Heart of FFT code:

$\log N$ stages

```
LE = 1.;
for (L=1; L<=M; L++)                          /* stage L */
 { LE1 = LE;                                  /* LE1 = LE/2 */
   LE  *= 2;                                  /* LE  = 2^L */

   Ur = 1.0;
   Ui = 0.;
   Wr = cos(M_PI/(float)LE1);
   Wi = -sin(M_PI/(float)LE1);
```

$\Big\}\ W = e^{-j2\pi/LE}$

```
   for (j=1; j<=LE1; j++)

    { for (i=j; i<=N; i+=LE) /* butterfly */
       { ip = i+LE1;
         Tr = ar[ip-1]*Ur-ai[ip-1]*Ui;
         Ti = ar[ip-1]*Ui+ai[ip-1]*Ur;
         ar[ip-1] = ar[i-1] - Tr;
         ai[ip-1] = ai[i-1] - Ti;
         ar[i-1]  = ar[i-1] + Tr;
         ai[i-1]  = ai[i-1] + Ti; }  /* end of butterfly */
```

merge

```
      Ur_old = Ur;
      Ur = Ur_old*Wr-Ui*Wi;
      Ui = Ur_old*Wi+Ui*Wr; }          /* end of j loop */

 } /* end of stage L */
```

U is successive powers of W

$\log N$ stages @ N per stage

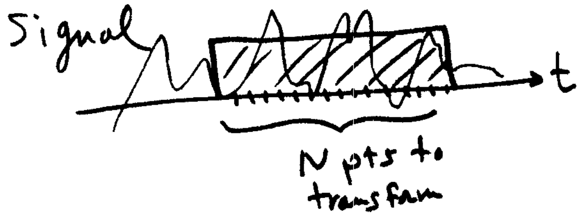
$\Rightarrow O(N \log N)$

(instead of $O(N^2)$)

Notice that this algorithm is IN-PLACE: results are re-written in one array.

The (re-) discovery of this algorithm in 1965 revolutionized signal processing of all kinds. [J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series," Math. Computation, 19, April 1965, pp. 297-301.]

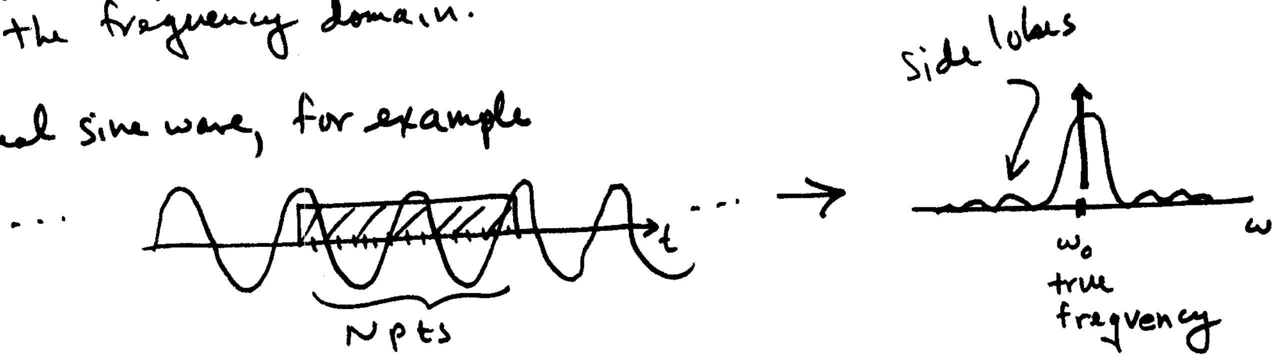
Windowing in using FFT:



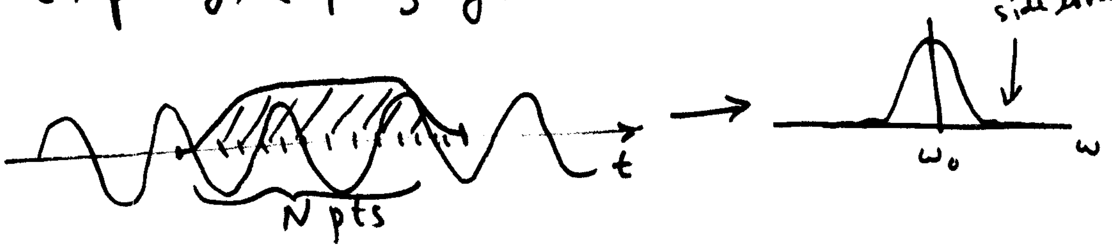
Square window

this abrupt turn-on/turn-off causes a diffraction pattern in the frequency domain.

Ideal sine wave, for example



to reduce this effect, we turn on the window **gradually** (tapering) (apodizing)

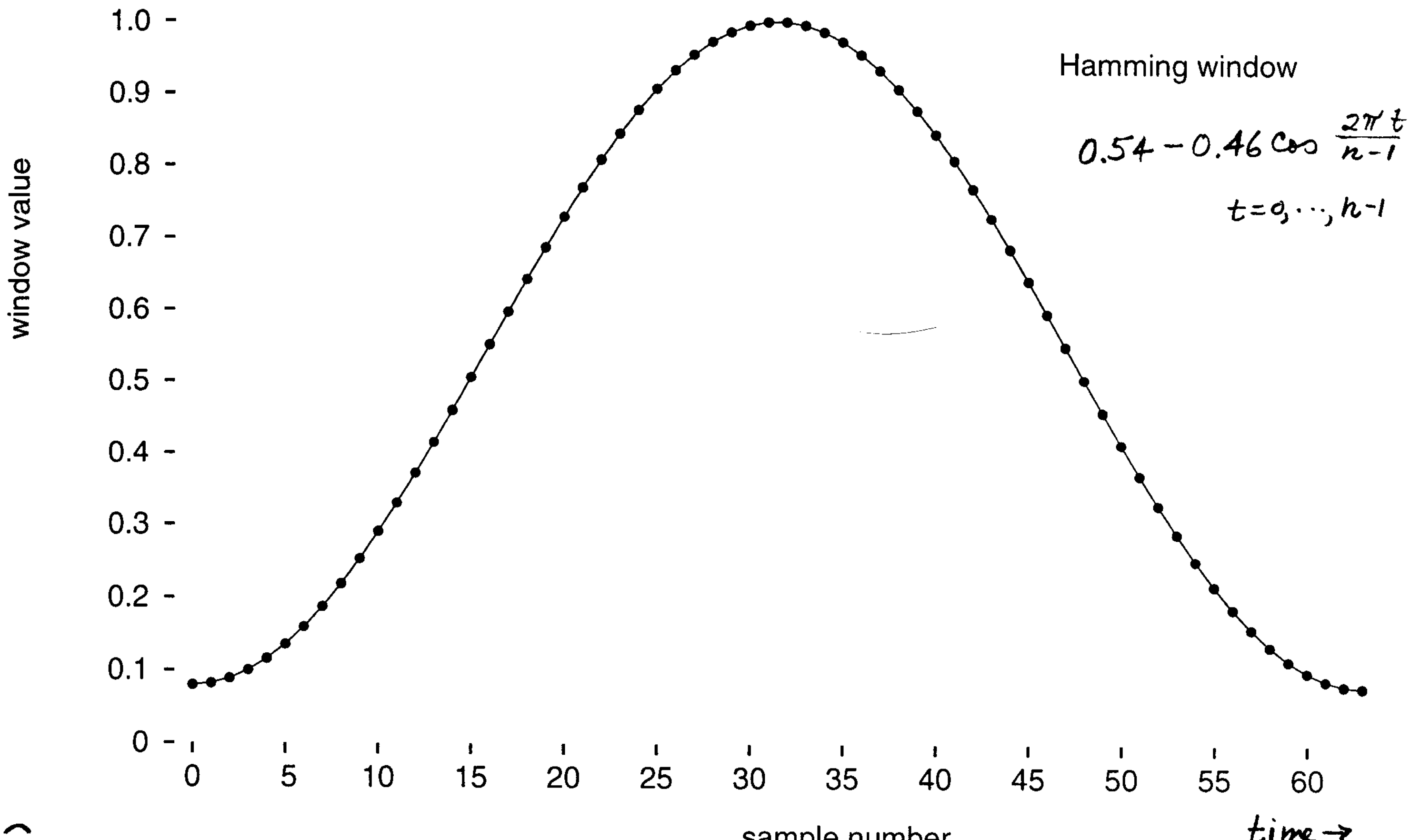


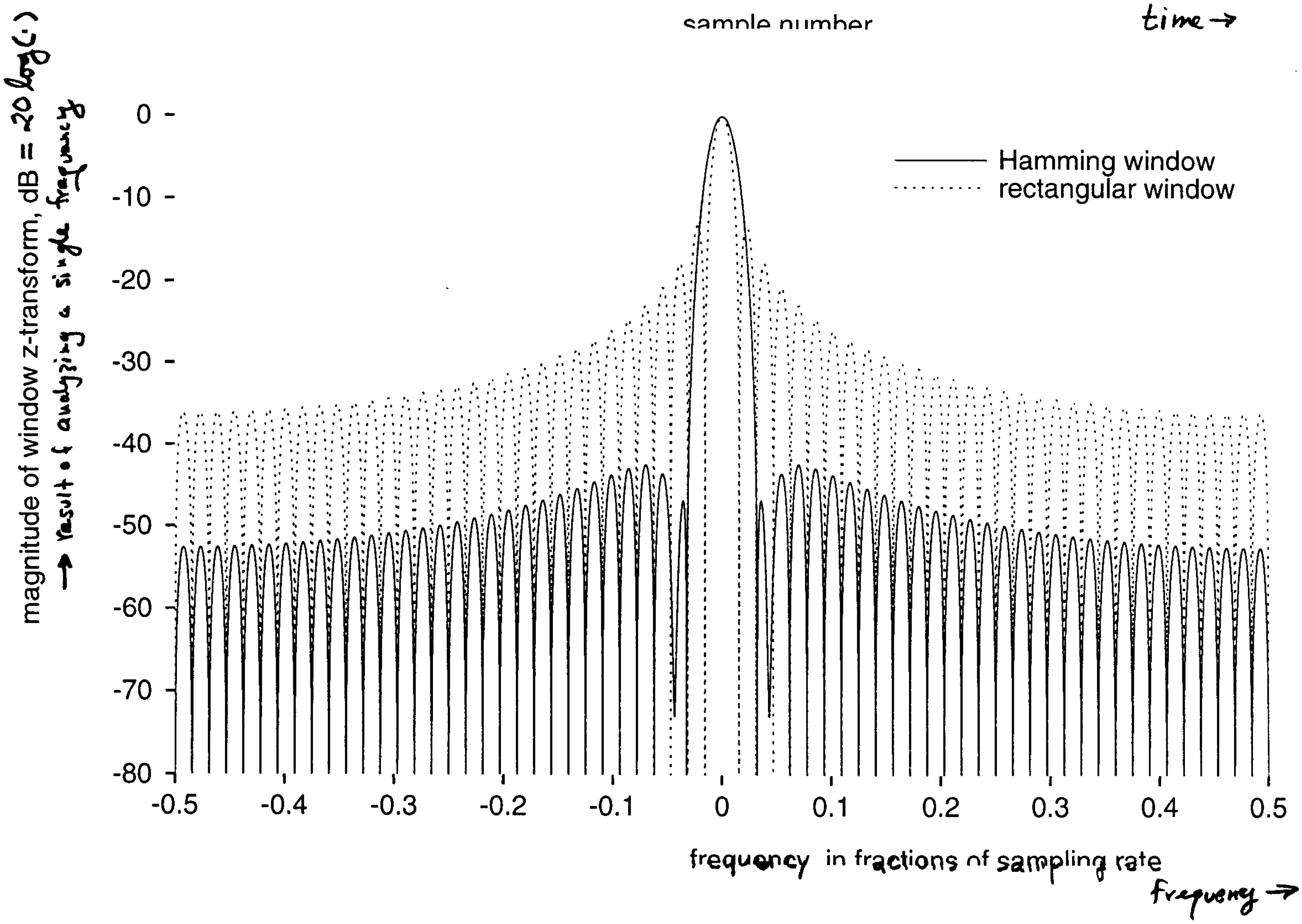
Popular windows

- Hamming raised cosine

- Kaiser weird Bessel fctn. that has close to optimal properties

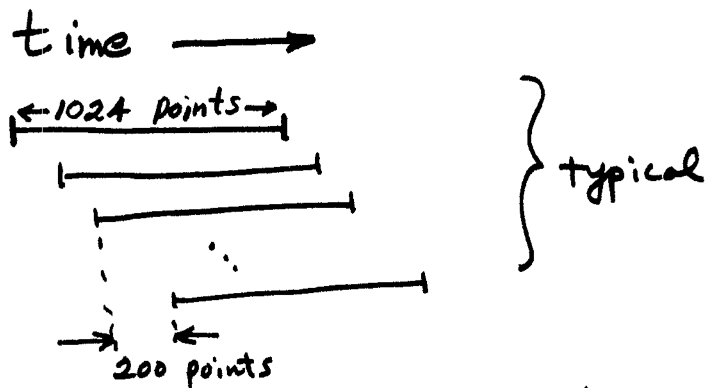
this is not a detail! – makes a big difference.

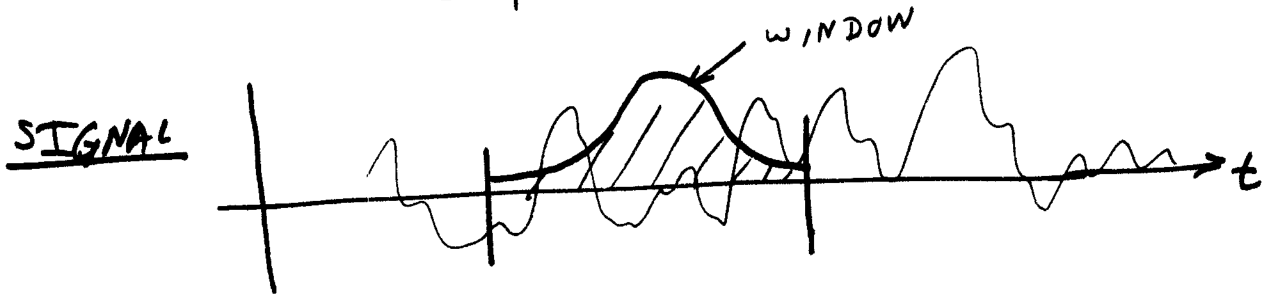

Hamming window
$0.54 - 0.46 \cos \frac{2\pi t}{n-1}$
$t = 0, \cdots, n-1$
window value
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40
45
50
55
60
sample number
time →


magnitude of window z-transform, dB = 20 log(·)
→ result of analyzing a single frequency
Hamming window
rectangular window
0
-10
-20
-30
-40
-50
-60
-70
-80
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
frequency in fractions of sampling rate
frequency →

Tradeoffs in Using FFT:

Sliding windows produce a spectrogram:





I. Narrow window $\Longrightarrow$ good time resolution
bad freq. resolution

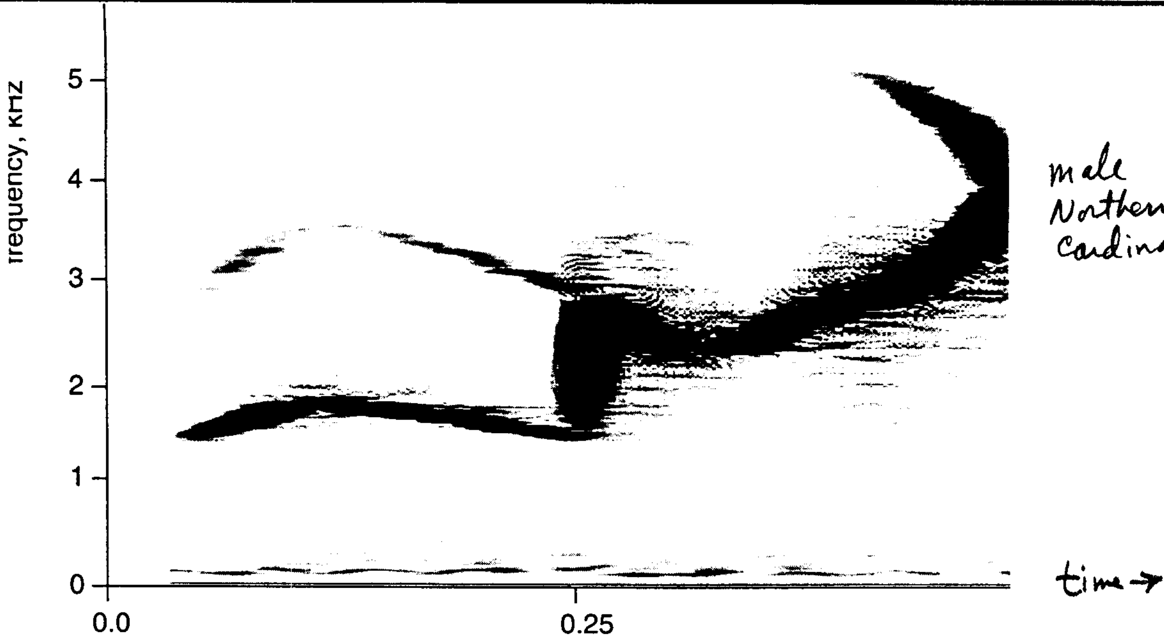
II. Wide window $\Longrightarrow$ bad time resolution
good freq. resolution

So the choice of window length depends on how fast the signal is changing.

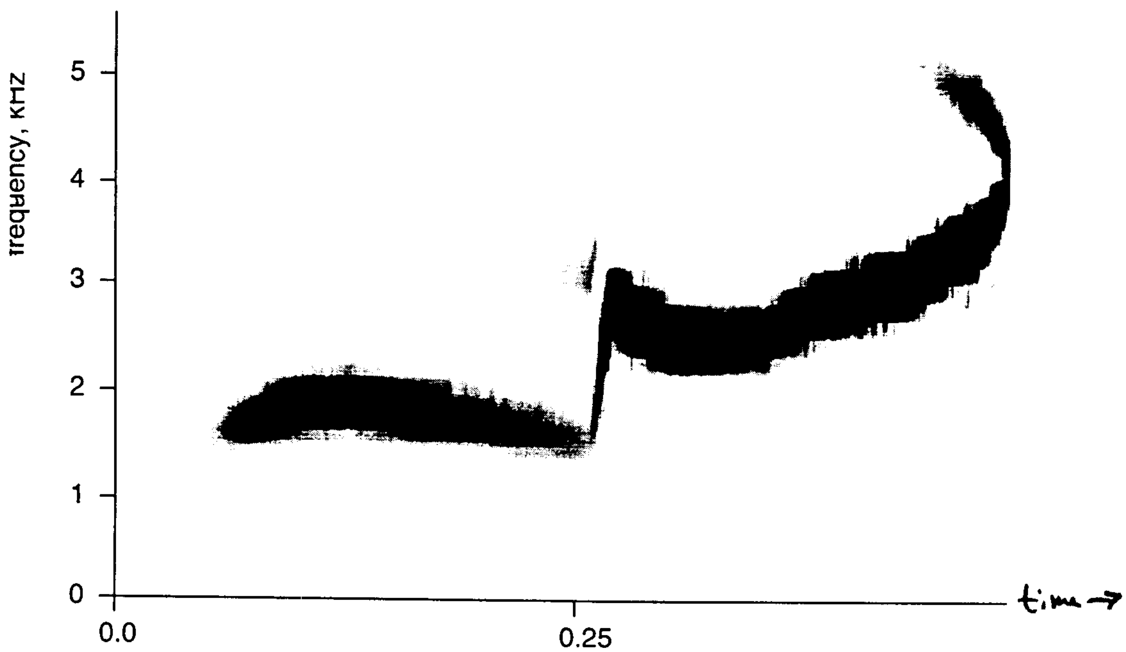
5.2.4

male Northern Cardinal

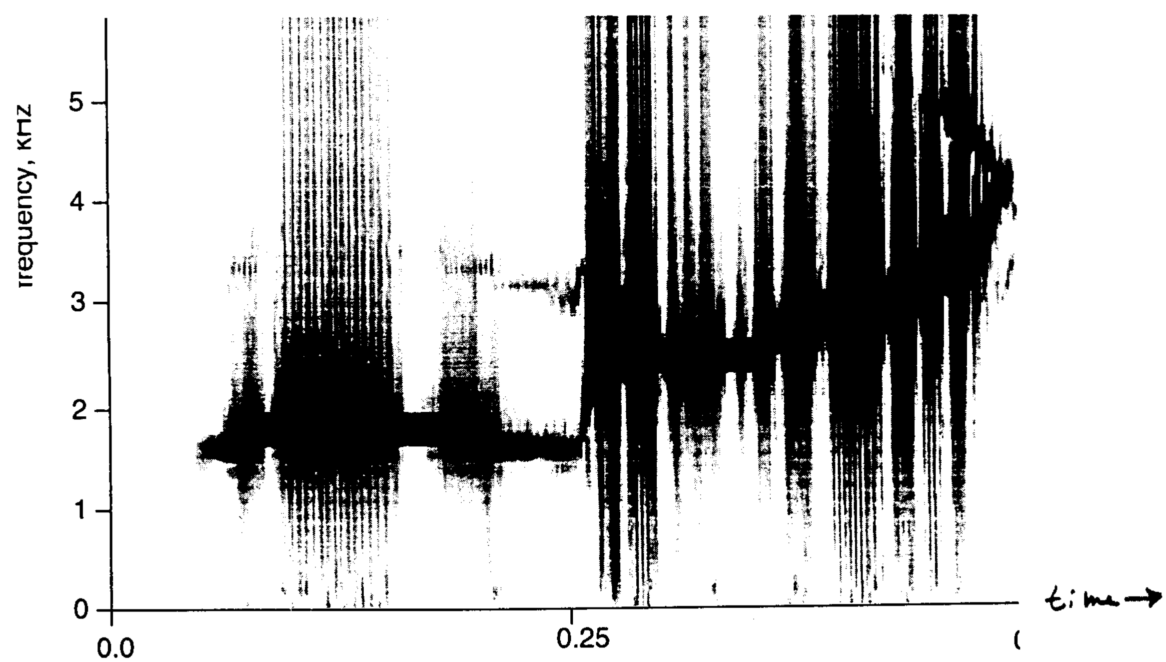
wide window, Hamming

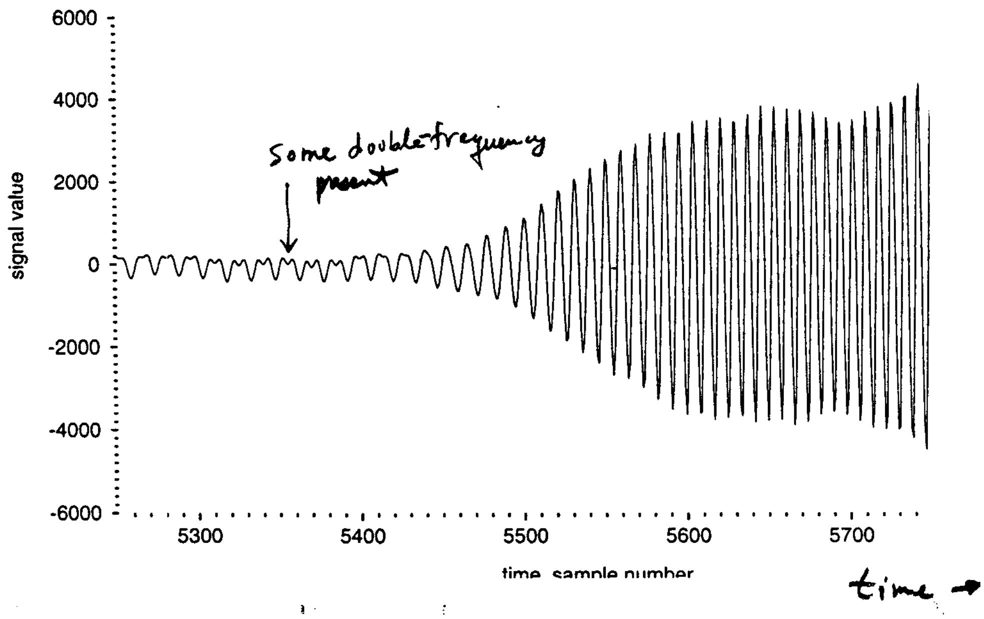


narrow window, Hamming



rectangular (no!) window

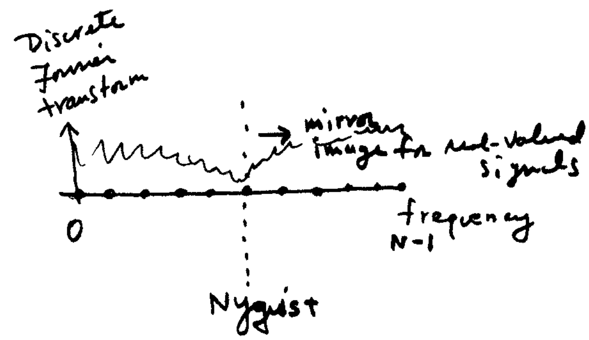


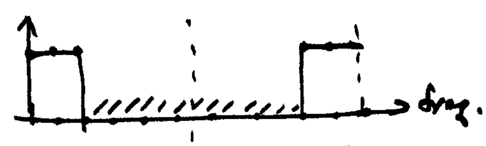


Waveform event — in <u>time</u> <u>domain</u>.
frequency suddenly doubles.

<u>Digital Filtering</u>

Naïve approach: signal $\xrightarrow{\text{FFT}}$

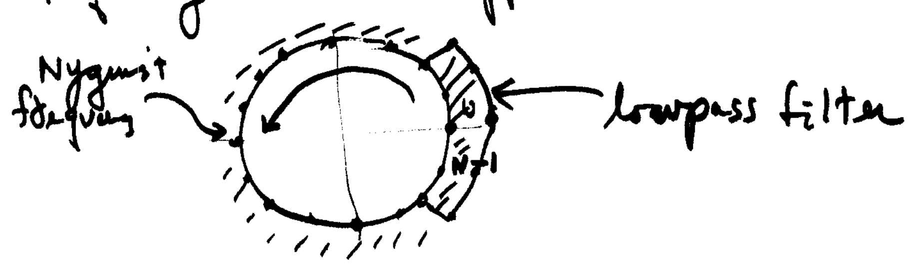




multiply by 1 in "passbands"
0 in "stopbands"

filtered signal $\xleftarrow[\text{FFT}]{\text{inverse}}$

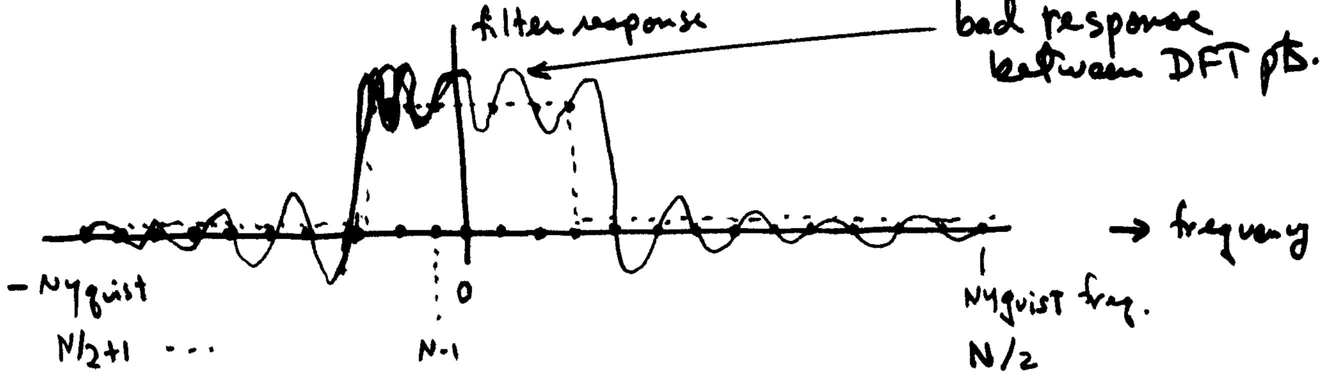
⇒ recall that frequency domain is wrapped around circle:



this is not a good way to design filters — although it works to some extent. One explanation is the same as for the inverse process of windowing in the time domain, except here we are windowing in the frequency domain. Suddenly turning the filter on at a particular frequency causes "diffraction" in the time domain – it's better to taper the filter more gradually.

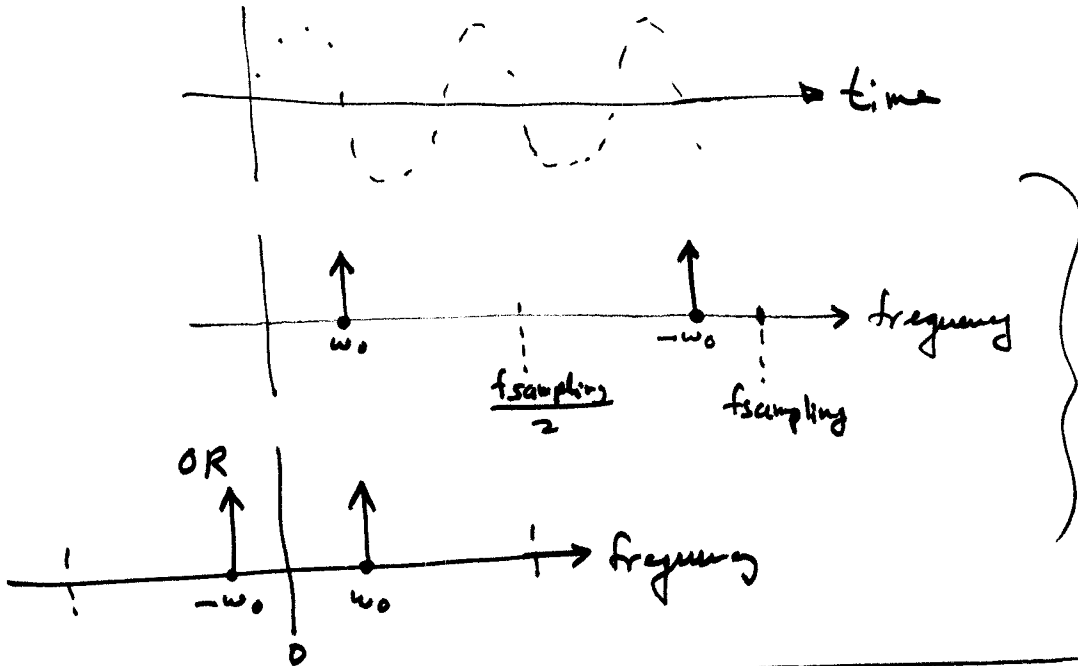
Another explanation: By making the filter 1 in the passbands and 0 in the stopbands, we control the response to different frequencies precisely at the DFT points, $k2\pi/N$. But aren't doing a good job between those points.

Example: lowpass filter



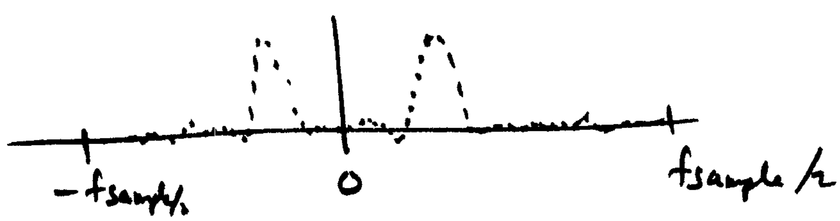
filter design is a highly refined art, using linear programming and other optimization algorithms [STE96].

FFT of sinusoid: $\cos\omega t = \frac{1}{2}\left(e^{j\omega t} + e^{-j\omega t}\right)$



freq. domain is circular

If frequency is not exactly at a DFT point, we get "blurred" version

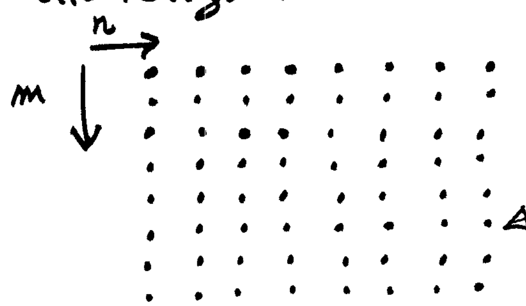


this "blurred" further by <u>finite</u> <u>window</u> in time domain.

the more points in window, the sharper the resolution — analogous to aperture of telescope in resolving stars (actually mathematically equivalent).

Two-dimensional DFT & FFT, for Images

image discretization (assume square for simplicity)



each pixel is usually 3 colors, each a number from $0 \to 255$ (8 bits = 1 byte) "24-bit color"

$N \times N$ pts

$m = 0, \cdots, N-1$

$n = 0, \cdots, N-1$

y_{mn}

Fourier Representation:

$$y_{mn} = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} Y_{kl}\, \underbrace{e^{j\left(\frac{2\pi}{N}k\right)m}}_{\text{m-direction frequency}}\, \underbrace{e^{j\left(\frac{2\pi}{N}l\right)n}}_{\text{n-direction frequency}}$$

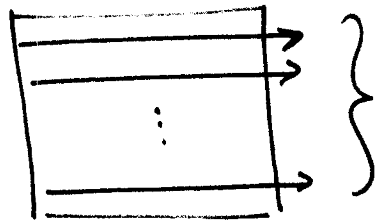
Y_{kl}: content at frequency k, l in frequency plane

Forward transform:

$$Y_{kl} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} y_{mn}\, e^{-j\left(\frac{2\pi}{N}k\right)m}\, e^{-j\left(\frac{2\pi l}{N}\right)n}$$

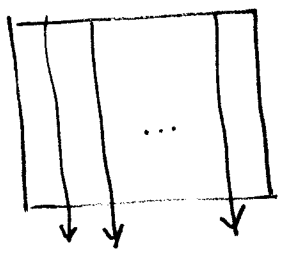
$$= \underbrace{\sum_{m=0}^{N-1} e^{-j\left(\frac{2\pi}{N}k\right)m} \underbrace{\left[\sum_{n=0}^{N-1} y_{mn}\, e^{-j\left(\frac{2\pi l}{N}\right)n}\right]}_{\text{1-dim. DFT of row } m}}_{\text{1-dim. DFT of column } l}$$

So to get 2-dim. DFT using FFT algorithm:



N FFT's of rows

followed by



N FFT's of columns that result

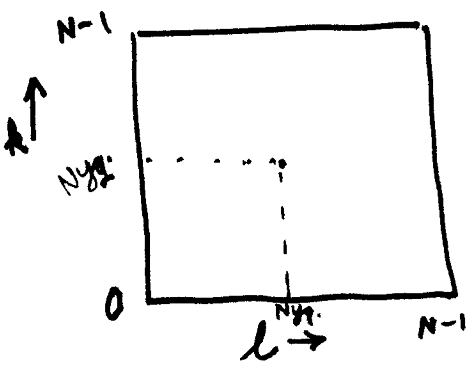
takes # of steps $= N\cdot(N\log N) + N\cdot(N\log N)$

$= 2N^2\log N$

$= N^2 \log N^2$

$= Q\log Q$, where $Q = N^2 = \#$ pts.

Frequency Domain



often drawn with 0-freq. pt. in center