23 Discrete Fourier transform

Any finite set of N evenly spaced samples $f(t_n) = f_n$ is produced by operations in the time domain

1. starting with a continuous function $f(t)$ with range $-\infty < t < +\infty$
2. multiplying by a comb with spacing $\Delta t = T/N$
3. applying a boxcar window over the range $0 < t < T$

which are equivalent to operations in the frequency domain

1. starting with a continuous function $F(\omega)$ with range $-\infty < \omega < +\infty$
2. convolving by a comb with spacing $\Delta\omega = 2\pi N/T$
3. convolving with a sinc function

that produce an infinite number of (degraded & possibly overlapping) copies of $F(\omega)$. Most of this information is redundant, and only N frequency components are actually required.

23.1 Aliasing

The second hand on an analog clock[1] completes one revolution every minute. A normal human observer will see a continuous clockwise progression with a 60-second period (0.0167 Hz frequency). If the observer is blinking every second, then they will instead see a discrete sequence of snapshots as shown in the first row of Figure 23.1. This sequence can be reasonably interpreted as being due to a continuous progression with a 60s period. Similarly, an observer that only looks at the clock once every 5 or 10-seconds can still make

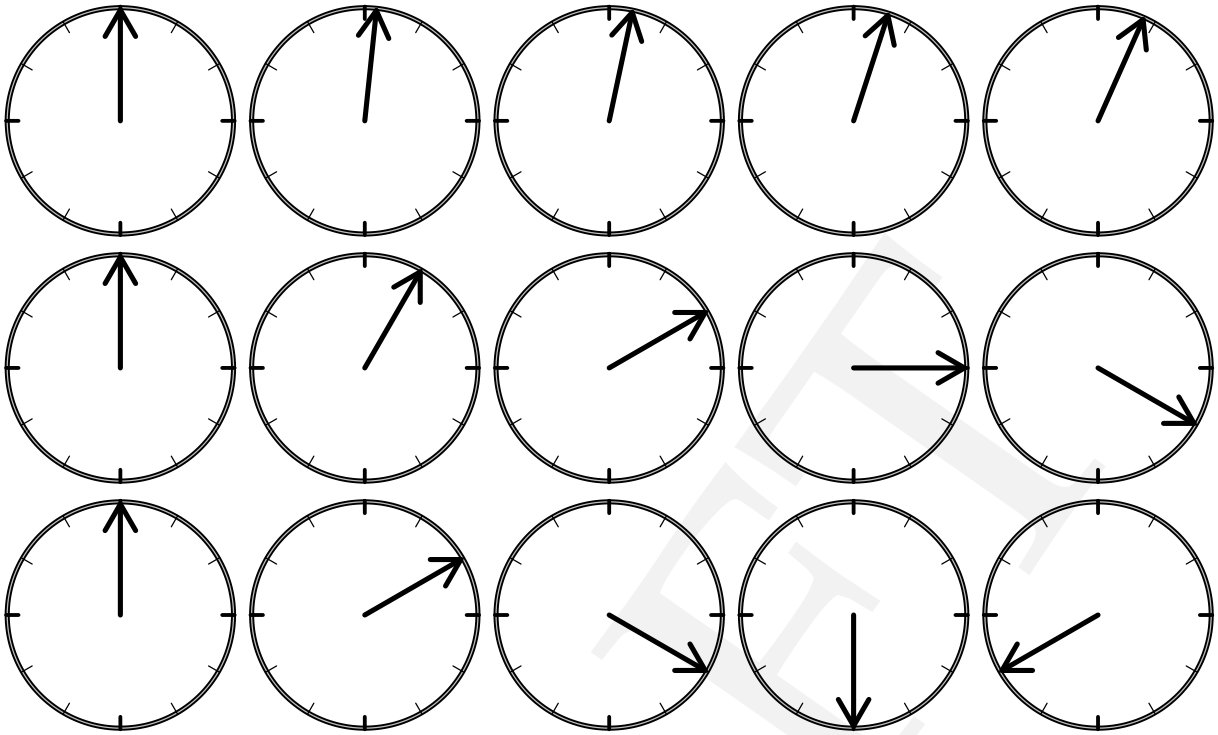

Figure 23.1: The second hand on a clock observed at 1, 5, and 10-second intervals

a reasonable case that the system has a 1-minute period.

The situation is less clear if observations are made only once per minute. In this case the clock appears the same each time, and it is impossible to distinguish between a broken clock and one that is working perfectly.

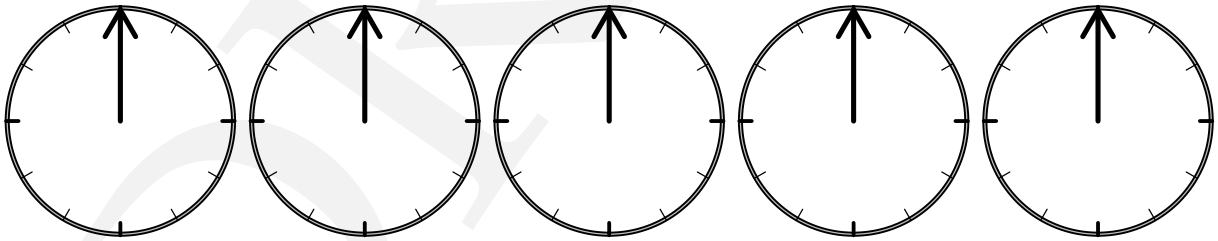

Figure 23.2: The second hand on a clock observed at 60-second intervals

Even more confusing, a clock observed at 59-second intervals will appear to be moving backwards with a period of 60-seconds.

[1] or consider the seconds field of a digital clock

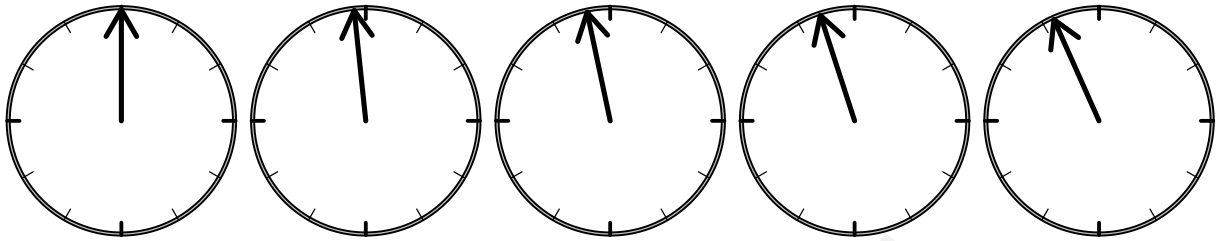

Figure 23.3: The second hand on a clock observed at 59-second intervals

23.1.1 Nyquist

A system with period T must be sampled at a rate at least twice per cycle to avoid ambiguity

$$\Delta t \geq \frac{1}{2}T \tag{23.1}$$

Alternatively, the *Nyquist frequency* is defined to be

$$f_N = \frac{1}{2\Delta t} \tag{23.2}$$

the highest frequency that can be resolved for a given sampling rate Δt

To do #23.1: Nyquist bandwidth

Strictly speaking, the Nyquist theorem states that the bandwidth is inversely proportional to sampling rate

$$f_2 - f_1 = \frac{1}{\Delta t}$$

which for a symmetric range centered on DC reduces to

$$+f_N - (-f_N) = 2f_N = \frac{1}{\Delta t} \qquad \Rightarrow \qquad f_N = \frac{1}{2\Delta t}$$

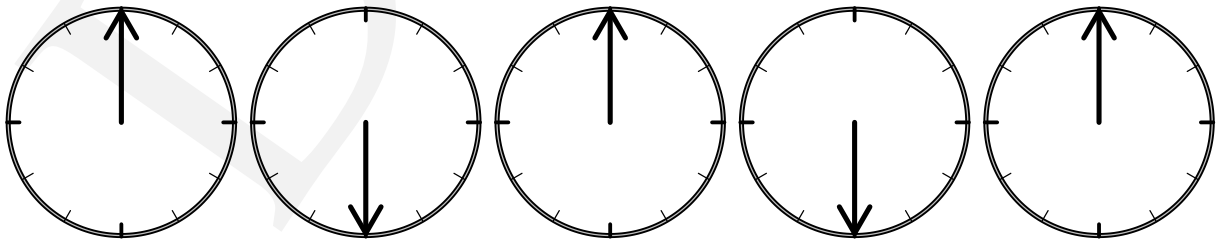

Figure 23.4: The second hand on a clock observed at 30-second intervals (Nyquist rate)

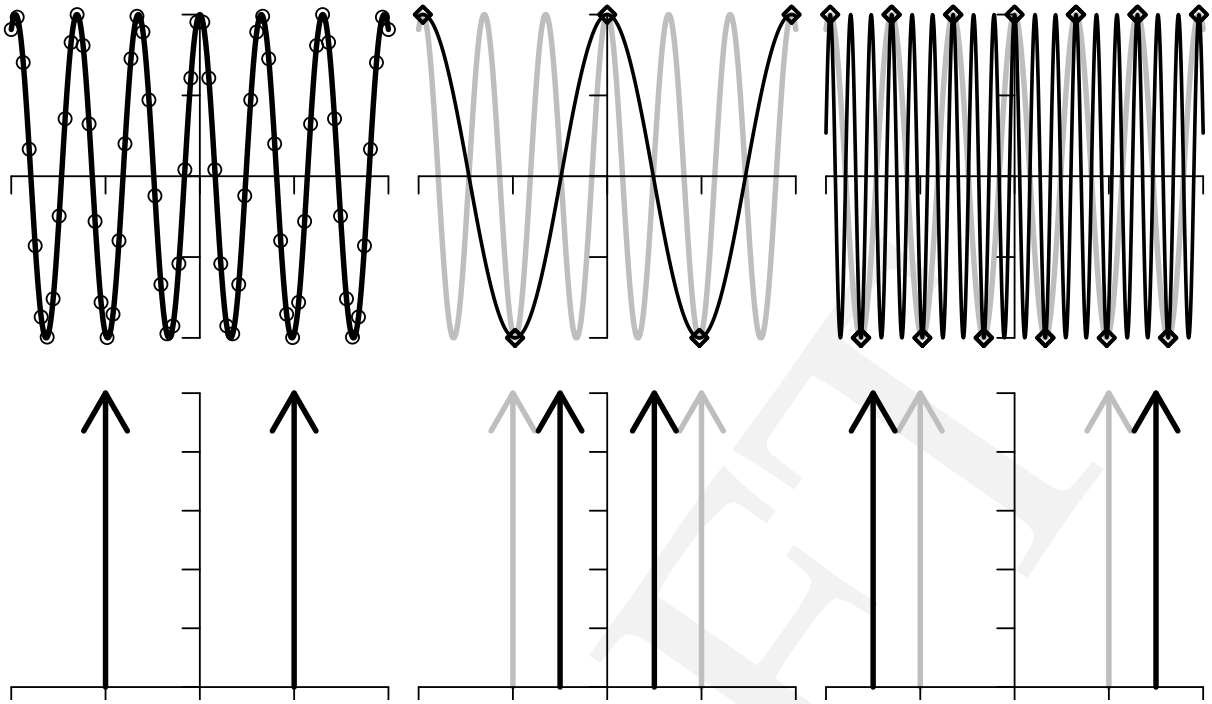

Figure 23.5: Sinusoids of very different frequencies may be indistinguishable at certain sampling rates

23.2 Discrete Fourier transform

$$F\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} e^{-i2\pi nk/N} f(kT) \qquad n = 0, 1, \ldots, N-1 \tag{23.3}$$

then $kT \to k$ and $n/NT \to n$ for simpler notation

$$F(n) = \sum_{k=0}^{N-1} e^{-i2\pi nk/N} f(k) = \sum_{k=0}^{N-1} W^{n\times k} f(k) \qquad W = e^{-i2\pi/N} = \sqrt[N]{e^{-i2\pi}} \tag{23.4}$$

The DFT is easy to compute, but is not usually provided by analysis tools. Instead there is an FFT routine that appears to do the same thing. For example, the Fourier transform of a delta function should be a constant

```
> w= c(1.0, 0,0,0,0,0,0,0); print(w)
[1] 1 0 0 0 0 0 0 0
> fft(w)
[1] 1+0i 1+0i 1+0i 1+0i 1+0i 1+0i 1+0i 1+0i
```

Take the Fourier transform of a shifted delta function.

```
> w= c(0.0, 1.0,0,0,0,0,0,0); print(w); fft(w)
[1] 0 1 0 0 0 0 0 0
[1]  1.0000000+0.0000000i  0.7071068-0.7071068i
[3]  0.0000000-1.0000000i -0.7071068-0.7071068i
[5] -1.0000000+0.0000000i -0.7071068+0.7071068i
[7]  0.0000000+1.0000000i  0.7071068+0.7071068i
```

Take the Fourier transform of a shifted imaginary delta function.

```
> w= complex(imaginary=c(0,1,0,0,0,0,0,0)); print(w); fft(w)
[1] 0+0i 0+1i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
[1]  0.0000000+1.0000000i  0.7071068+0.7071068i
[3]  1.0000000+0.0000000i  0.7071068-0.7071068i
[5]  0.0000000-1.0000000i -0.7071068-0.7071068i
[7] -1.0000000-0.0000000i -0.7071068+0.7071068i
```

23.2.1 N=1

```
np.set_printoptions(suppress=True,precision=3)
x=np.array([0]); y=np.fft.fft(x); print x,'=>',y
x=np.array([1]); y=np.fft.fft(x); print x,'=>',y

[0] => [ 0.+0.j]
[1] => [ 1.+0.j]
```

23.2.2 N=2

$$\begin{bmatrix} F(0) \\ F(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & W \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \end{bmatrix} \tag{23.5}$$

$$W = e^{-i2\pi/2} = \sqrt[2]{e^{-i2\pi}} = e^{-i\pi} = -1 \tag{23.6}$$

$$F_0 = f_0 + f_1 \tag{23.7}$$

$$F_1 = f_0 - f_1 \tag{23.8}$$

```
x=np.array([0,0]); y=np.fft.fft(x); print x,'=>',y
x=np.array([0,+1]); y=np.fft.fft(x); print x,'=>',y
x=np.array([-1,+1]); y=np.fft.fft(x); print x,'=>',y

[ 0  0] => [ 0.+0.j  0.+0.j]
[ 0  1] => [ 1.+0.j -1.+0.j]
[-1  1] => [ 0.+0.j -2.+0.j]
```

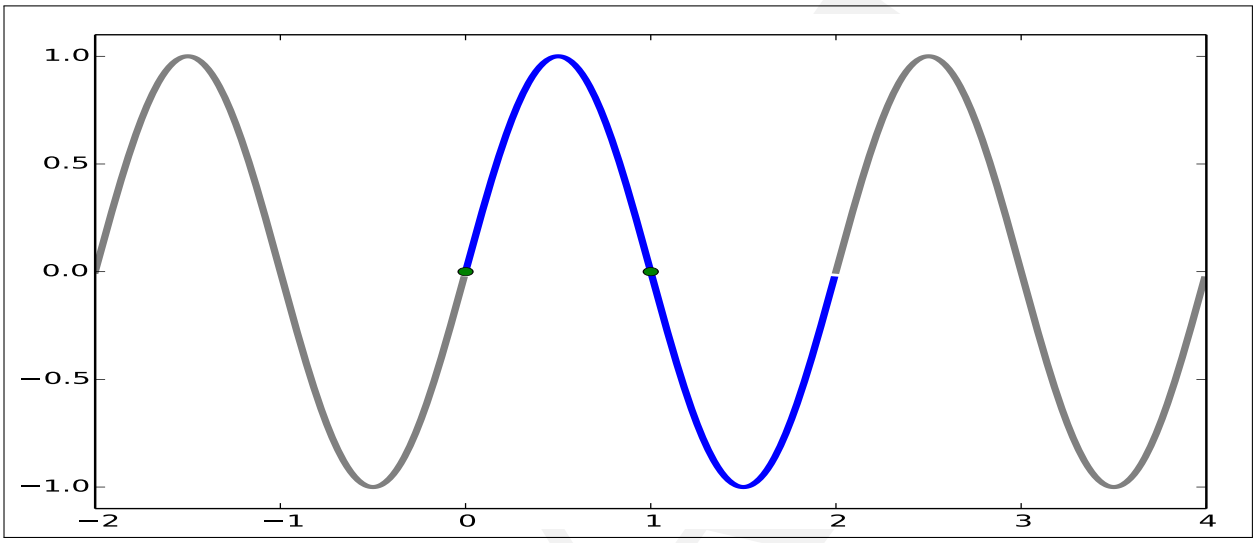

Figure 23.6

23.2.3 N=4

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{23.9}$$

$$W = e^{-i2\pi/4} = \sqrt[4]{e^{-i2\pi}} = e^{-i\pi/2} = i \tag{23.10}$$

$$\begin{aligned} F_0 &= f_0 + f_1 + f_2 + f_3 && (23.11)\\ F_1 &= f_0 + if_1 - f_2 - if_3 && (23.12)\\ F_2 &= f_0 - f_1 + f_2 - f_3 && (23.13)\\ F_3 &= f_0 - if_1 - f_2 + if_3 && (23.14)\\ & && (23.15) \end{aligned}$$

```
n=4  ; t = np.arange(0, n, 1)
for k in [0,1,2,3,4]:
    x = np.exp(2j*np.pi*t*k/n)
    y=np.fft.fft(x)
    print k,': ',x,' => ',y

0 :  [ 1.+0.j  1.+0.j  1.+0.j  1.+0.j]  =>  [ 4.+0.j  0.+0.j  0.+0.j
   0.+0.j]
1 :  [ 1.+0.j  0.+1.j -1.+0.j -0.-1.j]  =>  [-0.+0.j  4.-0.j  0.+0.j
   0.+0.j]
2 :  [ 1.+0.j -1.+0.j  1.-0.j -1.+0.j]  =>  [ 0.+0.j -0.+0.j  4.-0.j
   0.+0.j]
3 :  [ 1.+0.j -0.-1.j -1.+0.j  0.+1.j]  =>  [ 0.+0.j  0.+0.j -0.+0.j
   4.-0.j]
4 :  [ 1.+0.j  1.-0.j  1.-0.j  1.-0.j]  =>  [ 4.-0.j  0.+0.j  0.+0.j
   -0.+0.j]
```

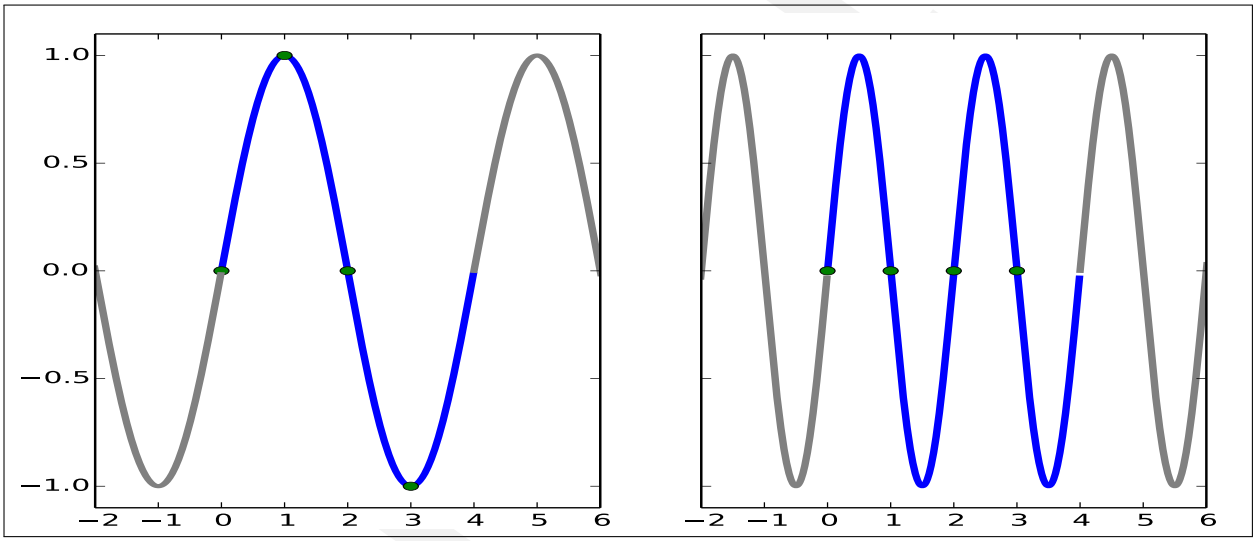

Figure 23.7

23.2.4 N=8

The discrete Fourier transform (DFT) for $N = 8$

$$F(k\Delta\omega) = \sum_{j=0}^{7} e^{-ijk/N} f(j\Delta t) \tag{23.16}$$

can be written as a matrix product

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \\ F(4) \\ F(5) \\ F(6) \\ F(7) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ 1 & W^2 & W^4 & W^6 & W^8 & W^{10} & W^{12} & W^{14} \\ 1 & W^3 & W^6 & W^9 & W^{12} & W^{15} & W^{18} & W^{21} \\ 1 & W^4 & W^8 & W^{12} & W^{16} & W^{20} & W^{24} & W^{28} \\ 1 & W^5 & W^{10} & W^{15} & W^{20} & W^{25} & W^{30} & W^{35} \\ 1 & W^6 & W^{12} & W^{18} & W^{24} & W^{30} & W^{36} & W^{42} \\ 1 & W^7 & W^{14} & W^{21} & W^{28} & W^{35} & W^{42} & W^{49} \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \end{bmatrix} \tag{23.17}$$

where

$$W = e^{-i2\pi/N} \tag{23.18}$$

This will clearly require N multiplications and additions to determine each frequency component, so the complete transform will have computational requirements of $\mathcal{O}(N^2)$.

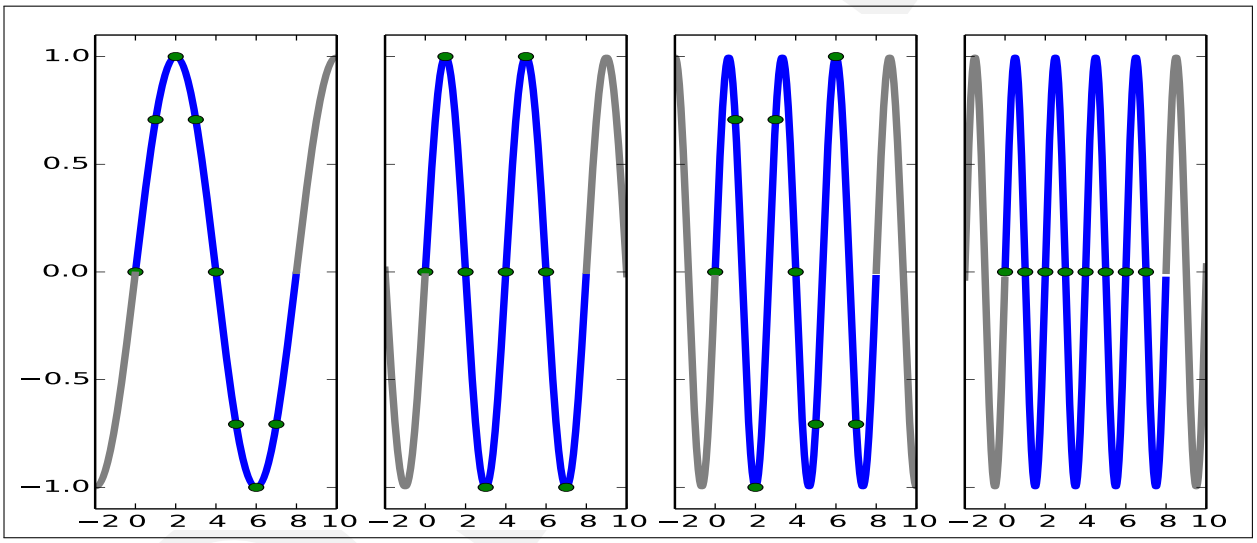

Figure 23.8

23.3 Fast Fourier Transform

The Fast Fourier Transform (FFT) is a computationally efficient algorithm for producing a DFT.

23.3.1 N=4

The 4-element DFT matrix representation in Equation 23.9 can be simplified

$$\begin{aligned}\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix}\end{aligned} \tag{23.19}$$

by applying the property

$$W^{n \times k} = W^{n \times k \mod N} \tag{23.20}$$

Example 23.1: Complex modulo
Consider the case $N = 4, n = 2, k = 3$

$$W^6 = \exp\left[-i\left(\frac{2\pi}{4}\right)6\right] = \exp\left[-i3\pi\right] = \exp\left[-i\pi\right] = \exp\left[-i\left(\frac{2\pi}{4}\right)2\right] = W^2 \tag{23.21}$$

so $W^2 = W^6 = W^{10} \ldots$

The modulo matrix can now be factored into two sparse 4×4 matrices (note the shuffling of F)

$$\begin{bmatrix} F(0) \\ F(2) \\ F(1) \\ F(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{23.22}$$

Converting f to F now involves two much simpler matrix multiplications. Half the elements of the factored matrices are zero, while half of the remainder will be ± 1. It can be shown that the reduced form requires only 4 complex multiplications and 8 complex additions versus 16 and 12 for the original case. This is a factor of 2 reduction in number of multiplications (which are much more "expensive" than addition).

23.3.2 N=8

It can also be shown that the 8-element DFT matrix (Equation 23.17) can be factored into four sparse matrices

$$F = M_1 \times M_2 \times M_3 \times M_4 \times f \tag{23.23}$$

where three of the matrices each require $2N$ multiplications and additions

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & W^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & W^3 \\ 1 & 0 & 0 & 0 & W^4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & W^5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & W^6 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & W^7 \end{bmatrix} \tag{23.24}$$

$$M_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & W^2 & 0 & 0 & 0 & 0 \\ 1 & 0 & W^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & W^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & W^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & W^4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & W^6 \end{bmatrix} \tag{23.25}$$

$$M_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & W^4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & W^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & W^4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & W^4 \end{bmatrix} \tag{23.26}$$

and the fourth just shuffles the elements.

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{23.27}$$

This result can be represented as a "Butterfly" diagram with three stages each only involving combinations of pairs of samples.

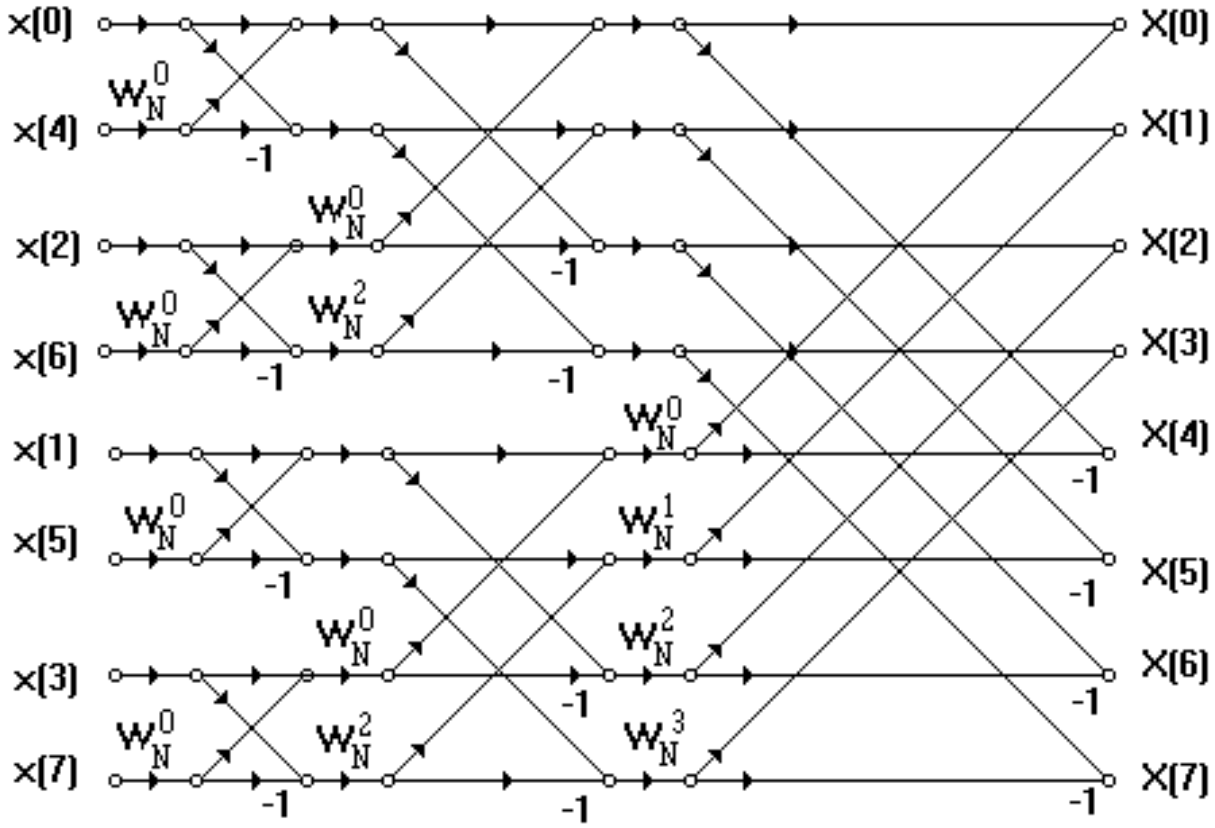

Figure 23.9

23.3.3 N=?

The simplest and most efficient FFTs algorithms apply for $N = 2^d$ where d is an integer (eg. $N = 2, 4, 8, \ldots, 256, 512, \ldots$). It can be shown that the computational requirements are $\mathcal{O}(NlogN)$, while straightforward matrix multiplication would require N^2 operations. The difference is negligible for small N but rapidly becomes important as N increases.

Example 23.2: log N versus N
If $N = 4$ then an FFT might only be 2 times faster than a DFT. For $N = 64$ the FFT would be 15 times faster, while for $N = 1024$ the DFT would takes 15000% more time than the FFT.

More general algorithms can work efficiently if N can be factored into small numbers (eg. $N = 1050 = 2 \times 3 \times 5 \times 5 \times 7$) but these reduce to the slow DFT if N is a prime number.

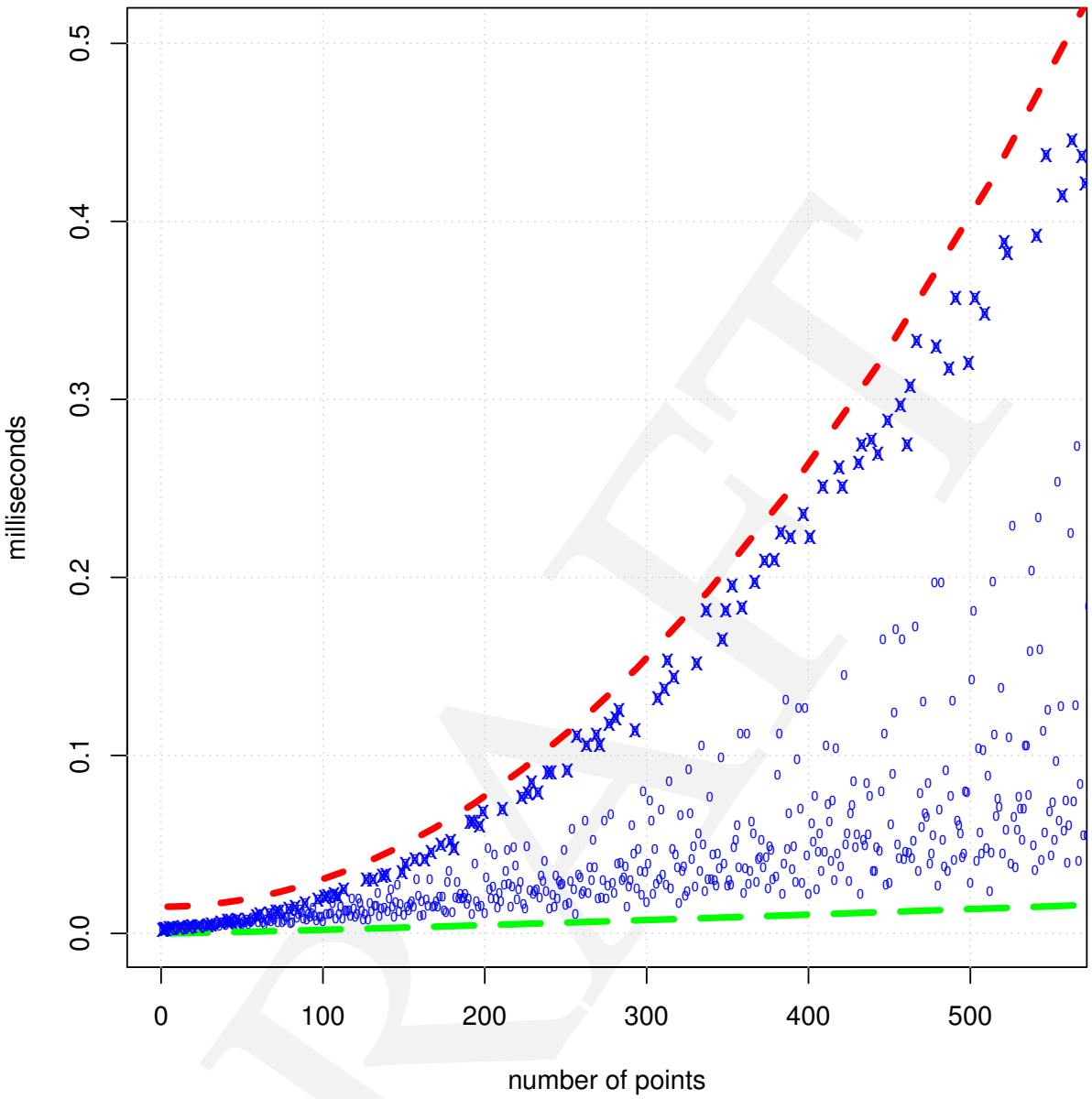

Figure 23.10: Time required for FFT of small number of points. Lower green dashed line is $N \log N$, upper red dashed line is N^2, prime numbers are plotted with an X.

23.3.4 Frequency packing

It is physically reasonable to think of frequency components in sequential order from minus Nyquist frequency though DC up to the positive Nyquist frequency

$$-f_n, -f_{n-1}, \ldots, -f_2, -f_1, DC, +f_1, +f_2, \ldots, +f_{n-1}, +f_n \tag{23.28}$$

and this is certainly how data are usually displayed. However, in order to maximize computational efficiency most FFT routines pack frequency domain components like this

$$DC, +f_1, +f_2, \ldots, +f_{n-1}, \pm f_n, -f_{n-1}, \ldots, -f_2, -f_1, \tag{23.29}$$