
       b=alimit_int(i+1)
        deltat=(b-a)/ntrapezoid
        t=a-deltat
c.....
          do 3 m=1, ntrapezoid
          t=t+deltat            ! Thus, t will start at value = "a"
          t1=t
          t2=t1+deltat
      call periodic_f(i, t1, function,alimit_int,k,nsegments)
      ff(i)=function
      call periodic_f(i, t2, function,alimit_int,k,nsegments)
      ff(i+1)=function
c..... compute ak
        if (iaoakbk .eq. 1) then
        ff(i)=ff(i)*cos(k*angfreq*t1)
        ff(i+1)=ff(i+1)*cos(k*angfreq*t2)
c..... compute bk
        elseif (iaoakbk .eq. 2) then
        ff(i)=ff(i)*sin(k*angfreq*t1)
        ff(i+1)=ff(i+1)*sin(k*angfreq*t2)
        endif
c
```

```
       area = area + ( ff(i)+ff(i+1) ) * deltat/2.d0
 3        continue
c
 2      continue
c
c      write(6,*) 'iaoakbk, k, area = ',iaoakbk, k, area
c
      if (iaoakbk .eq. 0) then
      aa0=area/period
      write(6,*) 'a0 = ', aa0
      write(6,*) '----------------------'
      elseif (iaoakbk .eq. 1) then
      aak=area*2.d0/period
      write(6,*) 'ak(',k,') = ', aak
      write(6,*) '----------------------'
      elseif (iaoakbk .eq. 2) then
      bbk=area*2.d0/period
      write(6,*) 'bk(',k,') = ', bbk
      endif
c
 1     continue
c
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine periodic_f(isegment, t, function,alimit_int,
     $                      kthfourier,nsegments)
      implicit real*8(a-h,o-z)
      dimension alimit_int(10)
c......user has to define the periodic function for each & every segment
c......within a period T
       pai=3.14159
       i=isegment
c=============================================================
       if (isegment .eq. 1  .and.  t .eq. alimit_int(1)  .and.
     $       kthfourier .eq. 1) then
c-------------------------------------------------------------------------------------------------------
c......test (Fall'2008 semester)
       write(6,*) 'segments integration limits = '
     $   ,(alimit_int(m),m=1,nsegments+1)
       write(6,*) 'segment #1 '
       write(6,*) 'function = -pai/2 '
       write(6,*) 'segment #2 '
       write(6,*) 'function = -t '
       write(6,*) 'segment #3 '
       write(6,*) 'function = -pai/2 '
```

```
c-----------------------------------------------------------------------------------------------------------
        endif
c====================================================================
       go to (11,12,13),i ! assume integral is splited into max. 3 segments
c...... compute Fourier series coefficient a0 (by default)
c......user's input to define: the function in EACH segment
c-----------------------------------------------------------------------------------------------------------
c......test (Fall'2008 semester)
 11     function=-pai/2.d0      ! user defined function for 1-st segment
       go to 444
 12     function=-t             ! user defined function for 2-nd segment
       go to 444
 13     function=-pai/2.d0      ! user defined function for 3-rd segment
       go to 444
c-----------------------------------------------------------------------------------------------------------
c
 444    continue
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Example 11.1

Using the continuous Fourier series to approximate the following periodic rectangular wave function:

$$f(t)=\begin{cases}-1;\ for\ -\dfrac{T}{2}<t<\dfrac{T}{4}\\ 1;\ for\ -\dfrac{T}{4}<t<\dfrac{T}{4}\\ -1;\ for\ \dfrac{T}{4}<t<\dfrac{T}{2}\end{cases} \quad \text{……..…….(11.17A)}$$

The above periodic function f(f) can be plotted, as shown in Fig.11.1

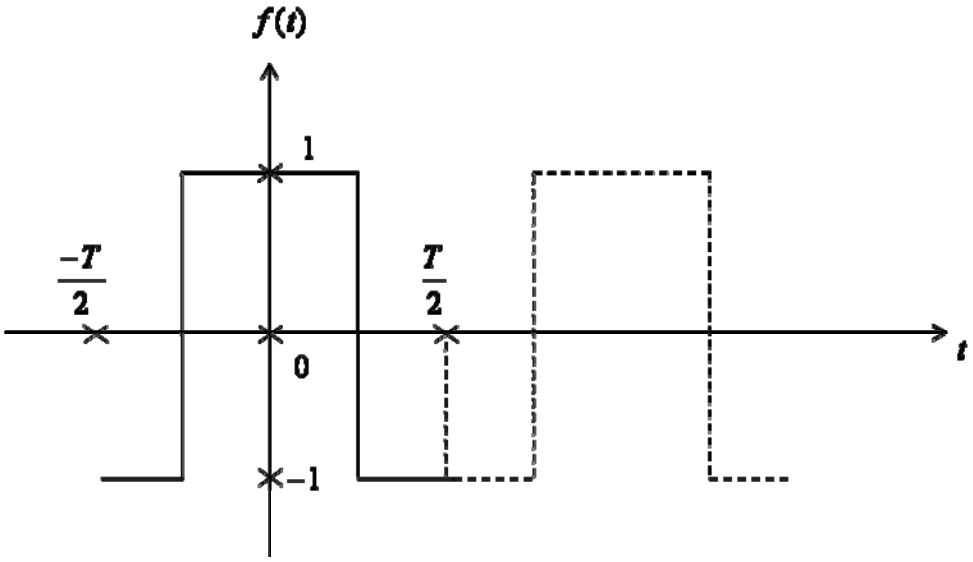


Fig.11.1 A Periodic Rectangular Wave Function.

From Eqs. (11.13-11.15), one obtains :

$$a_0=\left(\frac{1}{T}\right)\int_{\frac{-T}{2}}^{\frac{T}{2}} f(t)dt=\left(\frac{1}{T}\right)\left\{\int_{\frac{-T}{2}}^{\frac{-T}{4}}(-1)dt+\int_{\frac{-T}{4}}^{\frac{T}{4}}(1)dt+\int_{\frac{T}{4}}^{\frac{T}{2}}(-1)dt\right\}=0=a_0$$

The above numerical value for a_0 is expected, since it can be observed from Fig.11.1 that the "average" amplitude of the given periodic function f(t) is zero, for the period interval $\left[\frac{-T}{2},\frac{T}{2}\right]$.

$$b_k = \left(\frac{2}{T}\right)\int_{\frac{-T}{2}}^{\frac{T}{2}} f(t)\sin(kw_0 t)dt = \left(\frac{2}{T}\right)\left\{\int_{\frac{-T}{2}}^{\frac{-T}{4}} (-1)\sin(kw_0 t)dt + \int_{\frac{-T}{4}}^{\frac{T}{4}} (1)\sin(kw_0 t)dt + \int_{\frac{T}{4}}^{\frac{T}{2}} (-1)\sin(kw_0 t)dt\right\}$$

$$b_k = \left(\frac{2}{T}\right)\left(\frac{1}{kw_0}\right) * \left\{\left[\cos(kw_0 t)\right]_{\frac{-T}{2}}^{\frac{-T}{4}} - \left[\cos(kw_0 t)\right]_{\frac{-T}{4}}^{\frac{T}{4}} + \left[\cos(kw_0 t)\right]_{\frac{T}{4}}^{\frac{T}{2}}\right\}$$

or, since $w_0 = \frac{2\pi}{T}$, therefore :

$$b_k = \left(\frac{1}{k\pi}\right)\left\{\begin{array}{l} \cos(k * \frac{2\pi}{T} * \frac{-T}{4} - \cos\left(k * \frac{2\pi}{T} * \frac{-T}{2}\right) \\ -\cos(k * \frac{2\pi}{T} * \frac{T}{4} + \cos\left(k * \frac{2\pi}{T} * \frac{-T}{4}\right) \\ +\cos(k * \frac{2\pi}{T} * \frac{T}{2} - \cos\left(k * \frac{2\pi}{T} * \frac{T}{4}\right) \end{array}\right\}$$

Since cosine is an even function, hence $\cos(\alpha) = \cos(-\alpha)$; the above equation becomes:

$$b_k = \left(\frac{1}{k\pi}\right)\left\{\cos\left(\frac{k\pi}{2}\right) - \cos(k\pi) - \cos\left(\frac{k\pi}{2}\right) + \cos\left(\frac{k\pi}{2}\right) + \cos(k\pi) - \cos\left(\frac{k\pi}{2}\right)\right\}$$

$\boxed{b_k = 0;\ \text{for all k.}}$

$$a_k = \left(\frac{2}{T}\right)\int_{\frac{-T}{2}}^{\frac{T}{2}} f(t)\cos(kw_0 t)dt$$

$$a_k = \left(\frac{2}{T}\right)\left\{\int_{\frac{-T}{2}}^{\frac{-T}{4}}(-1)\cos(kw_0 t)dt + \int_{\frac{-T}{4}}^{\frac{T}{4}}(1)\cos(kw_0 t)dt + \int_{\frac{T}{4}}^{\frac{T}{2}}(-1)\cos(kw_0 t)dt\right\}$$

or

$$a_k = \left(\frac{2}{T}\right)\left(\frac{1}{kw_0}\right)\left\{\left[-\sin(kw_0 t\right]_{\frac{-T}{2}}^{\frac{-T}{4}} + \left[\sin(kw_0 t)\right]_{\frac{-T}{4}}^{\frac{T}{4}} + \left[-\sin(kw_0 t)\right]_{\frac{T}{4}}^{\frac{T}{2}}\right\}$$

since $w_0 = \frac{2\pi}{T}$, *hence*:

$$a_k = \left(\frac{1}{k\pi}\right)\left\{\begin{array}{l} -\sin\left(k*\frac{2\pi}{T}*\frac{-T}{4}\right)+\sin\left(k*\frac{2\pi}{T}*\frac{-T}{2}\right)+\sin\left(k*\frac{2\pi}{T}*\frac{T}{4}\right) \\ -\sin\left(k*\frac{2\pi}{T}*\frac{-T}{4}\right)-\sin\left(k*\frac{2\pi}{T}*\frac{T}{2}\right)+\sin\left(k*\frac{2\pi}{T}*\frac{T}{4}\right) \end{array}\right\}$$

Since sine is an "odd" function, hence $\sin(\alpha) = -\sin(-\alpha)$, the above equation becomes:

$$a_k = \left(\frac{1}{k\pi}\right)\left\{\sin\left(\frac{k\pi}{2}\right)-\sin(k\pi)+\sin\left(\frac{k\pi}{2}\right)+\sin\left(\frac{k\pi}{2}\right)-\sin(k\pi)+\sin\left(\frac{k\pi}{2}\right)\right\}$$

or

$$a_k = \left(\frac{1}{k\pi}\right)\left\{4\sin\left(\frac{k\pi}{2}\right)-2\sin(k\pi)\right\} \quad \text{.....(11.17B)}$$

For k = even integer = 2,4,6…, one gets

$$\boxed{a_{k=2,4,6,..} = 0}$$

For $k = 1 \Rightarrow a_{k=1} = \left(\frac{1}{\pi}\right)\{4-0\} = \frac{4}{\pi} = \frac{4}{(1)\pi}$

For $k = 5 \Rightarrow a_{k=5} = \left(\frac{1}{5\pi}\right)\{4-0\} = \frac{4}{(5)\pi}$

Thus

$$\boxed{a_k = \frac{4}{(k)\pi};\ for\ \text{k} = 1,5,9,\ldots}$$

For $k = 3 \Rightarrow a_{k=3} = \left(\frac{1}{3\pi}\right)\left\{4\sin\left(3*\frac{\pi}{2}\right)-2\sin(3*\pi)\right\}$

or $a_{k=3} = \left(\frac{1}{3\pi}\right)\{-4-0\} = \frac{-4}{(k)\pi}$

For $k = 7 \Rightarrow a_{k=7} = \left(\frac{1}{7\pi}\right)\left\{4\sin\left(7*\frac{\pi}{2}\right) - 2\sin(7*\pi)\right\}$

or $a_{k=7} = \left(\frac{1}{7\pi}\right)\{-4-0\} = \frac{-4}{(k)\pi}$

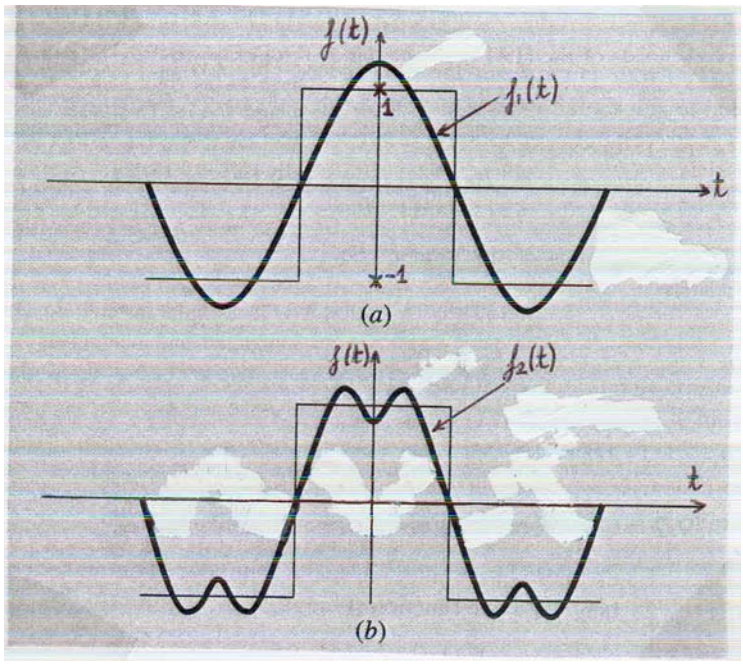
Hence $\boxed{a_k = \frac{-4}{(k)\pi};\ for\ \text{k} = 3,7,11,\ldots}$

In conclusion, the periodic rectangular wave function f(t) (shown in Eq.11.17.A) can be expressed as:

$$f(t) = \sum_{k=1,5,9,\ldots} a_k \cos(kw_0 t) + \sum_{k=3,7,11,\ldots} a_k \cos(kw_0 t)$$

or $f(t) = a_1 \cos(1w_0 t) + a_3 \cos(3w_0 t) + a_5 \cos(5w_0 t).....$

$$f(t) = \frac{4}{1\pi}\cos(1w_0 t) - \frac{4}{3\pi}\cos(3w_0 t) + \frac{4}{5\pi}\cos(5w_0 t).....$$



Notes:

(a) 1-Term Fourier Approximation of a Rectangular Wave Function

$$f(t) \approx f_1(t) = \frac{4}{1\pi}\cos(1w_0 t)$$

(b) 2-Term Fourier Approximation of Rectangular Wave Function

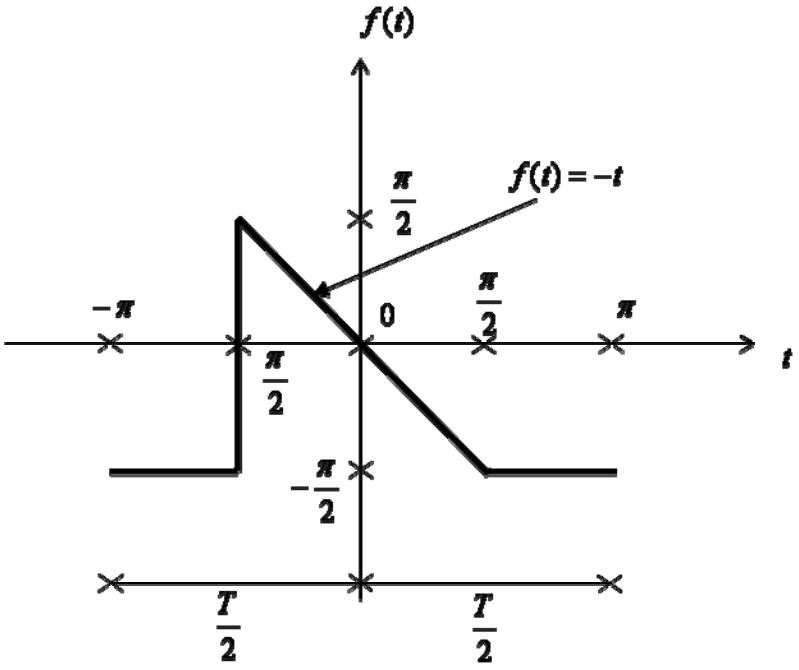
$$f(t) \approx f_2(t) = f_1(t) - \frac{4}{3\pi}\cos(3w_0 t)$$

Example 11.2

The periodic triangular wave function f(t) is defined as

$$f(t)=\begin{cases}-\frac{\pi}{2};\ for-\pi<t<\frac{-\pi}{2}\\ -t;\ for-\frac{-\pi}{2}<t<\frac{\pi}{2}\\ -\frac{\pi}{2};\ for\frac{\pi}{2}<t<\pi\end{cases}$$

Find the Fourier coefficients a_0, a_1, a_2, b_1, b_2 ???



Solutions:

From the developed computer program (see Table 11.1), one gets

$a_0 = -0.785$

$a_1 = 1.00$; $b_1 = -0.64$

$a_2 = 0.00$; $b_2 = -0.50$

$a_3 = -0.333$; $b_3 = 0.071$

$a_4 = 0.00$; $b_4 = 0.25$

$a_5 = 0.20$; $b_5 = -0.025$

$a_6 = 0.00$; $b_6 = -0.167$

$a_7 = -0.143$; $b_7 = 0.013$

$a_8 = 0.00$; $b_8 = 0.125$

(C) Complex Form of the Fourier Series:

Using Euler's identity, the sine and cosine can be expressed in the exponential form as:

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i} = \text{"}odd\text{" } function, \text{ since } \sin(x) = -\sin(-x) \quad \text{…….……(11.18)}$$

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2} = \text{"}even\text{" } function, \text{ since } \cos(x) = \cos(-x) \quad \text{……...….(11.19)}$$

Thus, the Fourier series (expressed in Eq.11.12) can be casted in the following form:

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k * \left(\frac{e^{ikw_0 t} + e^{-ikw_0 t}}{2} \right) + b_k * \left(\frac{e^{ikw_0 t} - e^{-ikw_0 t}}{2i} \right) \quad \text{…….(11.20)}$$

or

$$f(t) = a_0 + \sum_{k=1}^{\infty} e^{ikw_0 t} * \left(\frac{a_k}{2} + \frac{b_k}{2i} * \frac{i}{i} \right) + e^{-ikw_0 t} * \left(\frac{a_k}{2} - \frac{b_k}{2i} * \frac{i}{i} \right)$$

or, since $i^2 = -1,$ one obtains:

$$f(t) = a_0 + \sum_{k=1}^{\infty} e^{ikw_0 t} * \left(\frac{a_k - ib_k}{2} \right) + e^{-ikw_0 t} * \left(\frac{a_k + ib_k}{2} \right) \quad \text{……...……(11.21)}$$

Define the following constants:

$$\tilde{C}_0 \equiv a_0 \quad \text{……...….....(11.22)}$$

$$\tilde{C}_k \equiv \frac{a_k - ib_k}{2} \quad \text{…….(11.23)}$$

Hence:

$$\tilde{C}_{-k} \equiv \frac{a_{-k} - ib_{-k}}{2} \quad \text{……..…….……(11.24)}$$

Using the even, odd properties shown in Eqs. (11.14, 11.15), respectively,

Eq. (11.24) becomes:

$$\tilde{C}_{-k} \equiv \frac{a_k + ib_k}{2} \quad \text{……...(11.25)}$$

Substituting Eqs. (11.22,11.23,11.25) into Eq. (11.21), one gets:

$$f(t) = \tilde{C}_0 + \sum_{k=1}^{\infty} \tilde{C}_k e^{ikw_0 t} + \sum_{k=1}^{\infty} \tilde{C}_{-k} e^{-ikw_0 t}$$

$$f(t) = \sum_{k=0}^{\infty} \tilde{C}_k e^{ikw_0 t} + \sum_{k=-1}^{-\infty} \tilde{C}_k e^{ikw_0 t}$$

$$f(t) = \sum_{k=0}^{\infty} \tilde{C}_k e^{ikw_0 t} + \sum_{k=-\infty}^{-1} \tilde{C}_k e^{ikw_0 t}$$

or

$$f(t) = \sum_{k=-\infty}^{\infty} \tilde{C}_k e^{ikw_0 t} \quad \text{…….(11.26)}$$

The coefficient $\tilde{C}_k$ can be computed, by substituting Eqs.(11.14,11.25) into Eq.(11.23) to obtain:

$$\tilde{C}_k = \left(\frac{1}{2}\right)\left(\frac{2}{T}\right)\left\{\int_0^T f(t)\cos(kw_0 t)dt - i\int_0^T f(t)\sin(kw_0 t)dt\right\} \quad \text{.....(11.27)}$$

or

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * \left[\cos(kw_0 t) - i\sin(kw_0 t)\right]dt\right\}$$

Substituting Eqs. (11.18,11.19) into the above equation, one gets:

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * \left[\frac{e^{ikw_0 t} + e^{-ikw_0 t}}{2} - i * \frac{e^{ikw_0 t} - e^{-ikw_0 t}}{2i}\right]dt\right\}$$

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * e^{-ikw_0 t}dt\right\} \quad \text{.....(11.28)}$$

Thus, Eqs. (11.26,11.28) are the equivalent complex version of Eqs.(11.12-11.15).

(D) Fourier Transform Pair

As up to this point, Fourier approximation has been expressed in the time domain. The amplitude (vertical axis) of a periodic function can be plotted versus time (horizontal axis), but it can also be plotted versus frequency (horizontal axis).

The periodic rectangular wave function expressed in the time domain (see Fig.11.1), can also be plotted in the frequency domain as shown in Fig.11.2.

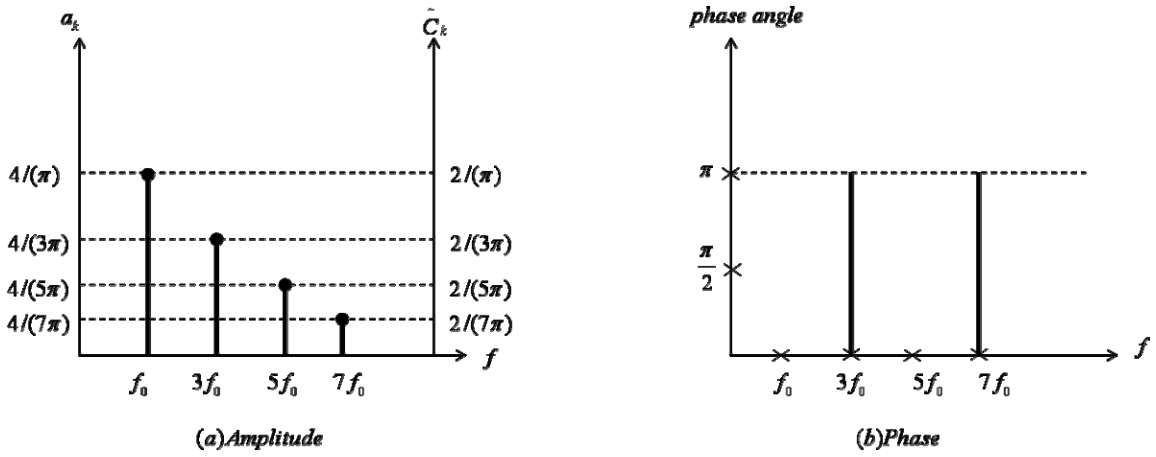


Figure 11.2 Periodic Rectangular Wave Function in Frequency Domain.

Explanation of Figures 11.2(a) and 11.2(b)

$$f(t) = \sum_{k=-\infty}^{\infty} \tilde{C}_k e^{ikw_0 t}$$

where

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * e^{-ikw_0 t} dt\right\}$$

For the periodic function shown in Example 11.1 (or Figure 11.1), one has:

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_{-\frac{T}{2}}^{-\frac{T}{4}} (-1) * e^{-ikw_0 t} dt + \int_{-\frac{T}{4}}^{\frac{T}{4}} (1) * e^{-ikw_0 t} dt + \int_{\frac{T}{4}}^{\frac{T}{2}} (-1) * e^{-ikw_0 t} dt\right\}$$

or

$$\tilde{C}_k = \left(\frac{1}{T}\right)\{A + B + C\}$$

where, making use of $w_0 = \frac{2\pi}{T}$; one obtains:

$$A \equiv \int_{-\frac{T}{2}}^{-\frac{T}{4}} - e^{-ikw_0 t} dt = \left(\frac{1}{ikw_0}\right)\left[e^{\frac{ik\pi}{2}} - e^{ik\pi}\right]$$

$$B \equiv \int_{-\frac{T}{4}}^{\frac{T}{4}} e^{-ikw_0 t} dt = \left(\frac{1}{ikw_0}\right)\left[-e^{\frac{-ik\pi}{2}} + e^{\frac{ik\pi}{2}}\right]$$

$$C \equiv \int_{\frac{T}{4}}^{\frac{T}{2}} - e^{-ikw_0 t} dt = \left(\frac{1}{ikw_0}\right)\left[e^{-ik\pi} - e^{\frac{-ik\pi}{2}}\right]$$

Hence:

$$\tilde{C}_k = \left(\frac{1}{ik2\pi}\right)\left\{2e^{\frac{ik\pi}{2}} - 2e^{\frac{-ik\pi}{2}} + e^{-ik\pi} - e^{ik\pi}\right\}$$

Recalled:

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

$$e^{-i\theta} = \cos(-\theta) + i\sin(-\theta) = \cos(\theta) - i\sin(\theta)$$

Then, the above equation for $\tilde{C}_k$ can be expressed as:

$$\tilde{C}_k = \left(\frac{1}{ik2\pi}\right)\left\{\begin{array}{l} 2\left\{\cos\left(\frac{k\pi}{2}\right) + i\sin\left(\frac{k\pi}{2}\right)\right\} - 2\left\{\cos\left(\frac{+k\pi}{2}\right) - i\sin\left(\frac{+k\pi}{2}\right)\right\} \\ + \{\cos(+k\pi) - i\sin(+k\pi)\} - \{\cos(k\pi) + i\sin(+k\pi)\} \end{array}\right\}$$

$$\tilde{C}_k = \left(\frac{1}{ik2\pi}\right)\left\{4i\sin\left(\frac{k\pi}{2}\right) - 2i\sin(k\pi)\right\}$$

$$\tilde{C}_k = \left(\frac{1}{2k\pi}\right)\left\{4\sin\left(\frac{k\pi}{2}\right) - 2\sin(k\pi)\right\} = \text{real number.}$$

Since $\tilde{C}_k = \frac{a_k - ib_k}{2} = \frac{a_k}{2}$; because $\tilde{C}_k$ = real number

Hence

$a_k = 2\tilde{C}_k$

For $k = 1 \Rightarrow \tilde{C}_1 = \left(\frac{1}{2\pi}\right)[4] = \frac{2}{\pi} \Rightarrow a_1 = \frac{4}{\pi} = \left(\frac{4}{\pi}\right)e^{i(0)}$

Hence the amplitude and phase angle are $\frac{4}{\pi}$ and (0) radian, respectively.

For $k = 2 \Rightarrow \tilde{C}_2 = \left(\frac{1}{4\pi}\right)[0] = 0 \Rightarrow a_2 = 0$

For $k = 3 \Rightarrow \tilde{C}_3 = \left(\frac{1}{6\pi}\right)[-4] = \frac{-2}{3\pi} \Rightarrow a_3 = \frac{-4}{3\pi} = \left(\frac{4}{3\pi}\right)e^{i(\pi)}$

Hence the amplitude and phase angle are $\frac{4}{3\pi}$ and (π) radian, respectively.

For $k = 4 \Rightarrow \tilde{C}_4 = \left(\frac{1}{8\pi}\right)[0] = 0 \Rightarrow a_4 = 0$

For $k = 5 \Rightarrow \tilde{C}_5 = \left(\frac{1}{10\pi}\right)[4] = \frac{4}{10\pi} = \frac{2}{5\pi} \Rightarrow a_5 = \frac{4}{5\pi} = \left(\frac{4}{5\pi}\right)e^{i(0)}$

Hence the amplitude and phase angle are $\frac{4}{5\pi}$ and (0) radian, respectively.

For $k = 6 \Rightarrow \tilde{C}_6 = \left(\frac{1}{12\pi}\right)[0] = 0 \Rightarrow a_6 = 0$

For $k = 7 \Rightarrow \tilde{C}_7 = \left(\frac{1}{14\pi}\right)[-4] = \frac{-2}{7\pi} \Rightarrow a_7 = \frac{-4}{7\pi} = \left(\frac{4}{7\pi}\right)e^{i(\pi)}$

Hence the amplitude and phase angle are $\frac{4}{7\pi}$ and (π) radian, respectively.

<u>Remarks:</u>

For k =0; then

$$a_0 = \left(\frac{1}{T}\right)\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)dt = 0 \text{ (See Example 11.1)}$$

(E) Non-Periodic Function

Recalled that a periodic function can be expressed in terms of the exponential form, accordingly to Eqs. (11.26,11.28) as :

$$f(t) = \sum_{k=-\infty}^{\infty} \tilde{C}_k \, e^{ikw_0 t} \quad \text{……..(11.26, repeated)}$$

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * e^{-ikw_0 t}\, dt\right\} \quad \text{……...(11.28, repeated)}$$

Define the following function:

$$x(ikw_0) = \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-ikw_0 t}\, dt \quad \text{…….(11.29)}$$

Then, Eq. (11.28) can be written as:

$$\tilde{C}_k = \left(\frac{1}{T}\right) * x(ikw_0) \quad \text{……(11.30)}$$

And Eq.(11.26) becomes

$$f(t) = \sum_{k=-\infty}^{\infty} \left(\frac{1}{T}\right) * x(ikw_0) e^{ikw_0 t} \quad \text{……....(11.31)}$$

A non-periodic function f_{np} can be considered as a periodic function, with the period

$T \to \infty$, or $\Delta f \equiv \frac{1}{T} \to 0$ (See Fig 11.3)

From Eqs. (11.6-11.7), one gets:

$$w_0 = 2\pi f = \frac{2\pi}{T} = 2\pi(\Delta f) \quad \text{……...(11.32)}$$

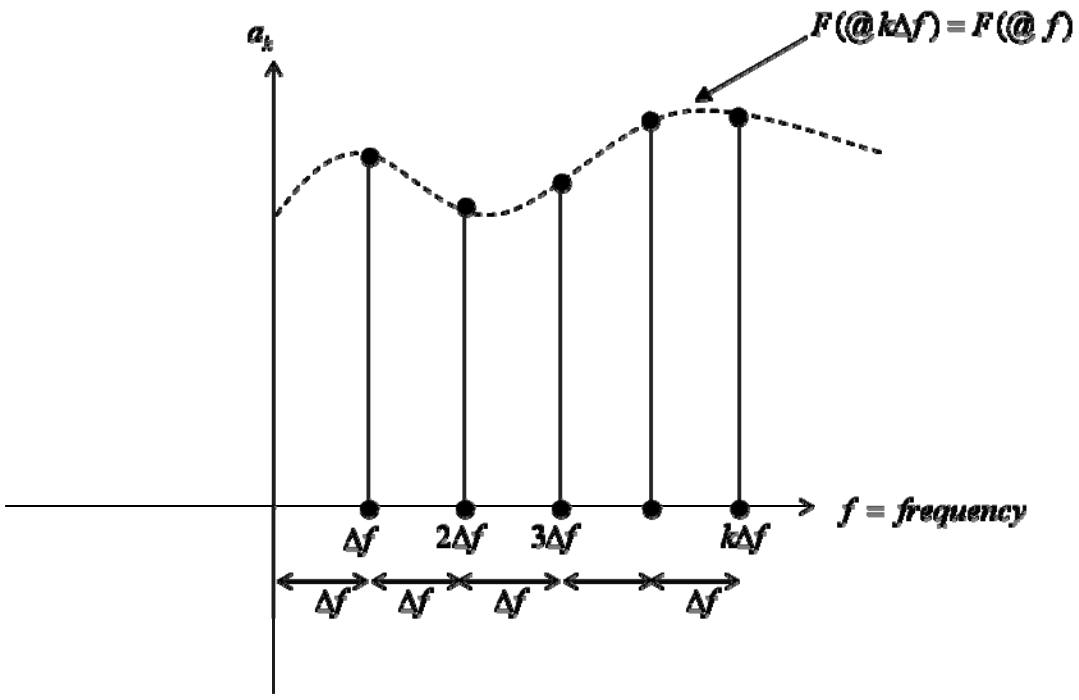


Fig. 11.3 : Frequency are Discretized.

From Eq.(11.31), one obtains:

$$f_{np}(t) = \lim_{\substack{T\to\infty \\ or\,\Delta f\to 0}} f(t) = \lim_{\Delta f\to 0} \sum_{k=-\infty}^{\infty} (\Delta f) * x(ikw_0) e^{ikw_0 t} \quad \text{……..(11.33)}$$

or,

$$f_{np}(t) = \lim_{\Delta f\to 0} \sum_{k=-\infty}^{\infty} (\Delta f) * x(ik2\pi\Delta f) e^{ik2\pi\Delta f t} \quad \text{…….....(11.34)}$$

$$f_{np}(t) = \int df * x(i2\pi f) e^{i2\pi f t}$$

$$f_{np}(t) = \int x(i2\pi f) e^{i2\pi f t} df \quad \text{……(11.35)}$$

$$f_{np}(t) = \left(\frac{1}{2\pi}\right) \int x(i2\pi f) e^{i2\pi f t} d(2\pi f)$$

$$f_{np}(t) = \left(\frac{1}{2\pi}\right) \int_{-\infty}^{\infty} x(iw_0) e^{iw_0 t} d(w_0) \text{ ; inverse Fourier transform……...…….(11.36)}$$

Using the definition stated in Eq.(11.29), one has

$$x(iw_0) = \int_{-\infty}^{\infty} f_{np}(t) e^{-iw_0 t} d(t) \text{ ; Fourier transform……...(11.37)}$$

Thus, Eqs. (11.37,11.36) will transform a non-periodic function from time domain to frequency domain, and from frequency domain to time domain, respectively.

(F) Discrete Fourier Transform (DFT)

Recalled the exponential form of Fourier series (see Eqs.11.26,11.28), one gets:

$$f(t) = \sum_{k=-\infty}^{\infty} \tilde{C}_k e^{ikw_0 t} \quad \text{.....(11.26, repeated)}$$

$$\tilde{C}_k = \left(\frac{1}{T}\right)\left\{\int_0^T f(t) * e^{-ikw_0 t} dt\right\} \quad \text{.....(11.28, repeated)}$$

If time "t" is discretized at $t_1 = \Delta t, t_2 = 2\Delta t, t_3 = 3\Delta t, ..., t_n = n\Delta t,$

Then Eq.(11.26) becomes:

$$f(t_n) = \sum_{k=0}^{N-1} \tilde{C}_k e^{ikw_0 t_n} \quad \text{.....(11.38)}$$

To simplify the notation, define:

$$t_n = n \quad \text{.....(11.39)}$$

Then, Eqs.(11.38) can be written as:

$$f(n) = \sum_{k=0}^{N-1} \tilde{C}_k e^{ikw_0 n} \quad \text{.....(11.40)}$$

Multiplying both sides of eq.(11.40) by $e^{-ilw_0 n}$, and performing the summation on "n", one obtains (note:l = integer number)

$$\sum_{n=0}^{N-1} f(n) * e^{-ilw_0 n} = \sum_{n=0}^{N-1}\sum_{k=0}^{N-1} \tilde{C}_k e^{ikw_0 n} * e^{-ilw_0 n} \quad \text{.....(11.41)}$$

or

$$\sum_{n=0}^{N-1} f(n) * e^{-ilw_0 n} = \sum_{n=0}^{N-1}\sum_{k=0}^{N-1} \tilde{C}_k e^{i(k-l)w_0 n} \quad \text{.....(11.42)}$$

$$= \sum_{n=0}^{N-1}\sum_{k=0}^{N-1} \tilde{C}_k e^{i(k-l)\frac{2\pi}{N} n} \quad \text{.....(11.43)}$$

Switching the order of summations on the right-hand-side of Eq.(11.43), one obtains:

$$\sum_{n=0}^{N-1} f(n) * e^{-il\left(\frac{2\pi}{N}\right)n} = \sum_{k=0}^{N-1} \tilde{C}_k \sum_{n=0}^{N-1} e^{i(k-l)\left(\frac{2\pi}{N}\right)n} \quad \text{.....(11.44)}$$

Define:

$$A = \sum_{n=0}^{N-1} e^{i(k-l)\left(\frac{2\pi}{N}\right)n} \quad \text{.....(11.45)}$$

There are 2 possibilities for (k-l) to consider in Eq. (11.45)

Case(1): (k-l) is a multiple integer of N, such as:

(k-l)=mN; or k=l+mN where $m = 0,\pm1,\pm2,.....$

Thus, Eq.(11.45) becomes:

$$A = \sum_{n=0}^{N-1} e^{im2\pi n} = \sum_{n=0}^{N-1} \cos(mn2\pi) + i\sin(mn2\pi) \quad \text{.....(11.46)}$$

Hence:

A=N……..……..(11.47)

Case(2): (k-l) is NOT a multiple integer of N

In this case, from Eq.(11.45) one has:

$$A = \sum_{n=0}^{N-1} \left\{ e^{i(k-l)\left(\frac{2\pi}{N}\right)} \right\}^n \quad \text{……..(11.48)}$$

Define:

$$a = e^{i(k-l)\frac{2\pi}{N}} = \cos\left\{(k-l)\frac{2\pi}{N})\right\} + i\sin\left\{(k-l)\frac{2\pi}{N})\right\} \quad \text{……..(11.49)}$$

$a \neq 1$; because (k-l) is "NOT" a multiple integer of N……..….(11.50)

Then, Eq. (11.48) can be expressed as:

$$A = \sum_{n=0}^{N-1} \{a\}^n \quad \text{……....(11.51)}$$

From mathematical handbooks, the right side of Eq. (11.51) represents the "geometric series", and can be expressed as:

$$A = \sum_{n=0}^{N-1} \{a\}^n = N; \text{ if } a = 1 \quad \text{……..(11.52)}$$

$$= \frac{1-a^N}{1-a}; \text{ if } a \neq 1 \quad \text{……..……(11.53)}$$

Because of Eq. (11.50), hence Eq. (11.53) should be used to compute A. Thus:

$$A = \frac{1-a^N}{1-a} = \frac{1-e^{i(k-l)2\pi}}{1-a} \text{ (See Eq. (11.49)) } \quad \text{……(11.54)}$$

Since (k-l) is still a multiple of 2π, hence

$$e^{i(k-l)2\pi} \equiv \cos\{(k-l)2\pi\} + i\sin\{(k-l)2\pi\} = 1 \text{.....(11.55)}$$

Substituting Eq. (11.55) into Eq. (11.54), one gets:

A=0……...……..(11.56)

Thus, combining the results of case (1) and case (2), one gets (see Eqs.11.47 and Eq.11.56):

A=N+0=N…….….(11.57)

Substituting Eq.(11.57) into Eq.(11.45), and then referring to Eq(11.44), one gets:

$$\sum_{n=0}^{N-1} f(n)e^{-ilw_0 n} = \sum_{k=0}^{N-1} \tilde{C}_k * N \text{ ……..(11.57A)}$$

Recalled k=l+mN (where l,m are integer numbers), and since k must be in the range $0 \to N-1$, therefore m=0. Thus:

k=l+mN becomes k=l

Eq(11.57A) can, therefore, be simplified to:

$$\sum_{n=0}^{N-1} f(n)e^{-ilw_0 n} = \tilde{C}_l * N \text{ ……...(11.57B)}$$

Thus:

$$\tilde{C}_k = \left(\frac{1}{N}\right)\sum_{n=0}^{N-1} f(n)e^{-ikw_0 n} = \left(\frac{1}{N}\right)\sum_{n=0}^{N-1} f(n)\{\cos(lw_0 n) - i\sin(lw_0 n)\} \text{……..(11.58)}$$

where $n \equiv t_n$

and

$$f(n) = \sum_{k=0}^{N-1} \tilde{C}_k\, e^{ikw_0 n} = \sum_{k=0}^{N-1} \tilde{C}_k \{\cos(kw_0 n) + i\sin(kw_0 n)\} \text{……..(11.38, repeated)}$$

FORTRAN code for computing the DFT, shown in Eq. (11.58) [or similarly shown in Eq.11.38] is listed in Table 11.2.

Remarks:

(a) Consider the exponential term in the above equation [Eq. (11.38, repeated)]. Let

$E = e^{(ikw_0 n)} = e^{(ik*\frac{2\pi}{N}*n)}$;

where $\pi = 3.14159$

If one replaces “n” by “-(N-n)” (or “n-N”) into the above equation, then one obtains:

$$e^{ik*\frac{2\pi}{N}*(n-N)} = e^{(ik*\frac{2\pi}{N}*n)} * \left[e^{(-ik*2\pi)} = 1\right] = E$$

Thus, Eq. (11.38, repeated) indicates that the force corresponding to frequencies of order “n” and “-(N-n) = n-N” have the same values. Hence:

$$w_n = n\overline{w} \quad \text{for } n \le \frac{N}{2}$$

$$= -(N-n)\overline{w} \text{ for } n > \frac{N}{2}$$

and the frequency corresponding to $n = \frac{N}{2}$ is the highest frequency that can be considered in the discrete Fourier series ($w_{\frac{N}{2}}$ is called the Nyquist frequency). If there are harmonic (force) components above $w_{\frac{N}{2}}$ in the original function, then these higher components will introduce distortions in the lower harmonic components (known as ALIASING phenomenon). Because of the ALIASING phenomenon, the number of (N) data points should be “at least twice” the highest harmonic component presents in the (forcing) function, for sufficient computational accuracy. As an example, if the forcing function is given as:

$$F(t) = \sum_{n=1}^{16} 100 * \cos(2\pi nt)$$

then, the minimum value of N (= Number of sample data points) should be $N_{\min} = 32$.

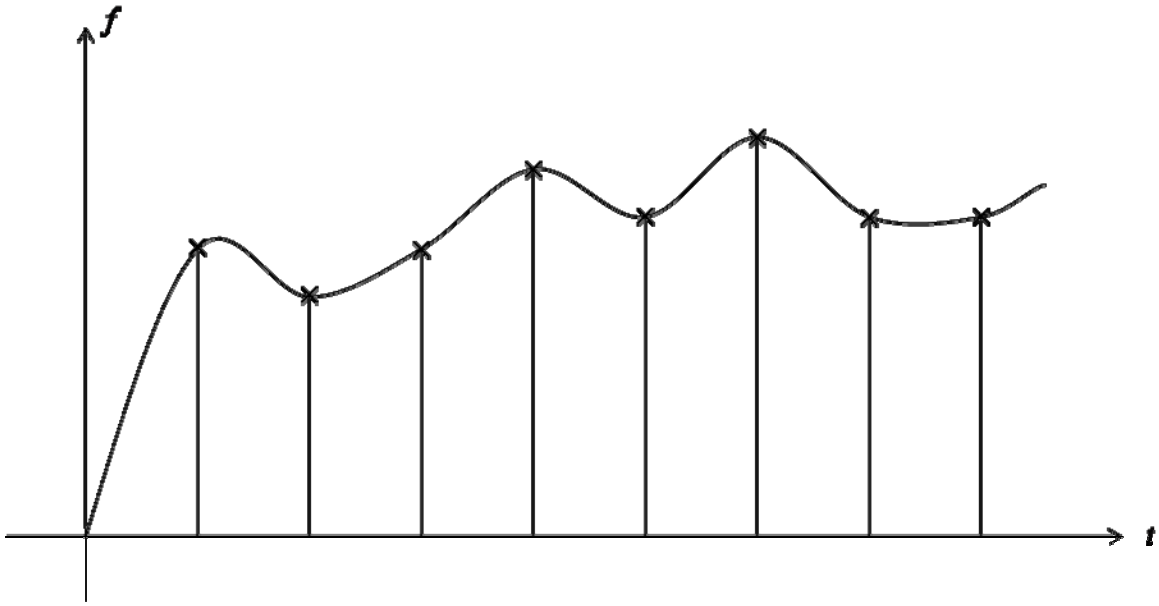


Figure 11.25: Discretize With Large step Size Will Introduce Large Error.

(b) The factor $\left(\frac{1}{N}\right)$, shown in the DFT Eq.(11.58), is merely a scale factor. It can also be placed in the inverse Fourier Transform Eq.(11.38), but not both !

Thus, Eqs. (11.58 & 11.38) can be re-written as:

$$\tilde{C}_n = \sum_{k=0}^{N-1} f(k) e^{-ik\left(w_0 = \frac{2\pi}{N}\right)n} \quad \text{.....(11.59)}$$

$$f(k) = \left(\frac{1}{N}\right) \sum_{n=0}^{N-1} \tilde{C}_n \, e^{ik\left(w_0 = \frac{2\pi}{N}\right)n} \quad \text{.....(11.60)}$$

To avoid computation with "complex numbers", Eq.(11.59) can be expressed as:

$$\tilde{C}_n^R + i\tilde{C}_n^I = \sum_{k=0}^{N-1} \left\{ f^R(k) + i \quad f^I(k) \right\} * \left\{ \cos(\theta) - i \sin(\theta) \right\} \text{.....(11.59A)}$$

where

$$\theta = k\left(w_0 = \frac{2\pi}{N}\right)n \text{.....(11.59B)}$$

$$\tilde{C}_n^R + i\tilde{C}_n^I = \sum_{k=0}^{N-1} \left\{ f^R(k) * \cos(\theta) + f^I(k) \sin(\theta) \right\} + i\left\{ f^I(k) \cos(\theta) - f^R(k) i \sin(\theta) \right\}$$

The above "complex number" equation is equivalent to the following 2 "real number" equations:

$$\tilde{C}_n^R = \sum_{k=0}^{N-1} \left\{ f^R(k) \cos(\theta) + f^I(k) \sin(\theta) \right\} \text{.....(11.59C)}$$

$$\tilde{C}_n^I = \sum_{k=0}^{N-1} \left\{ f^I(k) \cos(\theta) - f^R(k) i \sin(\theta) \right\} \text{.....(11.59D)}$$

Table 11.2 FORTRAN Coding For DFT (See Eqs. 11.59C, 11.59D)

```
c
      implicit real*8(a-h,o-z)
      dimension freal(1000000), fimag(1000000)
      write(6,*) '                              '
      write(6,*) '=============================='
      write(6,*) ' Prof. Nguyen Version Date: 08-08-2008'
      write(6,*) '=============================='
      write(6,*) '                              '
      read(5,*) iautodata, n, igama, method
      write(6,*) 'iautodata,n,igama, method = 1 (FFT); 2(DFT)'
      write(6,*)  iautodata,n,igama, method
      if (iautodata .eq. 1) then
       do 1 i=1,n
       freal(i)=dfloat(i)
       fimag(i)=0.d0
 1     continue
      elseif (iautodata .eq. 0) then
       read(5,*) ( freal(i),i=1,n )
       read(5,*) ( fimag(i),i=1,n )
      endif
c
       write(6,*) 'input data for FFT: i,freal,fimag ='
       do 22 i=1,n
       write(6,*) i, freal(i), fimag(i)
 22     continue
c
       if (method .eq. 1) then
c       call fft(freal,fimag,n,igama)
c
       write(6,*) 'output for FFT: i,freal,fimag ='
       do 23 i=1,n
       write(6,*) i, freal(i), fimag(i)
 23     continue
c
       elseif (method .eq. 2) then
      call dft(freal,fimag,n,igama)
       endif
 999   stop
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine dft(freal,fimag,nn,igama)
      implicit real*8(a-h,o-z)
      dimension freal(*), fimag(*)
c
```

```
      pai=3.14159d0
      w0=2.d0*pai/dfloat(nn)
      sumreal=0.d0
      sumimag=0.d0
      write(6,*) 'dft results: n,freal,fimag = '
       do 1 n=1,nn
       cnreal=0.d0
       cnimag=0.d0
         do 2 k=1,nn
         angle=(k-1)*w0*(n-1)
         c=cos(angle)
         s=sin(angle)
         cnreal=cnreal+freal(k)*c+fimag(k)*s
         cnimag=cnimag+fimag(k)*c-freal(k)*s
 2         continue
      write(6,*) n, cnreal, cnimag
      sumreal=sumreal+dabs(cnreal)
      sumimag=sumimag+dabs(cnimag)
 1     continue
      write(6,*) 'DFT: sumreal,sumimag = ',sumreal,sumimag
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

11.3 Intuitive Development of Fast Fourier Transform (FFT)

Recalled the DFT pairs of Eqs. (11.59,11.60) and swapping the indexes n,k one obtains:

$$\tilde{C}_n = \sum_{k=0}^{N-1} f(k) e^{-in\left(w_0=\frac{2\pi}{N}\right)k} \quad \text{……(11.61)}$$

$$f(k) = \left(\frac{1}{N}\right)\sum_{n=0}^{N-1} \tilde{C}_n \, e^{in\left(w_0=\frac{2\pi}{N}\right)k} \quad \text{……(11.62)}$$

Where n, k = 0,1,2,3,….N-1 ……(11.63)

Let $W = e^{-i\frac{2\pi}{N}}$ (hence $W^N = e^{-i2\pi} = 1$)……(11.64)

Then Eq. (11.61) becomes:

$$\tilde{C}_n = \tilde{C}(n) = \sum_{k=0}^{N-1} f(k) W^{nk} \quad \text{……(11.65)}$$

Assuming $N = 4 = 2^{(r=2)}$, *then*

$$\begin{Bmatrix} \tilde{C}(0) \\ \tilde{C}(1) \\ \tilde{C}(2) \\ \tilde{C}(3) \end{Bmatrix} = \begin{bmatrix} W^{(0)(0)} & W^{(0)(1)} & W^{(0)(2)} & W^{(0)(3)} \\ W^{(1)(0)} & W^{(1)(1)} & W^{(1)(2)} & W^{(1)(3)} \\ W^{(2)(0)} & W^{(2)(1)} & W^{(2)(2)} & W^{(2)(3)} \\ W^{(3)(0)} & W^{(3)(1)} & W^{(3)(2)} & W^{(3)(3)} \end{bmatrix} \begin{Bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{Bmatrix} \quad \text{……(11.66)}$$

$$\begin{Bmatrix} \tilde{C}(0) \\ \tilde{C}(1) \\ \tilde{C}(2) \\ \tilde{C}(3) \end{Bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{Bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{Bmatrix} \quad \text{……(11.67)}$$

For N=4, n=2 and k=3, then:

$$W^{nk} = W^6 = \left[W^{(n=4)}\right] W^2 = \left(e^{\frac{-i2\pi}{N}}\right)^N W^2 = \left[e^{-i2\pi}\right] W^2 = W^2$$

The term inside the square bracket is equal to 1, since

$$[e^{-i2\pi}] = \cos(-2\pi) + i\sin(-2\pi)$$

$$= \cos(2\pi) - i\sin(2\pi)$$

= 1 - i(0) = 1

. For N=4, n=3 and k=3, then

$$W^{nk} = W^{9} = [W^{8}]W^{1} = W^{1}$$

. Thus, in general (for $nk \geq N$)

$W^{nk} = W^{p}$ where p = mod(nk,N) …….(11.68)

or p = remainder of $\left(\frac{nk}{N}\right)$

Remarks:

(a) Matrix times vector, shown in Eq. (11.67), will require 16 (or N^2) complex multiplications and 12 (or N*{N-1}) complex additions.
(b) Usage of Eq. (11.68) will help to reduce the number of operation counts, as explained in the next section.

Factorized Matrix and Further Operation Count:

Eq. (11.67) can be factorized as:

$$\begin{Bmatrix} \tilde{C}(0) \\ \tilde{C}(2) \\ \tilde{C}(1) \\ \tilde{C}(3) \end{Bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{Bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{Bmatrix} \text{ ……...….(11.69)}$$

Remarks:

(a) The theoretical behind the 2 matrices on the right hand side (RHS) of Eq.(11.69) will be clearly explained soon !.
(b) The order of the left-hand-side (LHS) vector has been changed, such as rows 2 and 3 have been swapped !.
(c) Let the row-interchanged LHS vector be defined as:

$$\tilde{C}^{*}(n) = \begin{Bmatrix} \tilde{C}(0) \\ \tilde{C}(2) \\ \tilde{C}(1) \\ \tilde{C}(3) \end{Bmatrix} \text{ …….(11.70)}$$

Now performing the inner-product (matrix times vector) on the RHS of Eq. (11.69), one obtains:

$$\begin{Bmatrix} f_1(0) \\ f_1(1) \\ f_1(2) \\ f_1(3) \end{Bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{Bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{Bmatrix} \quad \text{.....(11.71)}$$

or

$$f_1(0) = f(0) + W^0 f(2) \quad \text{.....(11.71A)}$$
$$f_1(1) = f(1) + W^0 f(3) \quad \text{.....(11.72B)}$$
$$f_1(2) = f(0) + W^2 f(2)$$
$$= f(0) - W^0 f(2) \quad \text{.....(11.72C)}$$

since $W^2 = e^{-i\frac{2\pi}{4}*2} = e^{-i\pi} = -1 = -W^0$

$$f_1(3) = f(1) + W^2 f(3)$$
$$= f(1) - W^0 f(3) \quad \text{.....(11.72D)}$$

Eqs.(11.72A through 11.72D) for the "inner" matrix times vector requires 2 complex multiplications and 4 complex additions.

(d) In Eqs.(11.72A through 11.72D), W^0 is intentionally not reduced to the numerical value of 1.0 in order to facilitate the discussions of more general cases.

Finally, performing the "outer" product (matrix times vector) on the RHS of Eq.(11.69), one obtains:

$$\begin{Bmatrix} \tilde{C}(0) \\ \tilde{C}(2) \\ \tilde{C}(1) \\ \tilde{C}(3) \end{Bmatrix} = \begin{Bmatrix} f_2(0) \\ f_2(1) \\ f_2(2) \\ f_2(3) \end{Bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{Bmatrix} f_1(0) \\ f_1(1) \\ f_1(2) \\ f_1(3) \end{Bmatrix} \quad \text{.....(11.73)}$$

or

$$f_2(0) = f_1(0) + W^0 f_1(1) \quad \text{.....(11.74A)}$$
$$f_2(1) = f_1(0) + W^2 f_1(1) = f_1(0) - W^0 f_1(1) \quad \text{.....(11.74B)}$$
$$f_2(2) = f_1(2) + W^1 f_1(3) \quad \text{.....(11.74C)}$$
$$f_2(3) = f_1(2) + W^3 f_1(3) = f_1(2) + W^2 W^1 f_1(3)$$
$$= f_1(2) - W^1 f_1(3) \quad \text{.....(11.74D)}$$

Again, Eqs (11.74A-11.74D) requires 2 complex multiplications and 4 complex additions. Thus, the complete RHS of Eq.(11.69) can be computed by only 4 complex multiplications (or $N\frac{r}{2} = 4\frac{2}{2}$) and 8 complex additions (or Nr = 4*2). Since computational time is mainly controlled by the number of multiplications, hence implementing Eq.(11.69) will significantly reduce the number of multiplication, as compared to direct matrix times vector operations (as shown in Eq.11.67).

For large value of data points (=N), one obtains:

$$Ratio = \frac{N^2}{\left(\frac{Nr}{2}\right)} = \left(\frac{2N}{r}\right) \qquad \text{.....(11.75)}$$

For $N = 2048 = 2^{(r=11)}$, Eq. (11.75) gives:

$$Ratio = \frac{2(2048)}{11} = 372.36$$

Graphical flow of Eq.(11.69), for case $N = 2^r = 2^2 = 4$

Eq. (11.69) can also be presented in the graphical form, as shown in Figure11.4

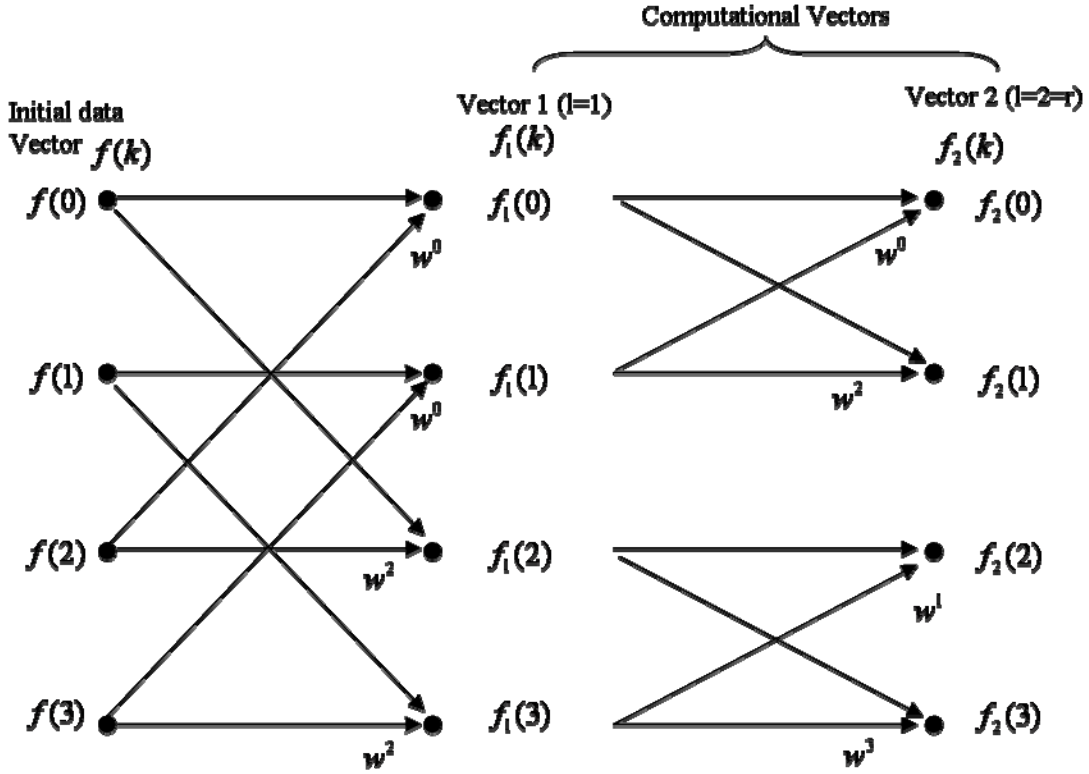


Figure 11.4 Graphical form of FFT (Eq.11.69). For the case $N = 2^r = 2^2 = 4$

Remarks

(a) Computed vector 1 does correspond to Eq.(11.71).
(b) Computed vector 2 does correspond to Eq.(11.74)
(c) Since r = 2 in this example, one needs to compute 2 vectors $\{= f_1(k) and f_2(k)\}$
(d) Each node in the graph is computed from 2(=r) nodes in the "previous" vector.
(e) Factor W^P (such as W^0, W^1, W^2, W^3) appears near the arrow head of the transmission path. Absence of W^P implies that $W^P = W^0 = 1$.
For example: $f_2(2) = f_1(2) + f_1(3)W^1$, which is the same as Eq.(11.74C)

Graphical Flow of Eq.(11.69), for case $N = 2^r = 2^4 = 16$

In order to see a more detailed computational patterns of FFT, a slightly larger data size ($N = 2^r = 2^4 = 16$) is shown in the graphical form, as indicated in Figure 11.5.

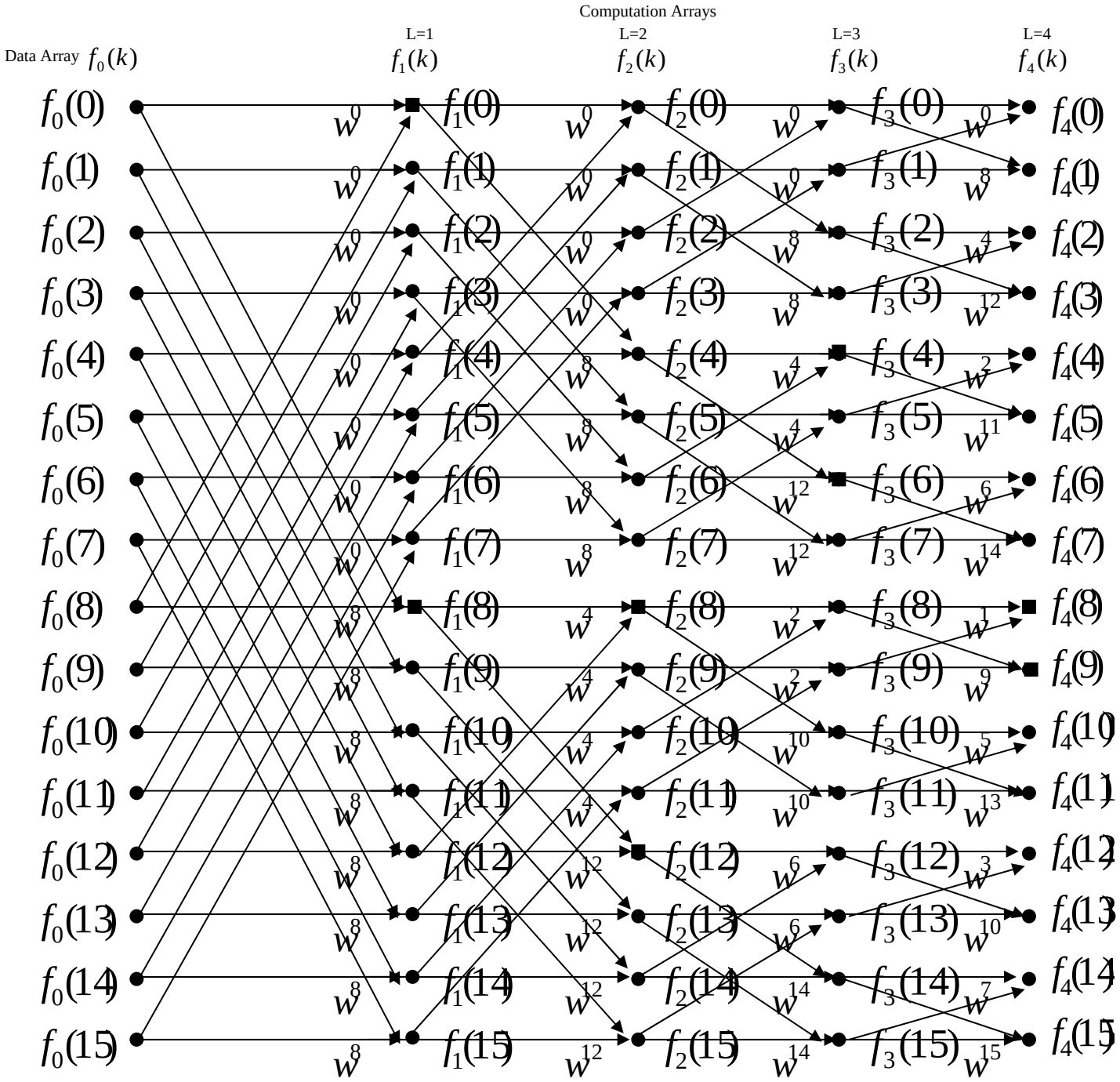


Figure 11.5 Graphical Form of FFT (Eq.11.69) For the case $N = 2^r = 2^4 = 16$.

Dual Node Observation:

Careful observation of Figure 11.5 has revealed that each computed l^{th}-vector (where l=1,2,….r; and $N = 2^r = 2^4 = 16$), we can always find two (dual) nodes which came from the same pair of nodes in the previous vector. For example, $f_1(0)$ and $f_1(8)$ are computed in terms of $f(0)$ and $f(8)$. Similarly, the dual nodes $f_2(8)$ and $f_2(12)$ are computed from the same pair of nodes $f_1(8)$ and $f_1(12)$, etc..

Furthermore, the computation of dual nodes are independent of other nodes (within the l^{th}-vector). Therefore, the computed $f_1(0)$ and $f_1(8)$ will override the original space of $f(0)$ and $f(8)$. Similarly, the computed $f_2(8)$ and $f_2(12)$ will over ride the space occupied by $f_1(8)$ and $f_1(12)$, which in turns, will occupy the original space of $f(8)$ and $f(12)$. Hence, only one complex vector (or 2 real vectors) of length N are needed for the entire FFT process !.

Dual Node Spacing.

Observing Fig 11.5, the following statements can be made:

(a) in the first vector (l=1), the dual nodes $f_1(0)$ and $f_1(8)$ is separated by k=8 (or $\frac{N}{2^l} = \frac{16}{2^1} = 8$) spaces.

(b) In the second vector (l=2), the dual nodes $f_2(8)$ and $f_2(12)$ is separated by k=4 (or $\frac{N}{2^l} = \frac{16}{2^2} = \frac{16}{4}$), etc..

Dual Node Computation:

The operation counts in any dual nodes (of the $l^{th} = 2^{nd}$ vector), such as $f_2(8)$ and $f_2(12)$ can be explained as (see Fig.11.5):

$$f_2(8) = f_1(8) + f_1(12) * W^4 \quad \text{……(11.76)}$$

$$\begin{aligned} f_2(12) &= f_1(8) + f_1(12) * W^{12} \\ &= f_1(8) + f_1(12) * W^8 W^4 \\ &= f_1(8) + f_1(12)\left[e^{-i\frac{2\pi}{(N=16)}} \right]^8 W^4 \\ &= f_1(8) + f_1(12)\left[e^{-i\pi}\right] W^4 \end{aligned}$$

$$f_2(12) = f_1(8) - f_1(12) * W^4 \quad \text{……..…(11.77)}$$

Thus, the dual nodes $f_2(8)$ and $f_2(12)$ computation will require 1 complex multiplication and 2 complex additions (See Eqs.(11.76 and 11.77). The weighting factors for the dual nodes [$f_2(8)$ and $f_2(12)$] are $W^4 (or W^P)$ and $W^{12} (or W^{P+\frac{N}{2}})$, respectively.
Thus, in general:

$$f_l(k) = f_{l-1}(k) + W^P f_{l-1}(k + \frac{N}{2^l}) \text{ .....(11.78)}$$

$$f_l(k + \frac{N}{2^l}) = f_{l-1}(k) - W^P f_{l-1}(k + \frac{N}{2^l}) \text{ .....(11.79)}$$

Skipping certain nodes' computation:

Because the pair of dual nodes "k" and "$k + \frac{N}{2^L}$" are separated by the "distance" ($= \frac{N}{2^L}$), hence, at the L^{th} level, after every $\frac{N}{2^L}$ node computation, then the next $\frac{N}{2^L}$ nodes will be skipped ! (see Fig 11.5)

Determination of W^P

The values of "P" can de determined by the following steps:

Step 1: Express the index k(=0,1,2,…,N-1) in binary form, using r bits. For k=8, and r =4; one obtains

$$k = 8 = 1,0,0,0 = (1)2^{r-1=3} + (0)2^2 + (0)2^1 + (0)2^0$$

Step2: Sliding this binary number "r-L = 4-2 =2" positions to the right, and fill in zeros, the results are:

$1,0,0,0 \rightarrow X, X,1,0 \rightarrow 0,0,1,0$
It is important to realize that the results of Step 2 (0,0,1,0) is equivalent to express an integer $M = \frac{k}{2^{r-L}} = \frac{8}{2^{4-2}} = 2$ in the binary formats. In other words: M=2=(0,0,1,0).

Step3: Reverse the order of the bits, then:

0,0,1,0 becomes 0,1,0,0 = P
Thus, $P = (0)2^3 + (1)2^2 + (0)2^1 + (0)2^0 = 4$
It is "NOT" really necessary to perform Step 3, since the results of Step 2 can be used to compute "P" as following:

$$P = (0)2^0 + (0)2^1 + (1)2^2 + (0)2^3 = 4$$

In conclusion, for $N = 2^r = 2^4 = 16; L = 2; k = 8$ and P=4; the computation of dual nodes from general formulas (See Eqs.11.78, 11.79) gives:

$$f_2(8) = f_1(8) + W^4 f_1(12)$$
$$f_2(12) = f_1(8) - W^4 f_1(12)$$

The above 2 equations are identical to Eqs.(11.76,11.77)!

Computer Implementation to Find Value of "P" (in W^P)

Based on the previous discussions (with the 3-step procedures), to find the value of "P", one only needs a procedure to express an integer $M = \frac{k}{2^{r-L}}$ in binary formats, with "r" bits.
Assuming M (a base 10 number) can be expressed as (assuming r=4bits):

$$M = a_4 a_3 a_2 a_1 = J_1 \quad \text{.....(11.80)}$$

Divide M by 2 (say, $J_2 = \frac{J_1}{2}$), multiply the truncated result by 2 (say, $JJ_2 = J_2 * 2$), and compute the difference between the original number (=M= J_1) & & JJ_2 :

$$IDIFF = J_1 - JJ_2 \left\{ = M - \left(\frac{M}{2} \right)_{Truncated} * 2 \right\} \quad \text{.....(11.81)}$$

If $IDIFF = 0,$ then the bit $a_1 = 0$
If $IDIFF \neq 0,$ then the bit $a_1 = 1$
Once the bit a_1 been determined, the value of J_1 is set to J_2 (or value of J_1 is reduced by a factor of 2; since previous $J_1 = M = a_4 a_3 a_2 a_1$. $J_1 = (a_1)2^0 + (a_2)2^1 + (a_3)2^2 + (a_4)2^3$ and similar process can be used to determine the value of bit a_2, etc…

Example 1: For k=8; $N = 16 = 2^r; r = 4$ bits and $L = 2.$ Find the value of "P" ??

$$M = \frac{k}{2^{r-L}} == \frac{8}{2^{4-2}} = 2 = J_1$$

Determine the bit a_1 : (Index I=1)

Initialize P=0

$$J_2 = \frac{J_1}{2} = \frac{2}{2} = 1$$
$$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 2 - (1)(2) = 0$$

Thus

$a_1 = 0$

$P = P*2 + IDIFF = 0*2 + 0 = 0$ or $[P = P + (a_1)2^{r-I} = 0 + (0)2^3 = 0]$

Determine the bit a_2 [Index I =2]

$J_1 = J_2 = 1$

$J_2 = \frac{J_1}{2} = \frac{1}{2} = 0$

$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 1 - (0*2) = 1$

Thus $a_2 = 1$

$P = P*2 + IDIFF = 0*2 + 1 = 1$ or $[P = P + (a_2)2^{r-I} = 0 + (1)2^2 = 4]$

Determine the bit a_3 [Index I =3]

$J_1 = J_2 = 0$

$J_2 = \frac{J_1}{2} = \frac{0}{2} = 0$

$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 0 - (0*2) = 0$

Thus $a_3 = 0$

$P = P*2 + IDIFF = 1*2 + 0 = 2$ or $[P = P + (a_3)2^{r-I} = 4 + (0)2^1 = 4]$

Determine the bit a_4 [Index I =4=r]

$J_1 = J_2 = 0$

$J_2 = \frac{J_1}{2} = \frac{0}{2} = 0$

$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 0 - (0)*2 = 0$

Thus $a_4 = 0$

$P = P*2 + IDIFF = 2*2 + 0 = 4$ or $[P = P + (a_4)2^{r-I} = 4 + (0)2^0 = 4]$

Remarks:

Although the "intermediate" results might be different, at the end of the do-loop process (computing a_4), both formulas for "P", such as

$P = P*2 + IDIFF; or$ ……(11.82)

$P = P + (a_I)2^{r-I}$; where I=1,2,3…,r ……(11.83)
will eventually give the same final answers for "P".

Example 2: For k=12; $N = 16 = 2^{r=4}$; and $L = 3$. Compute the corresponding value of "P" ??

One has:

$$M = \frac{k}{2^{r-L}} == \frac{12}{2^{4-3}} = 6 = J_1$$

Determine the bit a_1 :(Index I=1)

Initialize P=0

$$J_2 = \frac{J_1}{2} = \frac{6}{2} = 3$$

$$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 6 - (3)(2) = 0$$

Thus

$$a_1 = 0$$

$$P = P * 2 + IDIFF = 0 * 2 + 0 = 0 \text{ or } [P = P + (a_1)2^{r-1} = 0 + (0)2^3 = 0]$$

Determine the bit a_2 [Index I =2]

$$J_1 = J_2 = 3$$

$$J_2 = \frac{J_1}{2} = \frac{3}{2} = 1$$

$$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 3 - (1) * 2 = 1$$

Thus $a_2 = 1$

$$P = P * 2 + IDIFF = 0 * 2 + 1 = 1 \text{ or } [P = P + (a_2)2^{r-2} = 0 + (1)2^2 = 4]$$

Determine the bit a_3 [Index I =3]

$$J_1 = J_2 = 1$$

$$J_2 = \frac{J_1}{2} = \frac{1}{2} = 0$$

$$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 1 - (0) * 2 = 1$$

Thus $a_3 = 1$

$$P = P * 2 + IDIFF = 1 * 2 + 1 = 3 \text{ or } [P = P + (a_3)2^{r-3} = 4 + (1)2^1 = 6]$$

Determine the bit a_4 [Index I =4]

$$J_1 = J_2 = 0$$

$$J_2 = \frac{J_1}{2} = \frac{0}{2} = 0$$

$$IDIFF = J_1 - (JJ_2 = J_2 * 2) = 0 - (0) * 2 = 0$$

Thus $a_4 = 0$

$$P = P * 2 + IDIFF = 3 * 2 + 0 = 6 \text{ or } [P = P + (a_4)2^{r-4} = 6 + (0)2^0 = 6]$$

Remarks:

Although both formulas for “P”, shown in Eqs(11.82,11.83), will yield the same “final” value of “P”. Implementation of Eq.(11.82) will be more computationally efficient !.

UnSrambling the FFT.

For the case $N = 16 = 2^{r=4}$ (see Figure 11.5), the final ‘bit-reversing’ operation for FFT is shown in Fig. 11.6.

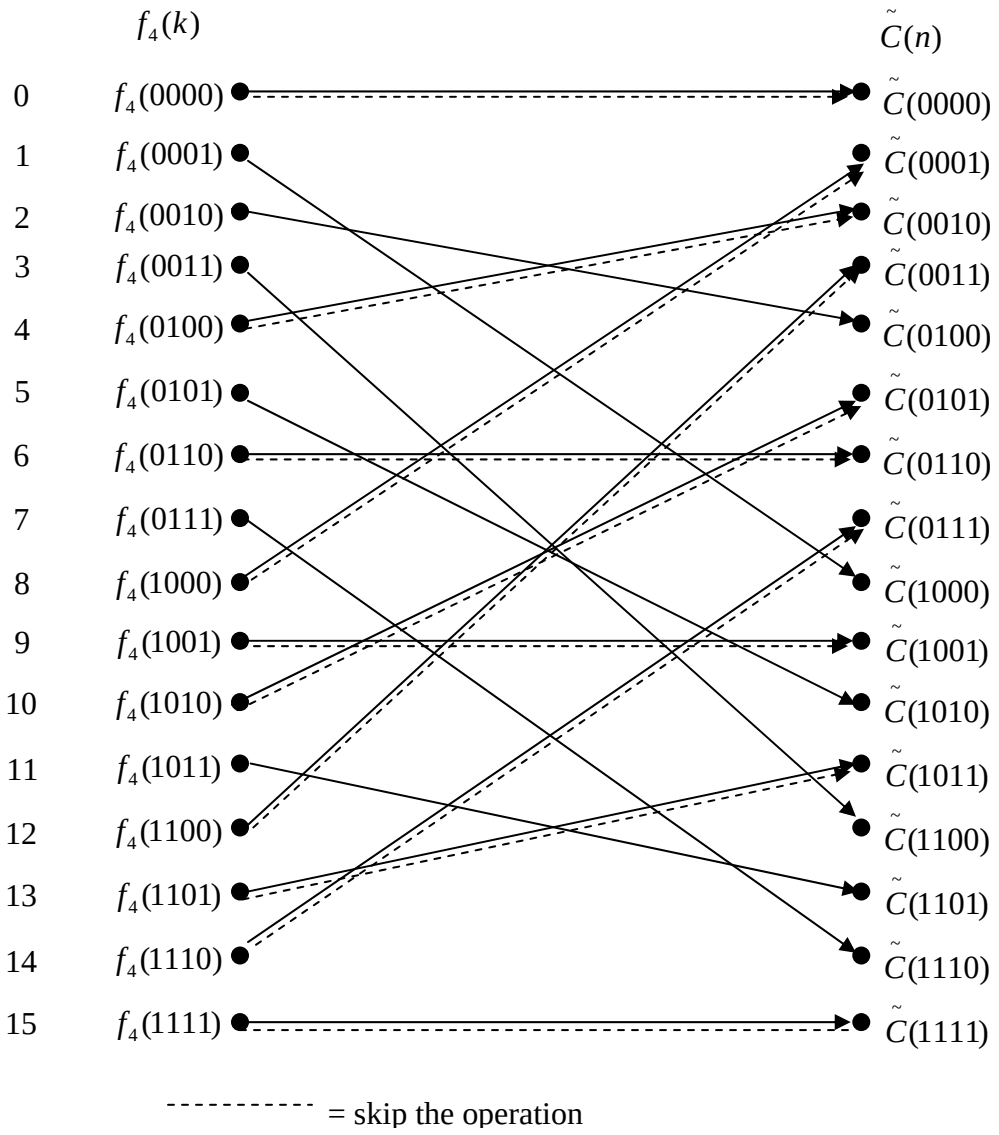


Figure 11.6: Final “Bit-Reversing” for FFT (with $N = 2^r = 2^4 = 16$)

For do-loop index k=0=(0,0,0,0) $\Rightarrow$ i=(0,0,0,0)=0

If (*i.GT.k*)*Then*

$T = f_4(k)$

$f_4(k) = f_4(i)$

$f_4(i) = T$

Endif

Hence, $f_4(0) = f_4(0)$; no swapping.

For k=1=(0,0,0,1) $\Rightarrow$ i=(1,0,0,0)=bit-reversion=8

If (*i.GT.k*)*Then*

$T = f_4(1)$

$f_4(1) = f_4(8)$

$f_4(8) = T$

Endif

Hence, $f_4(1) = f_4(8)$; are swapped.

. For k=2=(0,0,1,0) $\Rightarrow$ i=(0,1,0,0)=4

Hence, $f_4(2) = f_4(4)$; are swapped.

. For k=3=(0,0,1,1) $\Rightarrow$ i=(1,1,0,0)=12

Hence, $f_4(3) = f_4(12)$; are swapped.

. For k=4=(0,1,0,0) $\Rightarrow$ i=(0,0,1,0)=2

In this case, since "i" is not greater than "k".

Hence, no swapping, since $f_4(k = 2)$ and $f_4(i = 4)$; had already been swapped earlier !.

.

.

. etc…

Computer Implementation of FFT (for case $N = 2^r$).

The pair of dual nodes computation are given by Eqs.(11.78,11.79). To avoid "complex number" operations, Eq.(11.78) can be computed based on "real number" operations, as following:

$$\left\{f_L^R(k) + if_L^I(k)\right\} = \left\{f_{L-1}^R(k) + if_{L-1}^I(k)\right\} + \left\{W^{P,R} + iW^{P,I}\right\} * \left\{f_{L-1}^R(k + \frac{N}{2^L}) + if_{L-1}^I(k + \frac{N}{2^L})\right\} \quad \text{.....(11.84)}$$

In Eq. (11.84), the superscripts R and I denote Real and Imaginary components, respectively.

Multiplying the last 2 complex numbers, one obtains:

$$\left\{f_L^R(k)+if_L^I(k)\right\}=\left\{f_{L-1}^R(k)+if_{L-1}^I(k)\right\}$$
$$+\left\{W^{P,R}*f_{L-1}^R(k+\frac{N}{2^L})-W^{P,I}*f_{L-1}^I(k+\frac{N}{2^L})\right\}$$
$$+i\left\{W^{P,R}*f_{L-1}^I(k+\frac{N}{2^L})+W^{P,I}*f_{L-1}^R(k+\frac{N}{2^L})\right\}\ldots\ldots\ldots\ldots\ldots\ldots(11.85)$$

Equating the Real (and then, Imaginary) components on the Left-Hand-Side (LHS), and the Right-Hand-Side (RHS) of Eq.(11.85), one obtains:

$$\left\{f_L^R(k)\right\}=\left\{f_{L-1}^R(k)\right\}+\left\{W^{P,R}*f_{L-1}^R(k+\frac{N}{2^L})-W^{P,I}*f_{L-1}^I(k+\frac{N}{2^L})\right\}\ldots\ldots\ldots\ldots\ldots\ldots(11.86A)$$

$$\left\{f_L^I(k)\right\}=\left\{f_{L-1}^I(k)\right\}++\left\{W^{P,R}*f_{L-1}^I(k+\frac{N}{2^L})+W^{P,I}*f_{L-1}^R(k+\frac{N}{2^L})\right\}\ldots\ldots\ldots\ldots\ldots\ldots(11.86B)$$

Recalled Eq. (11.64):

$$W=e^{-i\frac{2\pi}{N}}$$

Hence:

$$W^P=\left(e^{-i\frac{2\pi}{N}}\right)^P=e^{-i\frac{2\pi P}{N}}=e^{-i\theta}=\cos(\theta)-i\sin(\theta)\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.87)$$

where:

$$\theta=\frac{2\pi P}{N}=\frac{6.28P}{N}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.88)$$

Thus:

$$W^{P,R}=\cos(\theta)\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.89A)$$

$$W^{P,I}=-\sin(\theta)\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11.89B)$$

Substituting Eqs.(11.89A,11.89B) into Eqs.(11.86A,11.86B), one gets:

$$\left\{f_L^R(k)\right\}=\left\{f_{L-1}^R(k)\right\}+\left\{\cos(\theta)*f_{L-1}^R(k+\frac{N}{2^L})+\sin(\theta)*f_{L-1}^I(k+\frac{N}{2^L})\right\}\ldots\ldots\ldots\ldots\ldots(11.90A)$$

$$\left\{f_L^I(k)\right\}=\left\{f_{L-1}^I(k)\right\}+\left\{\cos(\theta)*f_{L-1}^I(k+\frac{N}{2^L})-\sin(\theta)*f_{L-1}^R(k+\frac{N}{2^L})\right\}\ldots\ldots\ldots\ldots\ldots(11.90B)$$

Similarly, the single (complex number) Eq.11.79 can be expressed as 2 equivalent (real number) Eqs. Like Eqs. (11.90A,11.90B) !

```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine fft(freal,fimag,n,igama)
      implicit real*8(a-h,o-z)
      dimension freal(*),fimag(*)
c......purpose: fft algorithms (for general base 2)
c......programmed by: Prof. Duc T. Nguyen (DNguyen@odu.edu)
c......original date: 07-10-2008
c......freal(n)     = real number of N complex data points
c......fimag(n)     = imaginary number of N complex data points
c......n            = number of complex data points = 2**igama
c......example  n   = 2**4 = 16; hence igama = 4
c......remarks:      Both DFT & FFT did give IDENTICAL results !
c
c        write(6,*) 'inside routine fft: echo input freal,fimag = '
       do 24 i=1,n
c        write(6,*) i, freal(i), fimag(i)
 24     continue
      k=0
c        write(6,*) 'n, igama = ',n,igama
      do 1 L=1,igama
      n2=n/2**L
      igaminusL=igama-L
 123   do 2 i=1,n2
      m=k/2**igaminusL
      call bitreverse(m,igama,ip)
c        write(6,*) 'L, i, m, ip = ',L,i,m,ip
      theta=6.283185*ip/n
      c=cos(theta)
      s=sin(theta)
c        write(6,*) 'theta,c,s = ',theta,c,s
      k1=k+1
      nodedual=k1+n2
c        write(6,*) 'dual nodes = k1, nodedual = ',k1,nodedual
c......applying Duc's Eqs.(11.90A, 11.90B)
      partreal=c*freal(nodedual)+s*fimag(nodedual)
      partimag=c*fimag(nodedual)-s*freal(nodedual)
      freal(nodedual)=freal(k1)-partreal
      fimag(nodedual)=fimag(k1)-partimag
      freal(k1)=freal(k1)+partreal
      fimag(k1)=fimag(k1)+partimag
      k=k+1
c        write(6,*) 'partreal, partimag = ',partreal,partimag
 2     continue
c        write(6,*) 'computed array at level L = ',L
       do 26 kk=1,n
c        write(6,*) 'freal(kk),fimag(kk) = ',freal(kk),fimag(kk)
```

```
 26     continue
      k=k+n2
      if (k .lt. n) go to 123
      k=0
 1    continue
c
c       write(6,*) 'before unscramble FFT: i,freal,fimag ='
       do 22 i=1,n
c       write(6,*) i, freal(i), fimag(i)
 22     continue
c......unscramble results of FFT
      call unscramble(freal,fimag,n,igama)
c......output FFT solution
c       write(6,*) 'after unscramble FFT: i,freal,fimag ='
       sumreal=0.d0
       sumimag=0.d0
       do 42 i=1,n
c       write(6,*) i, freal(i), fimag(i)
       sumreal=sumreal+dabs( freal(i) )
       sumimag=sumimag+dabs( fimag(i) )
 42     continue
      write(6,*) 'FFT: sumreal,sumimag = ',sumreal,sumimag
 999   return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine bitreverse(m,igam,ip)
      ip=0
      j1=m
c
      do 2 i=1,igam
      j2=j1/2
      idiff=j1-j2*2
      ip=ip*2 + idiff
      j1=j2
 2    continue
c     write(6,*) 'p, or ii = ',ip
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine unscramble(freal,fimag,n,igama)
      implicit real*8(a-h,o-z)
      dimension freal(*),fimag(*)
c
      do 2 k=1,n
      m=k-1
      call bitreverse(m,igama,ii)
```

```
      i=ii+1
      if (i .le. k) go to 2
      temporeal=freal(k)
      tempoimag=fimag(k)
      freal(k)=freal(i)
      fimag(k)=fimag(i)
      freal(i)=temporeal
      fimag(i)=tempoimag
 2    continue
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

11.8 Brief Review About "MPI-parallel" FORTRAN-90 Programming.

```
c  ------------------------------------------------------------------------------------------------

c  Message Passing Interface (MPI) parallel application codes can be
c  implemented in either FORTRAN, or C++ language, under UNIX, LINUX,
c  or WINDOWS environments. The syntax for "parallel" MPI/FORTRAN-90 are
c  essentially identical to the ones used in "serial" FORTRAN-90, with
c  few exceptions for "specific parallel computational purposes".

c  Regardless the computer language adopted by the users (such as C, or
C++,
c  or FORTRAN-77, or FORTRAN-90, or BASIC etc...), one only needs to be
c  familiar with the syntax for "IF" statements, "DO" loop, "DIMENSION"
c  statements (for handling 1-D, and/or 2-D, and/or 3-D integer/real
arrays),
c  input/output, and usage of "subroutines".

c  The following listed MPI/FORTRAN-90 demonstrated code can be
conveniently
c  used to understand the "syntax" for writing any general application
codes.


c============================================================!000
c234567890123456789012345678901234567890123456789012345678901234567!001
c Purposes:  Reviewing some basic FORTRAN_90 syntax, and MPI_FORTRAN    !002
c Author(s): Prof. Duc Thai NGUYEN (757-683-3761; DNguyen@odu.edu)      !003
c Date:      June 10, 2008                                              !004
c Stored At: cd ~/cee/*odu*class*/teach_fortran90_mpi.f                 !005
c                                                                       !006
      implicit real*8(a-h,o-z)                                          !007
      include 'mpif.h'                                                  !008
      character*80 title                                                !009
      parameter (num=10)                                                !010
      parameter (master=0)                                              !011
      parameter (from_master=1)                                         !012
      parameter (from_worker=2)                                         !013
      dimension a(num),b(num)                                           !014
      allocatable:: ia(:),a11(:,:),a22(:,:)                             !015
c-----------------------------------------------------------------------------------------------!016
      call MPI_INIT(ierr)                                               !017
      call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)                      !018
      call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)                      !019
       if (me .eq. 0) then                                              !019.1
```

```
       write(6,*) '                              '                                !019.2
       write(6,*) '================================='                              !019.3
       write(6,*) 'Prof. Duc T. Nguyen; June 17, 2008'                            !019.4
       write(6,*) '================================='                              !019.5
       write(6,*) '                              '                                !019.6
      endif                                                                       !019.7
c-------------------------------------------------------------------------        !020
c     call MPI_BARRIER(MPI_COMM_WORLD, ierr)                                      !021
c     call MPI_SEND(num,1,MPI_INTEGER,i_destination,1,MPI_COMM_WORLD,!
022
c    $ierr)                                                                       !023
c     call MPI_RECV(num,1,MPI_INTEGER,master,mtype,MPI_COMM_WORLD, !024
c    $status,ierr)                                                               !025
c-------------------------------------------------------------------------       !026
     idum=0                                                                       !027
     sum=0.d0                                                                     !028
     do 1 i=1,num,1                                                               !029
      a(i)=drand(idum)                                                            !030
      sum=sum+a(i)                                                                !031
      if (i .le. 10) then                                                         !032
      write(6,*) 'i,a(i) = ',i,a(i)                                               !033
      elseif (i .gt. 10) then                                                     !034
      write(6,*) 'skip printing too many random numbers !'                        !035
      endif                                                                       !036
 1    continue                                                                    !037
c                                                                                 !038
      open (unit=7, file='K.INFO', status='old', form='formatted')                !039
c      open (unit=6, file='out1', status='old', form='formatted')                 !040
      read(7,115) title                                                           !041
 115   format(a60)                                                                !042
      write(6,115) title                                                          !043
c                                                                                 !044
      memory_need=2*num                                                           !045
      allocate ( ia(memory_need), a11(memory_need,memory_need),                   !046
     $           a22(num,num) )                                                   !047
     do 2 i=1,memory_need,1                                                       !048
      ia(i)=i                                                                     !049
 2    continue                                                                    !050
      deallocate(a11,a22)                                                         !051
     call dummy1(num,memory_need,a,sum_real)                                      !052
     write(6,*) ' sum_real= ', sum_real                                           !053
c-------------------------------------------------------------------------       !054
      num_workers=np-1                                                            !055
      bigest_local=0.d0                                                           !056
c.....each processor (master and workers) will:                                   !057
c.....generate its own portions of random (real) numbers                          !058
```

```
c......then, it will find its own local maximum number                    !059
      do 11 i=me+1, num, np                                               !060
       b(i)=drand(idum)                                                   !061
       if ( b(i) .gt. bigest_local ) bigest_local=b(i)                    !062
        write(6,*) 'processor id# ',me, 'i,b(i) = ',i,b(i)                !062.1
        write(6,*) 'processor id# ',me, 'bigest_local = ', bigest_local   !062.2
 11   continue                                                            !063
c                                                                         !064
c..... each worker will send its own local maximum to the master          !065
      if (me .gt. 0) then                                                 !066
       mtype=from_worker                                                  !067
       call MPI_SEND(bigest_local,1,MPI_DOUBLE_PRECISION,master,mtype     !068
     $,MPI_COMM_WORLD,ierr)                                               !069
       write(6,*) 'sent by worker # ',me, ' bigest_local= ',bigest_local  !069.1
c..... the master processor will receive local maximum                    !070
c..... (from each worker)                                                 !071
c..... and then, comparing amongst all local max to find/print            !072
c..... global max                                                         !073
      elseif (me .eq. 0) then                                             !074
      bigest_global=bigest_local                                          !075
      mtype=from_worker                                                   !076
       write(6,*) 'processor id # ',me, ' bigest_local= ',bigest_local    !076.1
      do 60 i=1,num_workers,1                                             !077
       isource=i                                                          !078
       call MPI_RECV(bigest_local,1,MPI_DOUBLE_PRECISION,isource,mtype,   !079
     $MPI_COMM_WORLD,status,ierr)                                         !080
       if (bigest_local .gt. bigest_global) bigest_global=bigest_local    !081
 60   continue                                                            !082
       write(6,*) 'amongst local max, the global max is ',bigest_global   !083
      endif                                                               !084
c                                                                         !085
      write(6,*) 'processor id# ',me, 'out of ',np, ' is alive'           !086
      call MPI_FINALIZE(ierr)                                             !087
c-----------------------------------------------------------------------  !088
      stop                                                                !089
      end                                                                 !090
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  !091
      subroutine dummy1(num,memory_need,a,sum_real)                       !092
      implicit real*8(a-h,o-z)                                            !093
      dimension a(*)                                                      !094
      sum_real=0.d0                                                       !095
      do 1 i=1,num,1                                                      !096
       sum_real=sum_real+a(i)                                             !097
 1    continue                                                            !098
      return                                                              !099
```

```
      end                                                        !100
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                                   !101

c  Lines #001-006:
c  In FORTRAN, if a character "c" is typed in column1, then the line will be
c  treated like a "comment" statement.
c
c  Line #007:
c  In FORTRAN, all "executable" statements should be typed between column
c  numbers 7 through 72. Any "real" array should be named with the first
c  character as a, b, c, ..., h, and o, p, q, ..., z. Any "integer" array
c  should be named with the first character as i,j,k,l,m,n.
c  This statement implies that each real number will need 8 bytes to store
c  (in double precision). Similarly, a statement:
c  implicit real*4(a-h,o-z) implies that each real number will need 4 bytes
c  to store (in single precision).

c  Line #008:
c  This include statement "MUST" always be followed the implicit statement
c  for any MPI/FORTRAN application code

c  Line #009 (also see lines # 041-043):
c  This statement is necessary only if the user want to read (or write)
c  a title heading, with upto 80 characters (also see lines # 041-043)

c  Lines # 010-013:
c  Numerical values of certain variables can be defined/given/assigned by
c  the parameter statements.

c  Line # 014:
c  Maximum dimension (or size) for certain arrays are defined by the
c  "dimension" statement. Note that the value of "num" must already be
c  earlier defined (through the parameter statements)

c  Line # 015 (also see lines # 045-047):
c  This is one of the "very useful" features in FORTRAN-90, for which
c  the users can declare some arrays for "dynamic storage allocation"
purposes.
c  The actual, exact "dimension" for these arrays do NOT have to be
declared
c  in the begining (such as arrays defined on line # 014). These "exact"
c  "dimension" needed can be declared "later on", whenever the user knows
c  exactly how much memory storage one needs for these arrays (also see
c  lines # 045-047)
```

```
c  Lines # 017-019:
c  These 3 "special" MPI/FORTRAN statements "MUST" be defined in any MPI
c  application codes (and should be inserted right after dimension
statements).
c  The variable "np" on line # 019 represents (Number of Processors". Thus,
c  if 3 processors are used, then np will be assigned the value 3 by the
system.
c  The variable "me" on line # 018 will have the values (assined by the
computer
c  system) 0,1,2, ..., np-1. This variable "me" will play a CRUCIAL role in
c  any MPI application codes.
c
c  It should be emphasized here that all processor ID # = 0,1,2, ..., np
c  will execute the same application code. However, depending on the
algorithms,
c  the user will have direct control of deciding "WHICH processor ID" will
c  execute on "WHAT portions of the code" etc..., through the usage of
variable
c  "me" (also refer to lines # 060-063)

c  Lines # 019.1-019.7:
c  Only the "master" processor (me=0) will execute this block of
statements,
c  which basically print out some output message [any desired output
message
c  can be placed inside (open/close) single quotes].

c  Lines # 020-026:
c  There are about 10-20 "special, parallel" MPI constructs that are very
c  commonly used in any application codes. Amongst these MPI statements,
c  however, BARRIER, SEND and RECV are probably the most important ones to
c  be used. Basically, BARRIER statement will make sure that all processors
c  have to arrive at this statement, before they can proceed to execute
c  subsequent statements of the application code. SEND statement will send
c  a message (such as an integer/real variable, or integer/real arrays)
from
c  one processor to another (specified) processor. Important argument lists
c  are explained as following:
c  1-st Argument = name of a variable (or array)
c  2-nd Argument = the "dimension" associated with this variable (or array)
c  3-rd Argument = the variable (or array) must be defined as INTEGER, or
c                  REAL (or DOUBLE PRECISION)
c  4-th Argument = send to WHICH processor ??
c  5-th Argument = message type #
c  6-th Argument = user does NOT need to know !
c  7-th Argument = user does NOT need to know !
```

```
c
c  RECV statement can be used for RECEIVING a message. Important argument
c  lists are explained as following:
c  1-st Argument = name of a variable (or array)
c  2-nd Argument = the "dimension" associated with this variable (or array)
c  3-rd Argument = the variable (or array) must be defined as INTEGER, or
c                  REAL (or DOUBLE PRECISION)
c  4-th Argument = receive from WHICH processor ??
c  5-th Argument = message type #
c  6-th Argument = user does NOT need to know !
c  7-th Argument = user does NOT need to know !
c  8-th Argument = user does NOT need to know !
c
c  The user does NOT need to know about the 2 argument lists of the MPI
c  BARRIER statement.

c  Lines # 027-037:
c  The purpose of this block of FORTRAN statements are:
c  to show the "syntax" of "do" loop (see line # 029), the integer index
"i"
c  will have the values from 1 through num (=10), with the increment of 1.
c  Lines # 027, and # 030 show how to use "built-in" library function to
c  generate a real random number (between 0.00 and 1.00).
c  to show the "syntax" of "IF" statement (see lines # 032, # 034, and #
036)
c  to show the "syntax" of writing/printing some intermediate output
variables.

c  Lines # 038-044:
c  Input (read), and output (write) data files can be used through the
"open"
c  statements on line # 039 and line # 040, respectively.

c  Lines # 045-050:
c  At this moment, the user knows "exactly" how much memory space that
he/she
c  needs to allocate (or assign) to INTEGER array ia(-), REAL arrays
a11(-,-),
c  and a22(-,-). Thus, request to allocate memory space was done on line #
046-
c  # 047.

c  Line # 051:
c  Assuming that at this stage the user does NOT need the arrays a11(-,-),
and
c  a22(-,-) any more, hence he/she can request to DELETE all memory spaces
```

```
c  allocated to these 2 arrays, through the DEALLOCATE statement.

c  Lines # 052-054:
c  A subroutine dummy1 is called by the main program, in order to perform a
c  certain task. In this example, the first 3 argument lists are "INPUT"
c  to this subroutine, and the 4-th argument list ( = sum_real) provide the

c  "OUTPUT" from this subroutine.

c  Line # 055:
c  Since in this example np = Number of Processors = 3, hence processor
ID#0
c  will be the "master" processor, and processor ID# 1, #2 are considered
c  as "worker" processors.

c  Lines # 056-063:
c  Each processor will generate its own random numbers, and also
compute/print
c  its own (local) maximum number (amongst its own random numbers). The
most
c  important statement for this block is shown on line # 060 (please pay
c  attention to variable "me").
c  For the "master" processor (such as me=0), it will generate random
numbers
c  coresponding to the do-loop integer index i = 1, 4, 7, and 10 (the
increment
c  for index i is np = 3).
c  For the "worker" processor (such as me=1), it will generate random
numbers
c  coresponding to the do-loop integer index i = 2, 5 and 8.
c  For the "worker" processor (such as me=2), it will generate random
numbers
c  coresponding to the do-loop integer index i = 3, 6 and 9.
c  Also, all 3 processors (such as the "master" processor me=0, and "slave"
c  processors me=1, 2) will compute its own local maximum value (stored in
c  variable name bigest_local)
c
c  Lines 064-069.1:
c  Upon completion its task, each "slave" worker will send its own local
maximum
c  to the "master" processor.
c
c  Lines 070-085
c  The "master" will receive all "slaves"' local maximum values, and it
will
c  compare all these local maximum (including the "master's" own local
```

```
maximum),
c  in order to identify , and print the global maximum (stored in variable
name
c  bigest_global).
c
c  Line 086
c  All (master and slave) processors will print out a message before
exiting.
c
c  Lines 087-091
c  This MPI_FINALIZE(ierr) "must" be placed before the program stops
c
c  Lines 092-101
c  This subroutine just computes some dummy works, such as calculating
c  the summation of a given 1-D real array
```

11.9 Parallel MPI/FORTRAN FFT Base-2 Algorithms.

Observing Figure 11.5 (FFT algorithms with $N = 2^r = 2^4 = 16$) and also referring to the 2^{nd} (or inner) do-loop index I ($= 1 \Rightarrow N_2$; where initial value for $N_2 = 8$), presented in the serial FFT code, the following major changes are necessary for converting the earlier serial code into parallel code (assuming NP=2 processors, with processor ME=0 and ME=1 are available). The entire parallel MPI/FFT code is listed in Table 11.10.

(a) Computation of "dual node" pair of an array, such as
$f_1(I = 1), f_1(I + N_2 = 9)$
$f_2(I = 2), f_1(I + N_2 = 10)$

$f_1(I = N_2 = 8), f_1(I + N_2 = 16)$
are completely independent from each other. Since FORTRAN does "not enjoy" with zero subscript, the above $f_1(1 \to 16)$ are correspondent to $f_1(0 \to 15)$, respectively. The computation of the dual pair $f_1(0)$ and $f_1(8)$ in Figure 11.5 will only require the terms f(0) and f(8) from previous array. Similarly, computation of the dual pair $f_1(7)$ and $f_1(15)$ will only require the terms f(7) & f(15) from previous array.

(b) Based on the above observation, the inner serial do-loop:
Do 2 i=1, N_2,1 should be replaced by the following parallel do-loop:
Do 2 i = ME+1, $N_2 = 8, NP = 2$
Thus, processor ME=0 will be assigned to compute
$f_1(1)$ & $f_1(9)$, $f_1(3)$ & $f_1(11)$, $f_1(5)$ & $f_1(13)$ and $f_1(7)$ & $f_1(15)$

while at the same time, processor ME=1 will try to compute
$f_1(2)$ & $f_1(10)$, $f_1(4)$ & $f_1(12)$, $f_1(6)$ & $f_1(14)$ and $f_1(8)$ & $f_1(16)$

(c) The "local" variable ICOUNT and "local" array index(icount), see MPI source code listing, are used to record which terms of the computed array $f_1(1 \to 16)$ were computed by which processors. These local arrays are required, since we do want to minimize processors' communication time by packing more data for each MPI_SEND (or MPI_BROADCAST) statement.

(d) The "Local" variable <u>increase</u> (initiated to zero) will help the parallel FFT algorithm to implement the patterns of computing N_2 terms, then skipping next N_2 terms, etc..

(e) Subroutine unscramble can also be parallelized, as indicated in the parallel MPI source code listing. However, due to unsignificant computational efforts occurred in a single (not nested) do-loop, serial coding for this subroutine is recommended.

(f) If the incore memory is limited, and is a concern for the user, then the entire do 28 loop (including the 2 real arrays tempo1real(-) and tempo1imag(-)) can be eliminated. Also, the 2 call MPI_BROADCAST statements should be placed right before the "2 continue" statement (or inside the loop do 2 i=me+1, N_2,NP). The trade-off in this case, ofcourse, will be a substantial increase in processors' communication cost!

(g) The suggested parallel FFT strategies are mainly designed for "educational" purpose, and might not be practical for the following reasons:

1. Due to the nature of FFT algorithms, parallel processing can only be done at the innermost (or 2^{nd}) do-loop, rather than at the preferable outermost (or1^{st}) do-loop!
2. Even for fairly large data points (say N is large), there are not-much computational efforts inside the "inner" do-loop.

(h) In the DFT (see Eq.11.65, or 11.67), matrix times vector operations are needed, which also requires two nested do-loops (see Table 11.2). It is a well-known fact that for "matrix*vector" operations, efficient parallel processing can be done at the "outermost" do loop while "unrolling strategies" can be exploited at the "innermost" do-loop [Refs. 5 – 6]. Despite of the above favorable features, DFT is not matched for FFT algorithms (recalled Eq.11.75, and for even small-medium size $N = 2^r = 2^{11} = 2048$, FFT offers 372.36 times less # operations as compared to DFT formula !)

Table 11.10: MPI/FORTRAN "FFT" Source Code

```
c
      implicit real*8(a-h,o-z)
      include 'mpif.h'
c.....purpose:  mpi/parallel fft algorithms & software
      dimension freal(1000000), fimag(1000000)
      dimension tempo1real(1000000),tempo1imag(1000000)
     $,         index(1000000)
      open(unit=5,file='fft.dat',status='old',form='formatted')
       call MPI_INIT(ierr)
       call MPI_COMM_RANK(MPI_COMM_WORLD,me,ierr)
       call MPI_COMM_SIZE(MPI_COMM_WORLD,np,ierr)
       write(6,*) 'processor ME = ',me, ' is alive !'
       if (me .eq. 0) then
      read(5,*) iautodata, n, igama, method
      write(6,*) 'iautodata,n,igama, method = '
      write(6,*)  iautodata,n,igama, method
      if (iautodata .eq. 1) then
       do 1 i=1,n
       freal(i)=dfloat(i)
       fimag(i)=0.d0
 1      continue
      elseif (iautodata .eq. 0) then
       read(5,*) ( freal(i),i=1,n )
       read(5,*) ( fimag(i),i=1,n )
      endif
       endif
c
c       write(6,*) 'input data for FFT: i,freal,fimag ='
c       do 22 i=1,n
c       write(6,*) i, freal(i), fimag(i)
c22     continue
c
       call MPI_BCAST(iautodata,1,MPI_INTEGER,0,
     $  MPI_COMM_WORLD,ierr)
       call MPI_BCAST(n,1,MPI_INTEGER,0,
     $  MPI_COMM_WORLD,ierr)
       call MPI_BCAST(igama,1,MPI_INTEGER,0,
     $  MPI_COMM_WORLD,ierr)
       call MPI_BCAST(method,1,MPI_INTEGER,0,
     $  MPI_COMM_WORLD,ierr)
       call MPI_BCAST(freal,n,MPI_DOUBLE_PRECISION,0,
     $  MPI_COMM_WORLD,ierr)
       call MPI_BCAST(fimag,n,MPI_DOUBLE_PRECISION,0,
     $  MPI_COMM_WORLD,ierr)
       if (method .eq. 1) then
```

```
      call fft(freal,fimag,n,igama,me,np,
     $ tempo1real,tempo1imag,index)
       elseif (method .eq. 2) then
      call dft(freal,fimag,n,igama)
       endif
c
c       write(6,*) 'output for FFT: i,freal,fimag ='
c       do 23 i=1,n
c       write(6,*) i, freal(i), fimag(i)
c23     continue
c
999    call MPI_FINALIZE(ierr)
      stop
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine dft(freal,fimag,nn,igama)
      implicit real*8(a-h,o-z)
      dimension freal(*), fimag(*)
c
      pai=3.14159d0
      w0=2.d0*pai/dfloat(nn)
      sumreal=0.d0
      sumimag=0.d0
       do 1 n=1,nn
       cnreal=0.d0
       cnimag=0.d0
         do 2 k=1,nn
         angle=(k-1)*w0*(n-1)
         c=cos(angle)
         s=sin(angle)
         cnreal=cnreal+freal(k)*c+fimag(k)*s
         cnimag=cnimag+fimag(k)*c-freal(k)*s
 2       continue
      write(6,*) 'dft results: n,freal,fimag = '
      write(6,*) n, cnreal, cnimag
      sumreal=sumreal+dabs(cnreal)
      sumimag=sumimag+dabs(cnimag)
 1    continue
      write(6,*) 'DFT: sumreal,sumimag = ',sumreal,sumimag
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine fft(freal,fimag,n,igama,me,np,
     $ tempo1real,tempo1imag,index)
      implicit real*8(a-h,o-z)
      include 'mpif.h'
```

```
      dimension freal(*),fimag(*)
     $,tempo1real(*),tempo1imag(*),index(*)
c.....purpose: fft algorithms (for general base 2)
c.....programmed by: Prof. Duc T. Nguyen (DNguyen@odu.edu)
c.....original date: 07-10-2008
c.....freal(n)     = real number of N complex data points
c.....fimag(n)     = imaginary number of N complex data points
c.....n            = number of complex data points = 2**igama
c.....example  n   = 2**4 = 16; hence igama = 4
c.....remarks:       Both DFT & FFT did give IDENTICAL results !
c
c-----------------------
       ntoddcount = 0      ! temp added by Todd
       if (me .eq. 0) then
      write(6,*) '                              '
      write(6,*) '================================'
      write(6,*) ' Prof. Nguyen Version Date: 07-29-2008'
      write(6,*) '================================'
      write(6,*) '                              '
!         write(6,*) 'inside routine fft: echo input freal,fimag = '
!         do 24 i=1,n
!         write(6,*) i, freal(i), fimag(i)
!  24     continue
       endif
c        call MPI_BARRIER(MPI_COMM_WORLD,ierr)
c-----------------------
      k=0
c       write(6,*) 'n, igama = ',n,igama
      igamatodd = 5        ! by Todd
      do 1 L=1,igamatodd
!       if (me.eq.0) write(6,*) '*************************'
!       if (me.eq.0) write(6,*) 'L=',L
!       if (me.eq.0) write(6,*) '*************************'
       write(6,*) me,'L=',L

      n2=n/2**L
      igaminusL=igama-L
       icount=0           ! parallel fft
       increase=0         ! parallel fft
       tempo1real(1:n)=0.d0
       tempo1imag(1:n)=0.d0
!         do 456 kk=1,n
!         tempo1real(kk)=0.d0
!  456    tempo1imag(kk)=0.d0
c123   do 2 i=1,n2
c        write(6,*) me,me+1,n2,np
```

```
 123   do 2 i=ME+1,n2,NP
!        write(6,*) 'processor ME = ',me, ' is alive !'
       k=i-1 + increase    ! parallel fft
c        write(6,*) me,'By Todd',k
      m=k/2**igaminusL
      call bitreverse(m,igama,ip)
c       write(6,*) 'L, i, m, ip = ',L,i,m,ip
      theta=6.283185*ip/n
      c=cos(theta)
      s=sin(theta)
c       write(6,*) 'theta,c,s = ',theta,c,s
       k1=k+1
       icount=icount+1     ! parallel fft
       index(icount)=k1    ! parallel fft
      nodedual=k1+n2
!        write(6,*) 'dual nodes = k1, nodedual = ',k1,nodedual
        icount=icount+1    ! parallel fft
        index(icount)=nodedual   ! parallel fft
c......applying Duc's Eqs.(11.90A, 11.90B)
      partreal=c*freal(nodedual)+s*fimag(nodedual)
      partimag=c*fimag(nodedual)-s*freal(nodedual)
      freal(nodedual)=freal(k1)-partreal
      fimag(nodedual)=fimag(k1)-partimag
      freal(k1)=freal(k1)+partreal
      fimag(k1)=fimag(k1)+partimag
      k=k+1
c       write(6,*) 'partreal, partimag = ',partreal,partimag
 2     continue
c......broadcast and update the computed array to all other processors
      do 28 jj=1,icount
       kk=index(jj)
       tempo1real(kk)=freal(kk)
       tempo1imag(kk)=fimag(kk)
 28    continue
!         call MPI_BARRIER(MPI_COMM_WORLD,ierr)
c       call MPI_REDUCE(tempo1real,freal,n,MPI_DOUBLE_PRECISION,
c    $  MPI_SUM,0,MPI_COMM_WORLD,ierr)
c       call MPI_BCAST(freal,n,MPI_DOUBLE_PRECISION,0,
c    $  MPI_COMM_WORLD,ierr)
c       call MPI_REDUCE(tempo1imag,fimag,n,MPI_DOUBLE_PRECISION,
c    $  MPI_SUM,0,MPI_COMM_WORLD,ierr)
c       call MPI_BCAST(fimag,n,MPI_DOUBLE_PRECISION,0,
c    $  MPI_COMM_WORLD,ierr)
c        write(6,*) me,'before mpi_allreduce',L

       call MPI_ALLREDUCE(tempo1real,freal,n,MPI_DOUBLE_PRECISION,
```

```
   $  MPI_SUM,MPI_COMM_WORLD,ierr)
c    $  MPI_SUM,COMM,ierr)
      call MPI_ALLREDUCE(tempo1imag,fimag,n,MPI_DOUBLE_PRECISION,
   $  MPI_SUM,MPI_COMM_WORLD,ierr)
c    $  MPI_SUM,COMM,ierr)
c      k=k+n2                ! parallel fft
c      if (k .lt. n) go to 123 ! parallel fft
       ntoddcount = ntoddcount+1       ! added by todd
c        if (L.eq.igamatodd) write(6,*) 'me,k,n,ncount',me,k,n,ntoddcount
       if (k .le. n) then
       increase=increase+n2
       if (L.eq.igamatodd)write(6,*) me,'Todd',ntoddcount,k,increase
       go to 123
       endif
      k=0
 1     continue
       goto 999           ! by Todd
c
c       write(6,*) 'before unscramble FFT: i,freal,fimag ='
c       do 22 i=1,n
c       write(6,*) i, freal(i), fimag(i)
c22     continue
c......unscramble results of FFT
c-----------------------------------------
       if (me .eq. 0) then
      call unscramble(freal,fimag,n,igama)
c       call MPI_BCAST(freal,n,MPI_DOUBLE_PRECISION,0,
c    $  MPI_COMM_WORLD,ierr)
c       call MPI_BCAST(fimag,n,MPI_DOUBLE_PRECISION,0,
c    $  MPI_COMM_WORLD,ierr)
c......output FFT solution
       write(6,*) 'after unscramble FFT: i,freal,fimag ='
       sumreal=0.d0
       sumimag=0.d0
       do 42 i=1,n
       write(6,*) i, freal(i), fimag(i)
       sumreal=sumreal+dabs( freal(i) )
       sumimag=sumimag+dabs( fimag(i) )
 42     continue
      write(6,*) 'FFT: sumreal,sumimag = ',sumreal,sumimag
       endif
c-----------------------------------------
 999   return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine bitreverse(m,igam,ip)
```

```
      ip=0
      j1=m
c
      do 2 i=1,igam
      j2=j1/2
      idiff=j1-j2*2
      ip=ip*2 + idiff
      j1=j2
 2    continue
c       write(6,*) 'p, or ii = ',ip
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine unscramble(freal,fimag,n,igama)
      implicit real*8(a-h,o-z)
c......purpose: parallel unscramble algorithms
      dimension freal(*),fimag(*)
c
      do 2 k=1,n
c     do 2 k=ME+1, n, NP
      m=k-1
      call bitreverse(m,igama,ii)
      i=ii+1
      if (i .le. k) go to 2
      temporeal=freal(k)
      tempoimag=fimag(k)
      freal(k)=freal(i)
      fimag(k)=fimag(i)
      freal(i)=temporeal
      fimag(i)=tempoimag
 2    continue
c......each processor has independently swap certain
c......terms of the arrays freal(n), and fimag(n).
c......now, we need to use appropriated mpi command
c......to "merge" all these partial results, and
c......make the entire updated array available to
c......all processors.
c     call MPI_merge & broadcast (freal, ...) ??
c     call MPI_merge & broadcast (fimag, ...) ??
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

References:

11.1 E.Oran Brigham, The Fast Fourier Transform, Prentice-Hall, Inc. (1974).

11.2 S.C. Chapra, and R.P. Canale, Numerical Methods for Engineers, 4th Edition, Mc-Graw Hill (2002).

11.3 W.H . Press, B.P. Flannery, S.A. Tenkolsky, and W.T. Vetterling, Numerical Recipies, Cambridge University Press (1989), Chapter 12.

11.4 M.T. Heath, Scientific Computing, Mc-Graw Hill (1997).

11.5 D.T. Nguyen, Parallel-Vector Equation Solvers For Finite Element Engineering Applications, Kluwer Academic/Plenum Publishers (2002).

11.6 D.T. Nguyen, Finite Element Methods: Parallel-Sparse Statics and Eigen-Solutions, Springer Publisher (2006).

11.7 Mario Paz, Structural Dynamics: Theory and Computation, 2nd Edition, Van Nostrand Inc. (1985).

11.8 R.W. Clough, and J. Penzien, Dynamics of Structures, Mc-Graw Hill (1975).