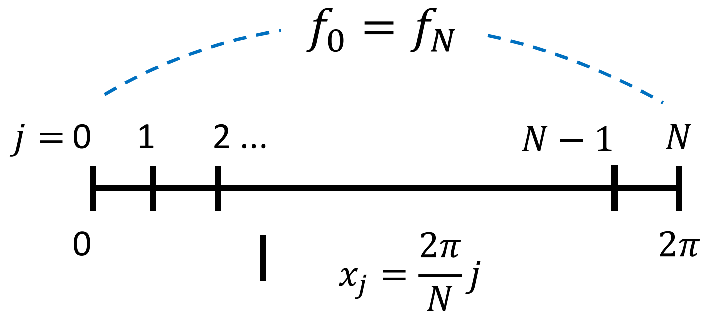
14. Fast Fourier Transform

Discrete Fourier Transform pair

- Given real f_j at $x_j = \frac{2\pi}{N} j \quad (j = 0, 1, 2, \dots, N-1)$

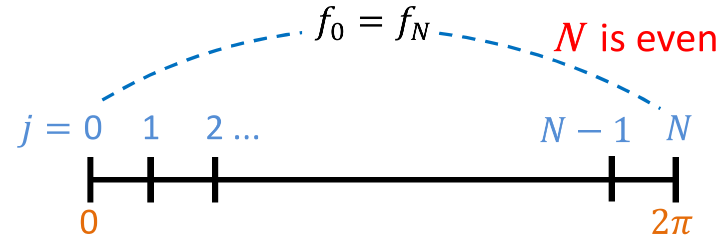


$$
\begin{cases}
f_j = \displaystyle\sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_n \, e^{inx_j} \\[2ex]
\hat{f}_n = \displaystyle\frac{1}{N} \sum_{j=0}^{N-1} f_j \, e^{-inx_j}
\end{cases}
$$

$$
\hat{f}_n = \hat{f}_{n+N} \qquad e^{inx_j} = e^{i(n+N)x_j}
$$

$$
\begin{cases}
f_j = \displaystyle\sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_n \, e^{inx_j} = \sum_{n=0}^{N-1} \hat{f}_n \, e^{inx_j} \\[2ex]
\hat{f}_n = \displaystyle\frac{1}{N} \sum_{j=0}^{N-1} f_j \, e^{-inx_j}
\end{cases}
$$

Computation of Discrete Fourier Transform



- Given real f_j at $x_j = \frac{2\pi}{N} j \quad (j = 0, 1, 2, \dots, N-1)$

- Compute complex $\hat{f}_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \cdot e^{-inx_j} = \frac{1}{N}\sum_{j=0}^{N-1} f_j \cdot w^{jn} \quad w \equiv e^{-i\frac{2\pi}{N}} \quad (n = 0, 1, 2 \dots, N-1)$

- Direct computation of discrete Fourier transform (DFT):

compute and store w^j

$$\hat{f}_1 = \frac{1}{N}\sum_{j=0}^{N-1} f_j \times w^j$$ compute and temporarily store $\underline{f_j \cdot \left(w^j\right)^1}$

$$\hat{f}_2 = \frac{1}{N}\sum_{j=0}^{N-1} \underline{f_j \cdot \left(w^j\right)^1} \times w^j$$ compute and temporarily store $f_j \cdot \left(w^j\right)^2$

……………………

$$\hat{f}_{N-1} = \frac{1}{N}\sum_{j=0}^{N-1} f_j \cdot \left(w^j\right)^{N-2} \times w^j$$

- The number of e^{-inx_j} computation is N.
- The total number of real multiplications $\approx 4N^2$, or the total number of complex multiplications $\approx N^2$

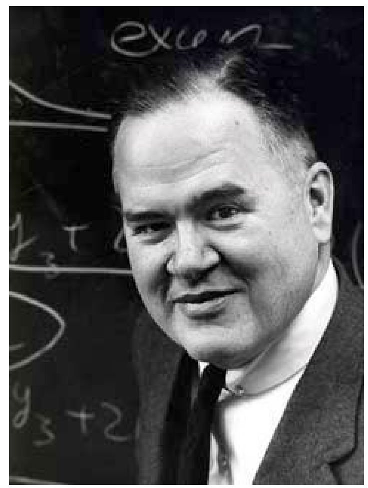
- There are many different fast algorithms to compute DFT involves total number of complex multiplications less than N^2.

Cooley–Tukey FFT Algorithm

- The best-known FFT algorithm (radix-2 decimation) is that developed in 1965 by J. Cooley and J. Tukey which reduces the number of complex multiplications to $O(N \log N)$.

  Cooley, J. & Tukey, J. 1965, An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*, vol.19, No.90, pp.297-301.

James William Cooley
(born 1926)

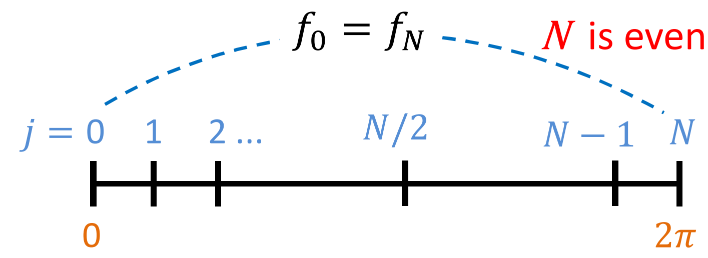


John Wilder Tukey
(1915 – 2000)

Carl Friedrich Gauss
(1777 – 1855)

- But it was later discovered that Cooley and Tukey had independently re-invented an algorithm known to Carl Friedrich Gauss around 1805.

Radix-2 Decimation



$$\hat{f}_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \cdot e^{-in\frac{2\pi}{N}j} \qquad n = 0, 1, 2 \ldots, N-1$$

$$\hat{f}_n = \frac{1}{N}\sum_{k=0}^{\frac{N}{2}-1} f_{2k} \cdot e^{-in\frac{2\pi}{N}2k} + \frac{1}{N}\sum_{k=0}^{\frac{N}{2}-1} f_{2k+1} \cdot e^{-in\frac{2\pi}{N}(2k+1)}$$

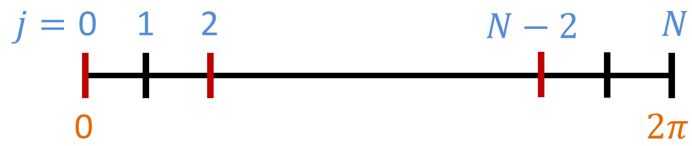
$$= \underbrace{\frac{1}{N}\sum_{k=0}^{\frac{N}{2}-1} f_{2k} \cdot e^{-in\frac{2\pi}{N/2}k}}_{\text{even-indexed part} \;\equiv\; \mathcal{E}_n} + e^{-in\frac{2\pi}{N}} \cdot \underbrace{\frac{1}{N}\sum_{k=0}^{\frac{N}{2}-1} f_{2k+1} \cdot e^{-in\frac{2\pi}{N/2}k}}_{\text{odd-indexed part} \;\equiv\; \mathcal{O}_n}$$

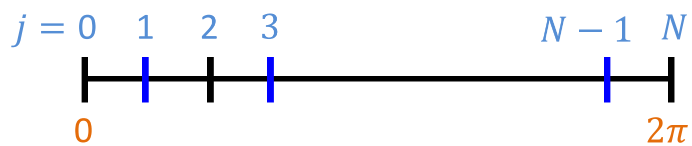
$$\because \begin{cases} \mathcal{E}_{n+\frac{N}{2}} = \dfrac{1}{N}\displaystyle\sum_{k=0}^{\frac{N}{2}-1} f_{2k} \cdot e^{-i\left(n+\frac{N}{2}\right)\frac{2\pi}{N/2}k} = \dfrac{1}{N}\displaystyle\sum_{k=0}^{\frac{N}{2}-1} f_{2k} \cdot e^{-in\frac{2\pi}{N/2}k} = \mathcal{E}_n \\ \mathcal{O}_{n+\frac{N}{2}} = \mathcal{O}_n \end{cases}$$

$$w \equiv e^{-i\frac{2\pi}{N}}$$

$$\therefore \left.\begin{cases} \hat{f}_n = \mathcal{E}_n + e^{-in\frac{2\pi}{N}} \cdot \mathcal{O}_n = \mathcal{E}_n + w^n \cdot \mathcal{O}_n \\ \hat{f}_{n+\frac{N}{2}} = \mathcal{E}_n + e^{-i\left(n+\frac{N}{2}\right)\frac{2\pi}{N}} \cdot \mathcal{O}_n = \mathcal{E}_n - e^{-in\frac{2\pi}{N}} \cdot \mathcal{O}_n = \mathcal{E}_n - w^n \cdot \mathcal{O}_n \end{cases}\right\} \; 0 \le n < \frac{N}{2}$$

$$\begin{cases} \mathcal{E}_n = \dfrac{1}{N}\displaystyle\sum_{k=0}^{\frac{N}{2}-1} f_{2k}\cdot e^{-in\frac{2\pi}{N/2}k} \\ \\ \mathcal{O}_n = \dfrac{1}{N}\displaystyle\sum_{k=0}^{\frac{N}{2}-1} f_{2k+1}\cdot e^{-in\frac{2\pi}{N/2}k} \end{cases}$$





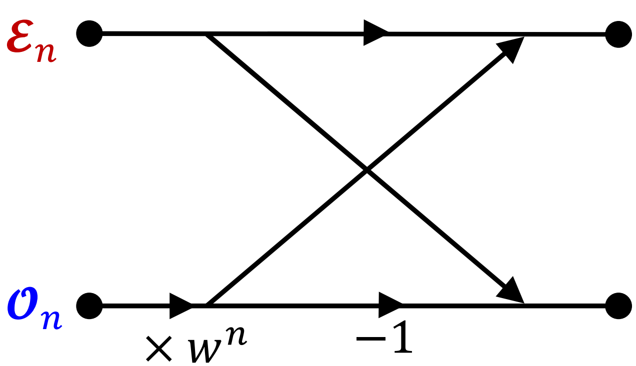
$\frac{N}{2}$ - point DFT

$$\begin{cases} \hat{f}_n = \mathcal{E}_n + w^n \cdot \mathcal{O}_n \\ \\ \hat{f}_{n+\frac{N}{2}} = \mathcal{E}_n - w^n \cdot \mathcal{O}_n \end{cases}$$

$0 \le n < \frac{N}{2}$

- **Butterfly diagram**

The above operation of radix-2 decimation in Cooley–Tukey FFT algorithm can be represented by the butterfly diagram:



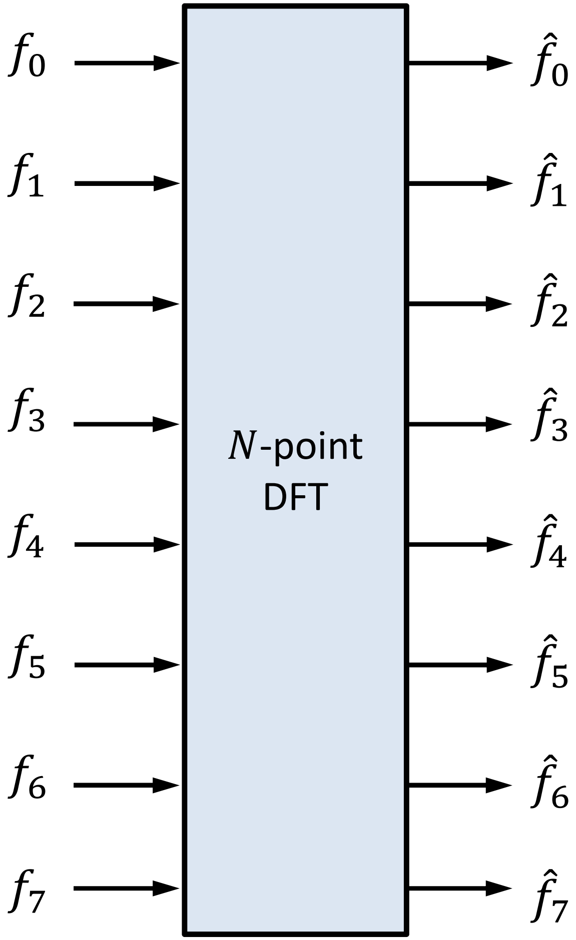
$\hat{f}_n = \mathcal{E}_n + w^n \cdot \mathcal{O}_n$

$\hat{f}_{n+\frac{N}{2}} = \mathcal{E}_n - w^n \cdot \mathcal{O}_n$

1 complex multiplication

$$\hat{f}_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \cdot e^{-in\frac{2\pi}{N}j} \qquad n = 0, 1, 2 \dots, N-1$$

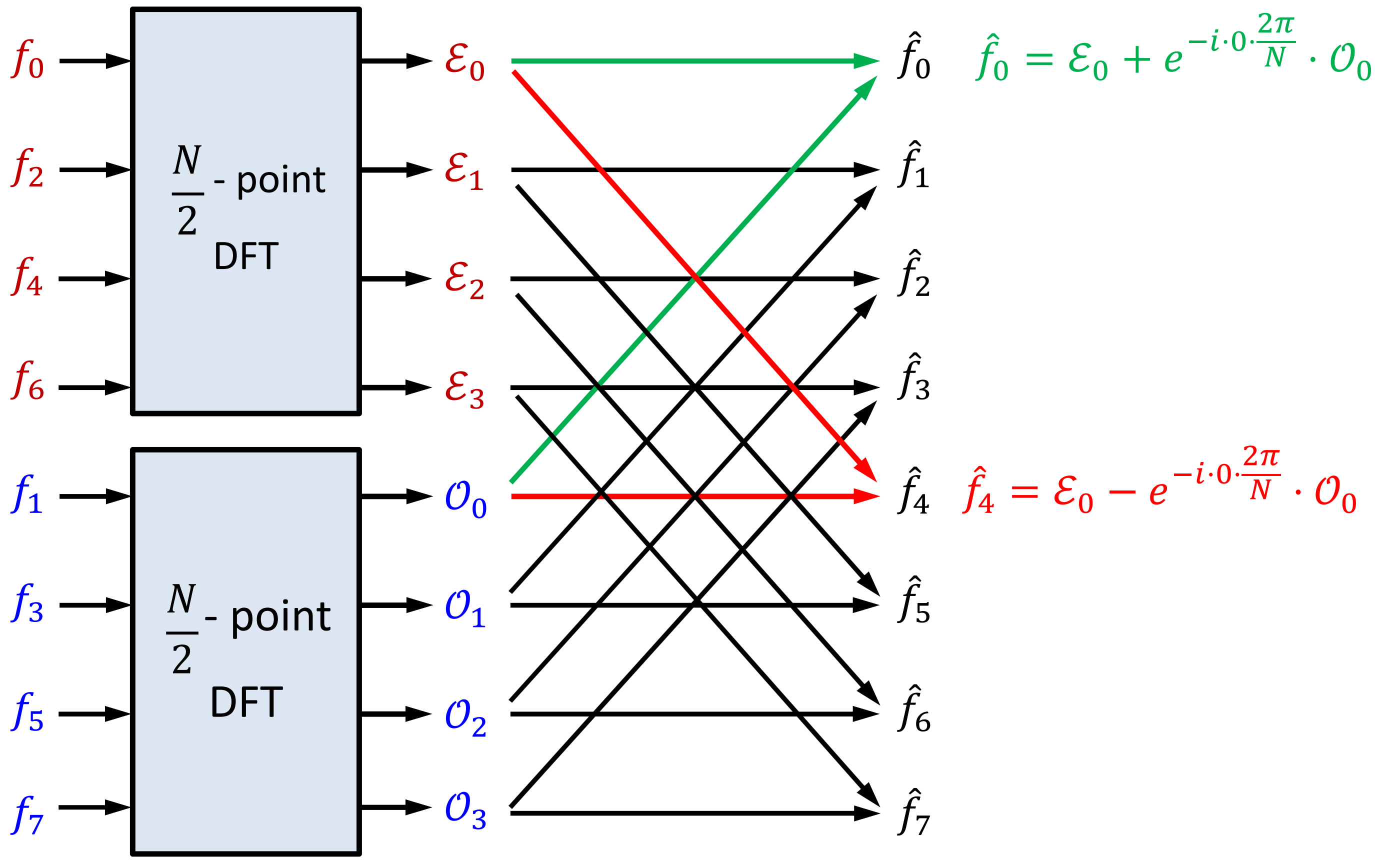
$\boxed{\text{eg: } N = 8}$



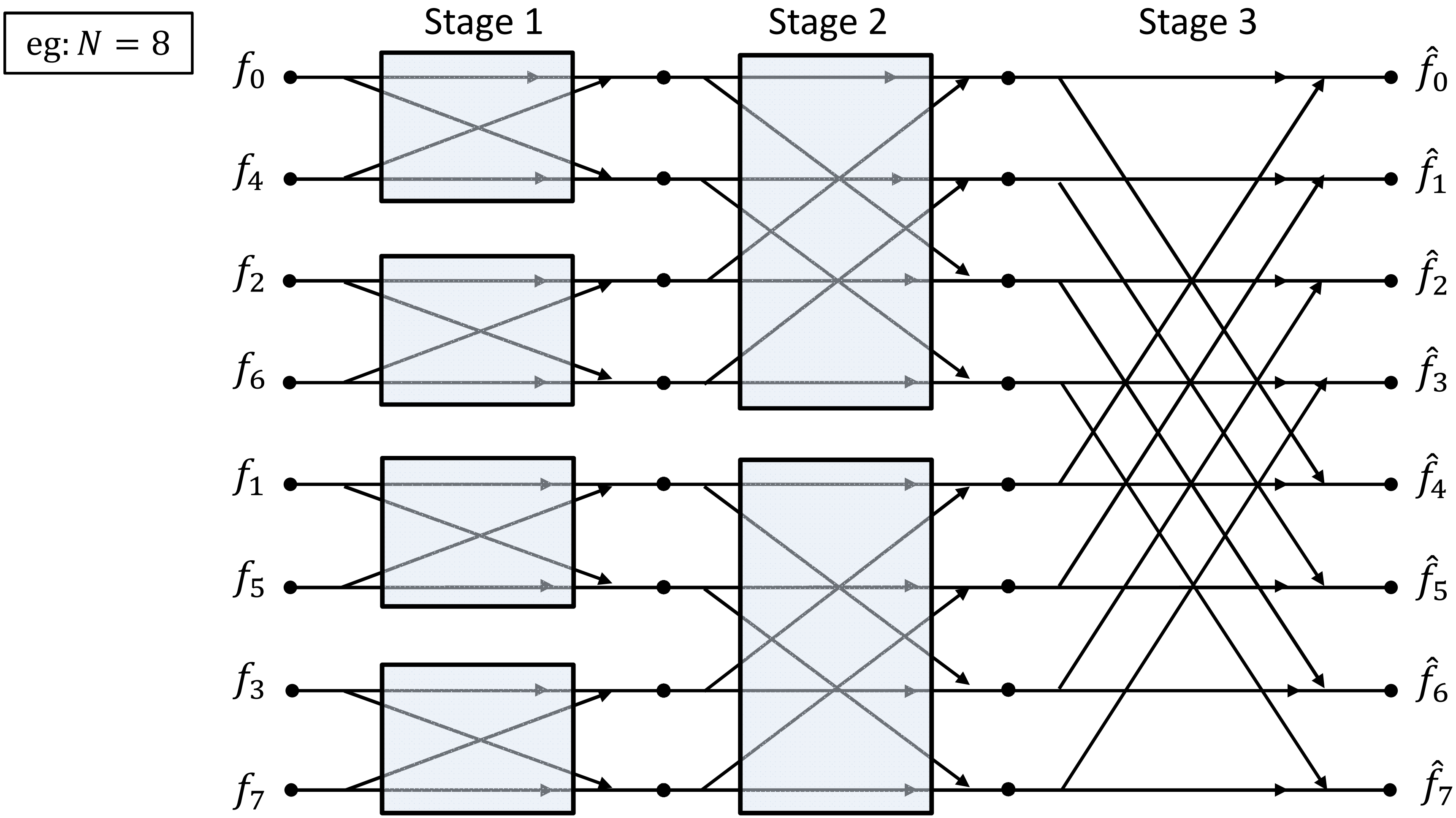
- N^2 complex multiplications

$$\boxed{\text{eg} : N = 8}$$

$$\begin{cases} \mathcal{E}_n = \frac{1}{8}\sum_{k=0}^{3} f_{2k} \cdot e^{-in\frac{2\pi}{4}k} \\ \mathcal{O}_n = \frac{1}{8}\sum_{k=0}^{3} f_{2k+1} \cdot e^{-in\frac{2\pi}{4}k} \end{cases} \qquad \begin{cases} \hat{f}_n = \mathcal{E}_n + w^n \cdot \mathcal{O}_n \\ \hat{f}_{n+\frac{N}{2}} = \mathcal{E}_n - w^n \cdot \mathcal{O}_n \end{cases} \qquad n = 0, 1, 2, 3$$



- $\left[\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2\right] + \frac{N}{2}$ complex multiplications
- The two $\frac{N}{2}$-point DFTs can be further broken down into four $\frac{N}{4}$-point DFTs.



- $\frac{N}{2} + \frac{N}{2} + \frac{N}{2}$ complex multiplications

- The DFTs can be broken down recursively.
- For $N = 2^{\nu}$, this radix-2 decimation can be performed $\log_2 N = \nu$ times.
- Thus the total number of complex multiplications is reduced to $\left(\frac{N}{2}\right) \log_2 N = \left(\frac{N}{2}\right) \nu$.

- **Comparison of numbers of complex multiplications**

	direct computing of DFT	Cooley–Tukey FFT algorithm
N	N^2	$(N/2)\log_2 N$
$2^2 = 4$	16	4
$2^4 = 16$	256	32
$2^8 = 256$	65,536	1024
$2^{10} = 1024$	1,048,676	5120
$2^{11} = 2048$	4,194,304	11,264
$2^{12} = 4096$	16,777,216	24,576

- The original Fortran program for computing DFT by FFT algorithm in the paper of Cooley, Lewis & Welch (1969)

[The Fast Fourier Transform and Its Applications, *IEEE Transactions on Education*, vol.12, no.1, pp.27-34]

```
      SUBROUTINE FFT(A,M)
      COPMLEX A(1024),U,W,T
      N = 2*M
      NV2 = N/2
      NMl = N-1
      J=1
      DO 7 I=1,NM1
      IF(I.GE.J) GO TO 5
      T = A(J)
      A(J) = A(I)
      A(I) = T
5     K=NV2
6     IF(K.GE.J) GO TO 7
      J = J-K
      K=K/2
      GO TO 6
7     J = J+K
      PI = 3.14159265358979
      DO 20 L=1,M
      LE = 2**L
      LE1 = LE/2
      U = (1.0,0.)
      W = CMPLX(COS(PI/LEl),SIN(PI/LEl))
      DO 20 J=1,LE1
      DO 10 I=J,N,LE
      IP = I+LE1
      T = A(IP)*U
      A(IP) = A(I)-T
10    A(I) = A(I)+T
20    U=U*W
      RETURN
      END
```

Storage in DFT of a Real Function

$$f_j = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_n \, e^{inx_j} = \sum_{n=0}^{N-1} \hat{f}_n \, e^{inx_j}$$

$$\hat{f}_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \, e^{-inx_j}$$

f_j complex $\longleftrightarrow$ $\hat{f}_n$ complex

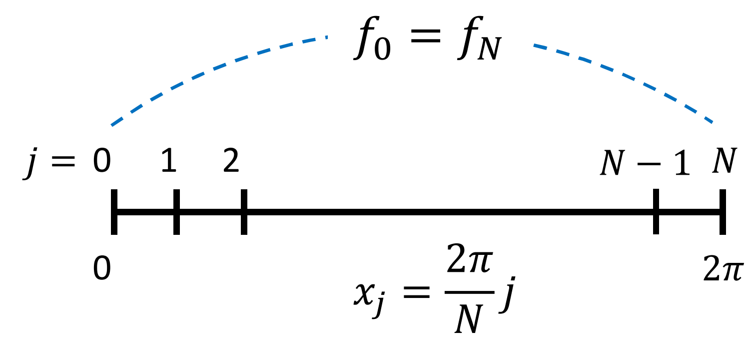
f_j real $\longleftrightarrow$ $\hat{f}_n$ complex

If f_j is real,

$$\hat{f}_0 = \frac{1}{N}\sum_{j=0}^{N-1} f_j = \text{real}$$

$$\because \quad \hat{f}_{-\frac{N}{2}} = \frac{1}{N}\sum_{j=0}^{N-1} f_j \, e^{-i\frac{(-N)}{2}\frac{2\pi}{N}j} = \text{real}$$

$$\hat{f}_{-n} = \frac{1}{N}\sum_{j=0}^{N-1} f_j \, e^{-i(-n)x_j} = \frac{1}{N}\sum_{j=0}^{N-1} f_j \, e^{inx_j} = \hat{f}_n^*$$



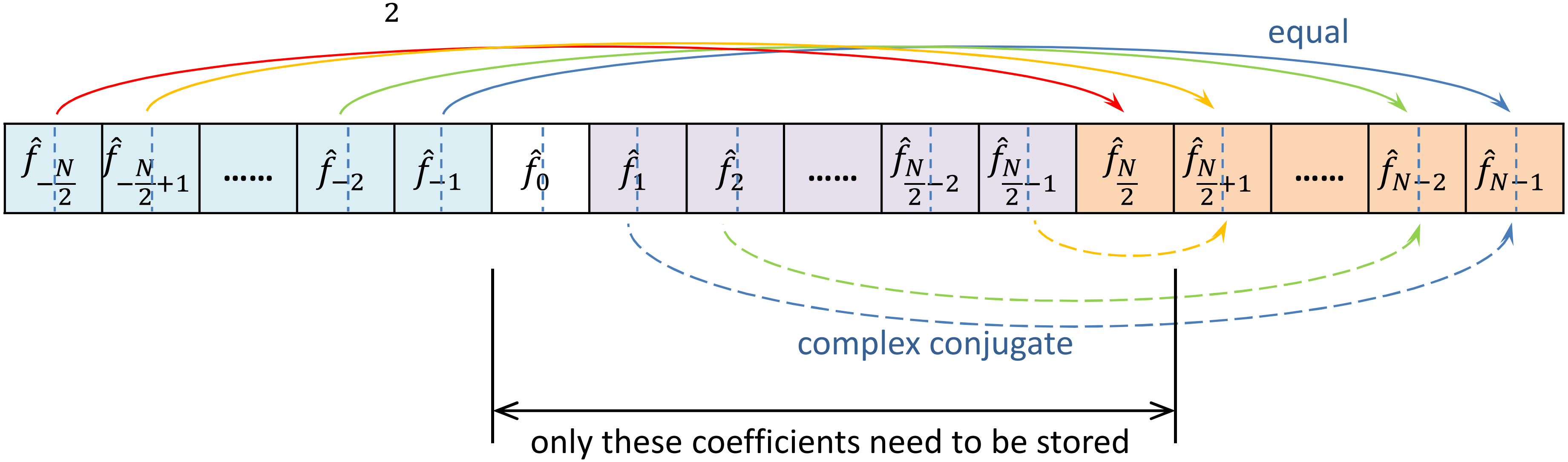
$\therefore$

equal

$\hat{f}_{-\frac{N}{2}}$	$\hat{f}_{-\frac{N}{2}+1}$	......	$\hat{f}_{-2}$	$\hat{f}_{-1}$	$\hat{f}_0$	$\hat{f}_1$	$\hat{f}_2$	......	$\hat{f}_{\frac{N}{2}-2}$	$\hat{f}_{\frac{N}{2}-1}$	$\hat{f}_{\frac{N}{2}}$	$\hat{f}_{\frac{N}{2}+1}$	......	$\hat{f}_{N-2}$	$\hat{f}_{N-1}$

complex conjugate

$\hat{f}_0 =$ real $\quad \hat{f}_{-\frac{N}{2}} =$ real $\quad \hat{f}_{-n} = \hat{f}_n^*$



The storage required for real f_j j= $0\sim N-1$:

$j=0$	1									$N-2$	$N-1$
f_0	f_1	f_2	f_3	......					......	f_{N-2}	f_{N-1}

∴ The storage required for complex $\hat{f}_n$ reduces to about half $n = 0\sim N/2$:

$n=0$	1	1	2	2					$\frac{N}{2}-1$	$\frac{N}{2}-1$	$\frac{N}{2}$
$\hat{f}_0^r$	$\hat{f}_1^r$	$\hat{f}_1^i$	$\hat{f}_2^r$	$\hat{f}_2^i$	......			......	$\hat{f}_{\frac{N}{2}-1}^r$	$\hat{f}_{\frac{N}{2}-1}^i$	$\hat{f}_{\frac{N}{2}}^r$

- Note that the coefficients of the DFT still include

$$n = -N/2 \sim N/2-1$$

$$f_j = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_n\, e^{inx_j} = \sum_{n=0}^{N-1} \hat{f}_n\, e^{inx_j}$$

Some FFT packages for computing discrete Fourier transform

Numerical Recipes

- in the book of Numerical Recipes: The Art of Scientific Computing.
- Use Cooley–Tukey radix-2 decimation algorithm.
- Note the expressions of the transform pair are different from the conventional ones.

FFTPACKT (FFT99 & FFT991)

- Developed by *Clive Temperton* at European Centre for Medium-Range Weather Forecasts (ECMWF).
- Optimized for Math Library in Cray and Compaq machines.
- Use mixed-radix algorithm.
- Available from ECMWF.
- N must be even and is a product of 2, 3 or 5 only.

NCAR FFTPACK

- Developed by Paul Swarztrauber of the National Center for Atmospheric Research (NCAR).
- Use mixed-radix algorithm by Clive Temperton.
- Includes complex, real, sine, cosine, and quarter-wave transforms.
- Available from NCAR or Netlib.
- The transform is most efficient when N is a product of small primes (2, 3, 5, 7, …).

FFTW

- Developed by Matteo Frigo and Steven G. Johnson of MIT.
- Available from FFTW.
- Written in C but there is Fortran interface.
- Can efficiently handle arbitrary size of data.
- Works best on sizes with small prime factors, with powers of two being optimal and large primes being worst case.
- Self optimizing: It automatically tunes itself for each hardware platform in order to achieve maximum performance.
- Known as the fastest free software implementation of the FFT algorithm.

  (Fastest Fourier Transform in the West)

Storage of the complex coefficient for different real FFT packages

Numerical Recipes

$n=0$	$\frac{N}{2}$	1	1	2	2					$\frac{N}{2}-1$	$\frac{N}{2}-1$
$\hat{f}_0^r$	$\hat{f}_{\frac{N}{2}}^r$	$\hat{f}_1^r$	$-\hat{f}_1^i$	$\hat{f}_2^r$	$-\hat{f}_2^i$	……			……	$\hat{f}_{\frac{N}{2}-1}^r$	$-\hat{f}_{\frac{N}{2}-1}^i$

NCAR FFTPACK

$n=0$	1	1	2	2					$\frac{N}{2}-1$	$\frac{N}{2}-1$	$\frac{N}{2}$
$\hat{f}_0^r$	$\hat{f}_1^r$	$\hat{f}_1^i$	$\hat{f}_2^r$	$\hat{f}_2^i$	……			……	$\hat{f}_{\frac{N}{2}-1}^r$	$\hat{f}_{\frac{N}{2}-1}^i$	$\hat{f}_{\frac{N}{2}}^r$

FFTW

$n=0$	1	2				$\frac{N}{2}-1$	$\frac{N}{2}$	$\frac{N}{2}-1$				2	1
$\hat{f}_0^r$	$\hat{f}_1^r$	$\hat{f}_2^r$	……		……	$\hat{f}_{\frac{N}{2}-1}^r$	$\hat{f}_{\frac{N}{2}}^r$	$\hat{f}_{\frac{N}{2}-1}^i$	……		……	$\hat{f}_2^i$	$\hat{f}_1^i$

FFT pairs computed in different real FFT packages

- **FTPACKT**

$$\begin{cases} \hat{f}_n = \dfrac{1}{N}\displaystyle\sum_{j=0}^{N-1} f_j\, e^{-inx_j} \\ f_j = \displaystyle\sum_{n=0}^{N-1} \hat{f}_n\, e^{inx_j} \end{cases}$$

- **Numerical Recipes**

- $\breve{f}_n$ and f_j' are forward and inverse FFT outputs from Numerical Recipes

$$\begin{cases} \breve{f}_n = \displaystyle\sum_{j=0}^{N-1} f_j\, e^{-inx_j} & \therefore\ \widehat{f}_n = \dfrac{1}{N}\breve{f}_n \\ f_j' = \dfrac{1}{2}\displaystyle\sum_{n=0}^{N-1} \widehat{f}_n\, e^{inx_j} & \therefore\ f_j = 2f_j' \end{cases}$$

- **NCAR FFTPACK & FFTW**

- $\breve{f}_n$ and f_j are forward and inverse FFT outputs from NCAR FFT and FFTW

$$\begin{cases} \breve{f}_n = \displaystyle\sum_{j=0}^{N-1} f_j\, e^{-inx_j} & \\ f_j = \displaystyle\sum_{n=0}^{N-1} \widehat{f}_n\, e^{inx_j} & \therefore\ \widehat{f}_n = \dfrac{1}{N}\breve{f}_n \end{cases}$$

- **Procedure to call 1-D real FFTW :**

1. Create a "plan" for FFT which contains all information necessary to compute the transform:

```
CALL DFFTW_PLAN_R2R_1D(PLAN_NAME,N,IN,OUT,KIND,FLAG)
```

`D`: double precision
`R2R`: real-to-real transform

`PLAN_NAME`: integer to store the plan name

`N`: array size

`IN`: input real array

`OUT`: output real array

`KIND` = `FFTW_R2HC (0)`; forward DFT, `OUT` stores the non-redundant half of the complex coefficients:

$$\hat{f}_0^{\,r},\ \hat{f}_1^{\,r},\ \hat{f}_2^{\,r},\ \hat{f}_3^{\,r},\dots,\ \hat{f}_{\frac{N}{2}-1}^{\,r},\ \hat{f}_{\frac{N}{2}}^{\,r},\ \hat{f}_{\frac{N}{2}-1}^{\,i},\dots,\ \hat{f}_3^{\,i},\ \hat{f}_2^{\,i},\ \hat{f}_1^{\,i}$$

= `FFTW_HC2R` (1); for inverse transform

`FLAG`: control the rigor and time of planning process

= `FFTW_MEASURE (0)`: Finds an optimized plan by actually computing several FFTs which may cost a few seconds. This flag will overwrite the input/output array, so the plan must be created before initializing the `IN` array.

= `FFTW_ESTIMATE (64)`: Just builds a reasonable plan that is probably sub-optimal. With this flag, the input/output arrays are not overwritten during planning.

2. Execute the plan for discrete fast Fourier transform:

```
CALL DFFTW_EXECUTE_DFT_R2R(PLAN_NAME,IN,OUT)
```

- **Example of calling FFTW 1-D real FFT routines**
  1. Download [a zipped file containing pre-built FFTW library]().
  2. Unzip and store the files (`libfftw3-3.dll libfftw3-3.lib`) in `GNU_emacs_Fortran` folder.
  3. Download [`fftw3.f90`]() and save it in the folder of the program.

```
> gfortran -L. -lfftw3-3 example_FFTW.f90
```

```
program example_fftw
! Example to call 1-D real FFT routine of FFTW

implicit none
include 'fftw3.f90'
integer, parameter :: N=16
integer*8 :: PLAN_FOR,PLAN_BAC
real*8,dimension(N) :: IN,OUT,IN2
real*8 :: xj
integer :: j,k,mode
real*8, parameter :: twopi=2.*acos(-1.)

! Discrete data of function f(x)=cos(x)+0.2*sin(2x)
do j=0,N-1
  xj=twopi*real(j)/real(N)
  IN(j)=cos(xj) +0.2*sin(2.*xj)
end do

write(*,*) "Original data"
do j=1,N
  write(*,100) j,IN(j)
end do
100 format(i4,f12.5)
```

(continued)

```
! Forward transform
call dfftw_plan_r2r_1d(PLAN_FOR,N,IN,OUT,FFTW_R2HC,FFTW_ESTIMATE)
call dfftw_execute_r2r(PLAN_FOR,IN,OUT)
OUT=OUT/real(N,KIND=8)         ! Normalize

write(*,*) "Fourier coefficient after forward FFT"
do k=1,N
  mode=k-1
  if(k > N/2+1) mode=N-k+1
  write(*,100) mode,OUT(k)
end do

! Backward transform
call dfftw_plan_r2r_1d(PLAN_BAC,N,OUT,IN2,FFTW_HC2R,FFTW_ESTIMATE)
call dfftw_execute_r2r(PLAN_BAC,OUT,IN2)
write(*,*) "Data after backward FFT"

do j=1,N
  write(*,100) j,IN2(j)
end do

! Destroy the plans
call dfftw_destroy_plan(PLAN_FOR)
call dfftw_destroy_plan(PLAN_BAC)

end program example_fftw
```

fftw3.f90

```fortran
integer ( kind = 4 ), parameter :: fftw_r2hc = 0
integer ( kind = 4 ), parameter :: fftw_hc2r = 1
integer ( kind = 4 ), parameter :: fftw_dht = 2
integer ( kind = 4 ), parameter :: fftw_redft00 = 3
integer ( kind = 4 ), parameter :: fftw_redft01 = 4
integer ( kind = 4 ), parameter :: fftw_redft10 = 5
integer ( kind = 4 ), parameter :: fftw_redft11 = 6
integer ( kind = 4 ), parameter :: fftw_rodft00 = 7
integer ( kind = 4 ), parameter :: fftw_rodft01 = 8
integer ( kind = 4 ), parameter :: fftw_rodft10 = 9
integer ( kind = 4 ), parameter :: fftw_rodft11 = 10
integer ( kind = 4 ), parameter :: fftw_forward = -1
integer ( kind = 4 ), parameter :: fftw_backward = +1
integer ( kind = 4 ), parameter :: fftw_measure = 0
integer ( kind = 4 ), parameter :: fftw_destroy_input = 1
integer ( kind = 4 ), parameter :: fftw_unaligned = 2
integer ( kind = 4 ), parameter :: fftw_conserve_memory = 4
integer ( kind = 4 ), parameter :: fftw_exhaustive = 8
integer ( kind = 4 ), parameter :: fftw_preserve_input = 16
integer ( kind = 4 ), parameter :: fftw_patient = 32
integer ( kind = 4 ), parameter :: fftw_estimate = 64
integer ( kind = 4 ), parameter :: fftw_estimate_patient = 128
integer ( kind = 4 ), parameter :: fftw_believe_pcost = 256
integer ( kind = 4 ), parameter :: fftw_dft_r2hc_icky = 512
integer ( kind = 4 ), parameter :: fftw_nonthreaded_icky = 1024
integer ( kind = 4 ), parameter :: fftw_no_buffering = 2048
integer ( kind = 4 ), parameter :: fftw_no_indirect_op = 4096
integer ( kind = 4 ), parameter :: fftw_allow_large_generic = 8192
integer ( kind = 4 ), parameter :: fftw_no_rank_splits = 16384
integer ( kind = 4 ), parameter :: fftw_no_vrank_splits = 32768
integer ( kind = 4 ), parameter :: fftw_no_vrecurse = 65536
integer ( kind = 4 ), parameter :: fftw_no_simd = 131072
```

```
>gfortran -L. -lfftw3-3 example_FFTW.f90
>a.exe
```

```
Original data
  1      1.00000
  2      1.06530
  3      0.90711
  4      0.52410
  5     -0.00000
  6     -0.52410
  7     -0.90711
  8     -1.06530
  9     -1.00000
 10     -0.78246
 11     -0.50711
 12     -0.24126
 13      0.00000
 14      0.24126
 15      0.50711
 16      0.78246
```

```
Fourier coefficient after forward FFT
  0      0.00000
  1      0.50000
  2      0.00000
  3     -0.00000
  4     -0.00000
  5     -0.00000
  6     -0.00000
  7     -0.00000
  8     -0.00000
  7      0.00000
  6      0.00000
  5      0.00000
  4      0.00000
  3      0.00000
  2     -0.10000
  1      0.00000
```

```
Data after backward FFT
  1      1.00000
  2      1.06530
  3      0.90711
  4      0.52410
  5     -0.00000
  6     -0.52410
  7     -0.90711
  8     -1.06530
  9     -1.00000
 10     -0.78246
 11     -0.50711
 12     -0.24126
 13      0.00000
 14      0.24126
 15      0.50711
 16      0.78246
```

Storage of complex coefficients in FFTW:

$n=0$	1	2					$\frac{N}{2}-1$	$\frac{N}{2}$	$\frac{N}{2}-1$					2	1
$\hat{f}_0^r$	$\hat{f}_1^r$	$\hat{f}_2^r$	……			……	$\hat{f}_{\frac{N}{2}-1}^r$	$\hat{f}_{\frac{N}{2}}^r$	$\hat{f}_{\frac{N}{2}-1}^i$	……			……	$\hat{f}_2^i$	$\hat{f}_1^i$

$$\cos(x) + 0.2\sin(2x)$$

$$= -0.1\sin(-2x) + 0.5\cos(-x) + 0.5\cos(x) + 0.1\sin(2x)$$

$n=0$	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
$\hat{f}_0^r$	$\hat{f}_1^r$	$\hat{f}_2^r$	$\hat{f}_3^r$	$\hat{f}_4^r$	$\hat{f}_5^r$	$\hat{f}_6^r$	$\hat{f}_7^r$	$\hat{f}_8^r$	$\hat{f}_7^i$	$\hat{f}_6^i$	$\hat{f}_5^i$	$\hat{f}_4^i$	$\hat{f}_3^i$	$\hat{f}_2^i$	$\hat{f}_1^i$

$n=0$	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
0.	0.5	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.1	0.

Test drivers for different real FFT packages

```
> gfortran -fdefault-real-8 NR_fft.f t_NR_fft.f90
> gfortran -fdefault-real-8 NCAR_fft.f t_NCAR_fft.f90
> gfortran -L. -lfftw3-3 t_fftw.f90
```

Numerical Recipes

```
Random number array:
n         array
0  0.99755959009261719
1  0.56682470761127335
2  0.96591537549612494
3  0.74792768547143218
4  0.36739089737475572
5  0.48063689875473148
6  7.37542636339845181E-002
7  5.35522927772724699E-003

Original arrangement of transfered array:
n         array
0  0.52567058096408081
1  7.54844506852897501E-002
2  2.07543321898765419E-002
3  0.18477288940714129
4  4.06601060421579452E-002
5  3.67723364521056612E-002
6  0.13678784098958879
7 -3.82673885583936502E-002

Re-arranged coefficients:
n        real                    imaginary
0  0.52567058096408081        -0.0000000000000000
1  2.07543321898765419E-002 -0.18477288940714129
2  4.06601060421579452E-002 -3.67723364521056612E-002
3  0.13678784098958879        3.82673885583936502E-002
4  7.54844506852897501E-002  -0.0000000000000000

Backward transfered array:
n         array
0  0.99755959009261719
1  0.56682470761127335
2  0.96591537549612483
3  0.74792768547143229
4  0.36739089737475583
5  0.48063689875473142
6  7.37542636339844071E-002
7  5.35522927772730251E-003
```

NCAR FFTPACK

```
Random number array:
n         array
0  0.99755959009261719
1  0.56682470761127335
2  0.96591537549612494
3  0.74792768547143218
4  0.36739089737475572
5  0.48063689875473148
6  7.37542636339845181E-002
7  5.35522927772724699E-003

Original arrangement of transfered array:
n         array
0  0.52567058096408081
1  2.07543321898766217E-002
2 -0.18477288940714132
3  4.06601060421579452E-002
4 -3.67723364521056612E-002
5  0.13678784098958874
6  3.82673885583938028E-002
7  7.54844506852897501E-002

 Re-arranged coefficients:
n        real                    imaginary
0  0.52567058096408081         0.0000000000000000
1  2.07543321898766217E-002 -0.18477288940714132
2  4.06601060421579452E-002 -3.67723364521056612E-002
3  0.13678784098958874        3.82673885583938028E-002
4  7.54844506852897501E-002   0.0000000000000000

 Backward transfered array:
n         array
0  0.99755959009261730
1  0.56682470761127335
2  0.96591537549612483
3  0.74792768547143229
4  0.36739089737475578
5  0.48063689875473137
6  7.37542636339844071E-002
7  5.35522927772730251E-003
```

FFTW

```
Random number array:
n         array
0  0.99755959009261719
1  0.56682470761127335
2  0.96591537549612494
3  0.74792768547143218
4  0.36739089737475572
5  0.48063689875473148
6  7.37542636339845181E-002
7  5.35522927772724699E-003

Original arrangement of transfered array:
n         array
0  0.52567058096408081
1  2.07543321898766148E-002
2  4.06601060421579313E-002
3  0.13678784098958877
4  7.54844506852897779E-002
5  3.82673885583937959E-002
6 -3.67723364521056612E-002
7 -0.18477288940714132

Re-arranged coefficients:
n        real                    imaginary
0  0.52567058096408081         0.0000000000000000
1  2.07543321898766148E-002 -0.18477288940714132
2  4.06601060421579313E-002 -3.67723364521056612E-002
3  0.13678784098958877        3.82673885583937959E-002
4  7.54844506852897779E-002   0.0000000000000000

Backward transfered array:
n         array
0  0.99755959009261719
1  0.56682470761127335
2  0.96591537549612494
3  0.74792768547143229
4  0.36739089737475572
5  0.48063689875473142
6  7.37542636339845042E-002
7  5.35522927772715332E-003
```

Comparison of efficiency of various FFT packages

Perform forward FFT of 2048 real random numbers, and repeat 1000 times

Test environment:

Operation System: Windows 7   64bit

CPU: Intel i5-750 2.66 GHz

Compiler: GNU gfortran

method	CPU time (seconds)
direct summation	711.8014
direct summation (store trigonometric terms)	67.8448
FFT in Numerical Recipes	3.7128
NCAR FFTPACK5	0.3276
FFTW 3.3	0.0624

Homework

1. Revise the two DFT subroutines developed in previous lecture on [Discrete Fourier Analysis] such that they also compute inverse DFT. Use an integer flag to control the direction of the transform.

2. Generate and store 1024 real random numbers ranging between -1 and 1.

3. Test the developed DFT subroutines by performing the forward transform and then the backward transform of the real random numbers. The backward calculation should result in the original data. Compare the backward-transformed results with the stored random numbers and find the maximum absolute error. Output the results.

4. Perform the same test using the real FFT subroutine in FFTW.

5. Compute forward DFT of the 1024 real random numbers using the two developed DFT subroutines and the real FFT subroutine in FFTW. Repeat each computation 100 times and monitor the CPU time. Output the results.

How to turn in the program:

1. Create the file name based on your student ID and the lecture number. For example, if your student ID is b01234567 and the lecture number is 14, the filename of your program is `b01234567_hw14.f90`

2. Use your NTU email account to send out an email to: `d01525011@ntu.edu.tw`
   Use the filename of your program as the subject of the email.
   Attach the file to the email.

   Mail to: `d01525011@ntu.edu.tw`
   Subject: `b01234567_hw14.f90`
   File to attach: `b01234567_hw14.f90`

3. Ask to receive a return receipt as shown in the figure.
   Please check with the TA, if you do not receive the receipt 1 hour after sending the email.