C++ Real-Time Audio Programming with Bela

Dr Andrew McPherson

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London

Founder and Director, Bela

Course topics

Programming topics

Working in real time
Buffers and arrays
Parameter control
Classes and objects
Analog and digital I/O
Filtering
Timing in real time
Circular buffers
State machines
MIDI
Block-based processing
Threads
Fixed point arithmetic
ARM assembly language

Music/audio topics

Oscillators
Samples
Wavetables
Control voltages
Gates and triggers
Filters
Metronomes and clocks
Delays and delay-based effects
Envelopes
ADSR
MIDI
Additive synthesis
Phase vocoders
Impulse reverb

Today

Lecture 19: Phase vocoder, part 2

What you'll learn today:

Working in the frequency domain
Frequency detection from phase
Window functions

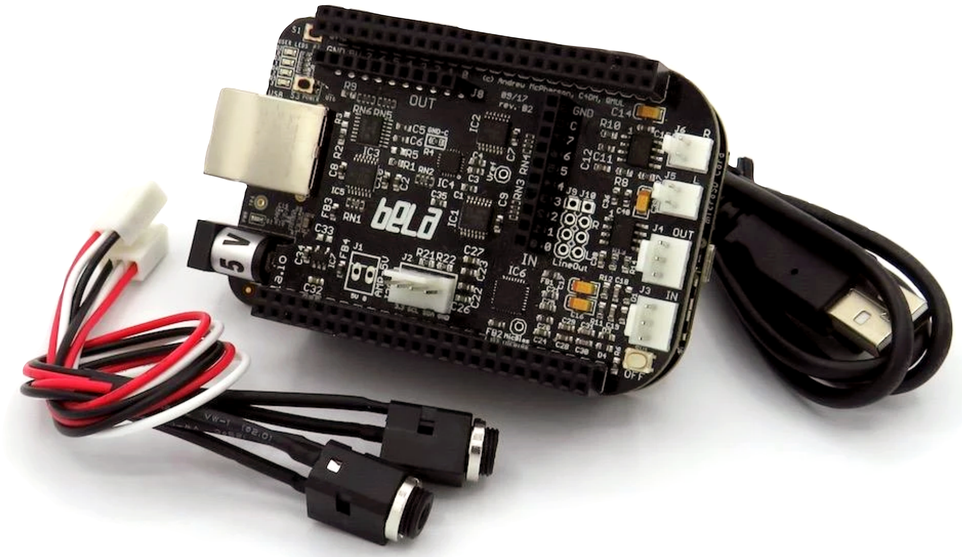
What you'll make today:

An FFT-based frequency detector

Companion materials:

github.com/BelaPlatform/bela-online-course

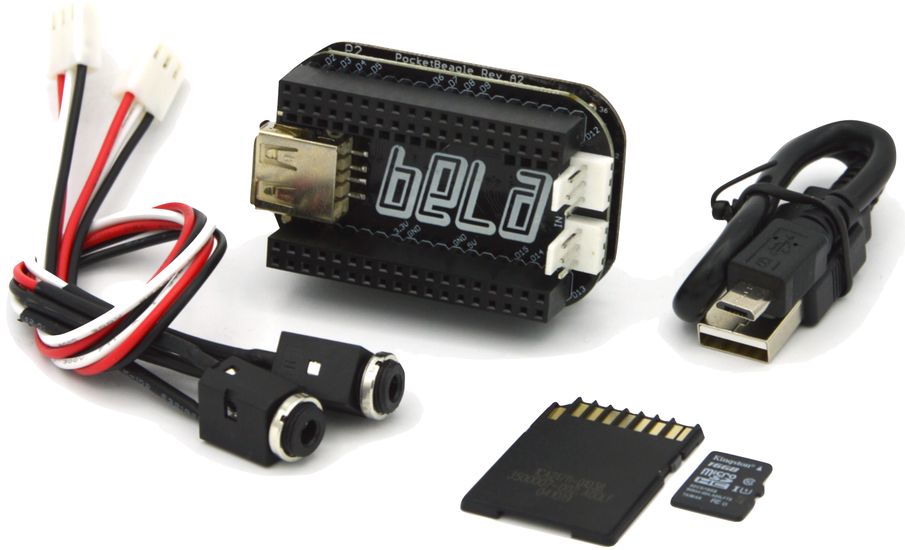
What you'll need



Bela Starter Kit

or

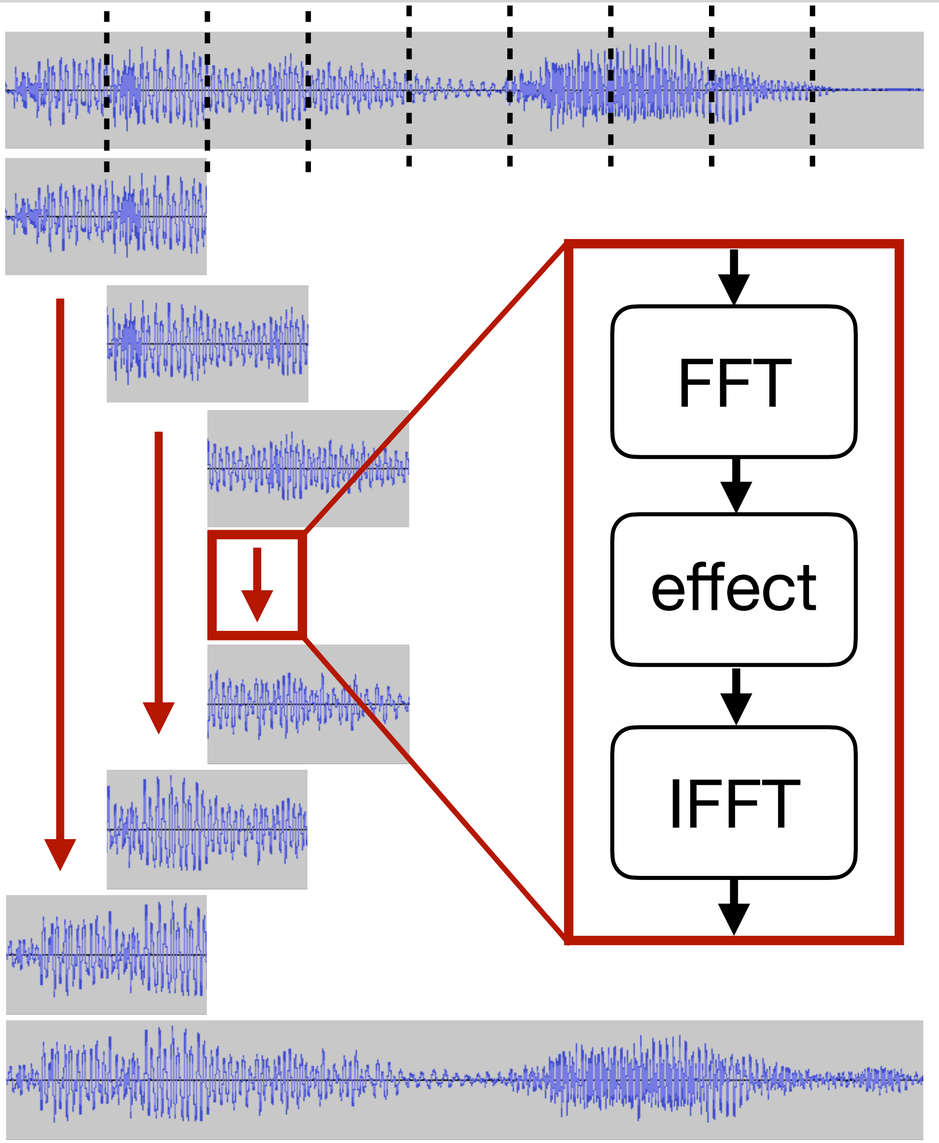
[shop.bela.io]



Bela Mini Starter Kit

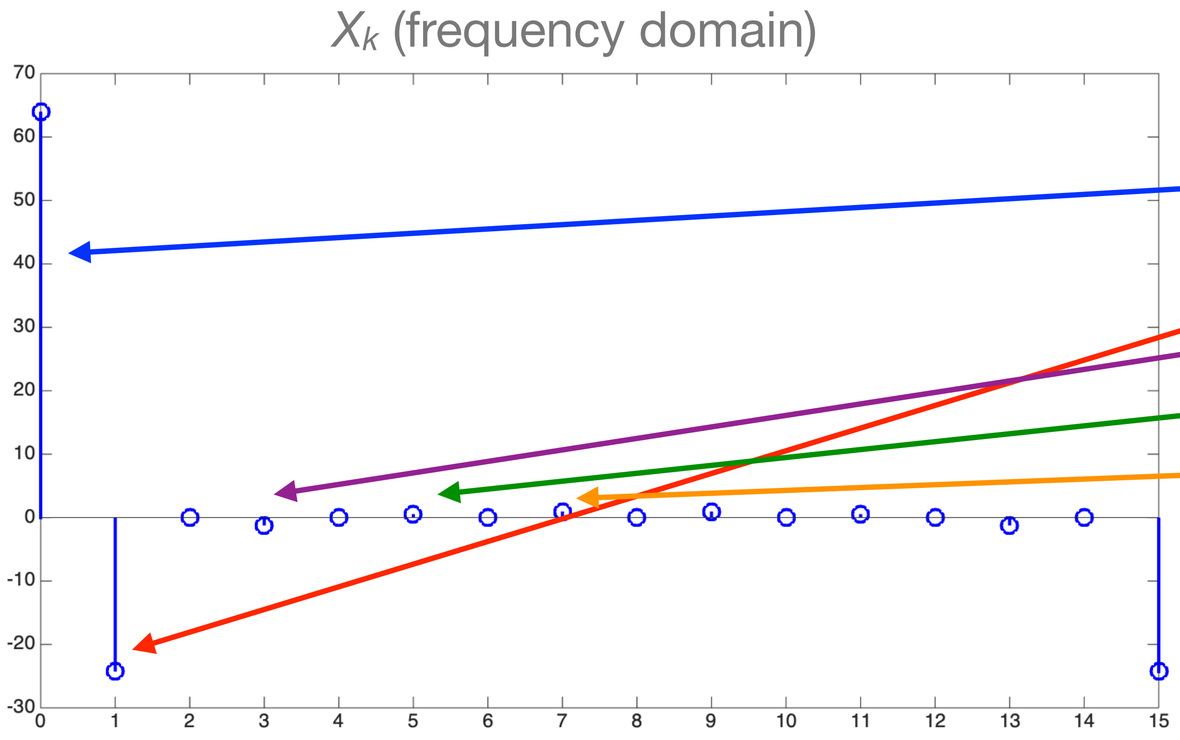
Overlap-Add

- A standard approach for block-based processing with overlapping blocks is called overlap-add:
  1. Isolate a block of length M using a windowing function
  2. Take an FFT of length N ≥ M of the segment
     - If N > M, zero-pad the block (add zeros to end of the window)
  3. Do something interesting to frequency-domain data
  4. Take an IFFT to get a new time domain segment
  5. Add the segment to an output buffer which also contains the preceding segments
     - As we'll see, we can't write the segment directly to the audio output
  6. Advance by the hop size (H) to the next frame and repeat
     - In real time, count samples until H more have arrived
     - Hop size H is less than window size M, hence the **overlap**



The Fast Fourier Transform

- Most common use of block-based processing: the Fast Fourier Transform (FFT)
  - Efficient computational algorithm for calculating the Discrete Fourier Transform (DFT)
  - The DFT is a mathematical formula for calculating the frequency content of a signal
- Any discrete-time signal of N points can be expressed as the sum of N sinusoids:



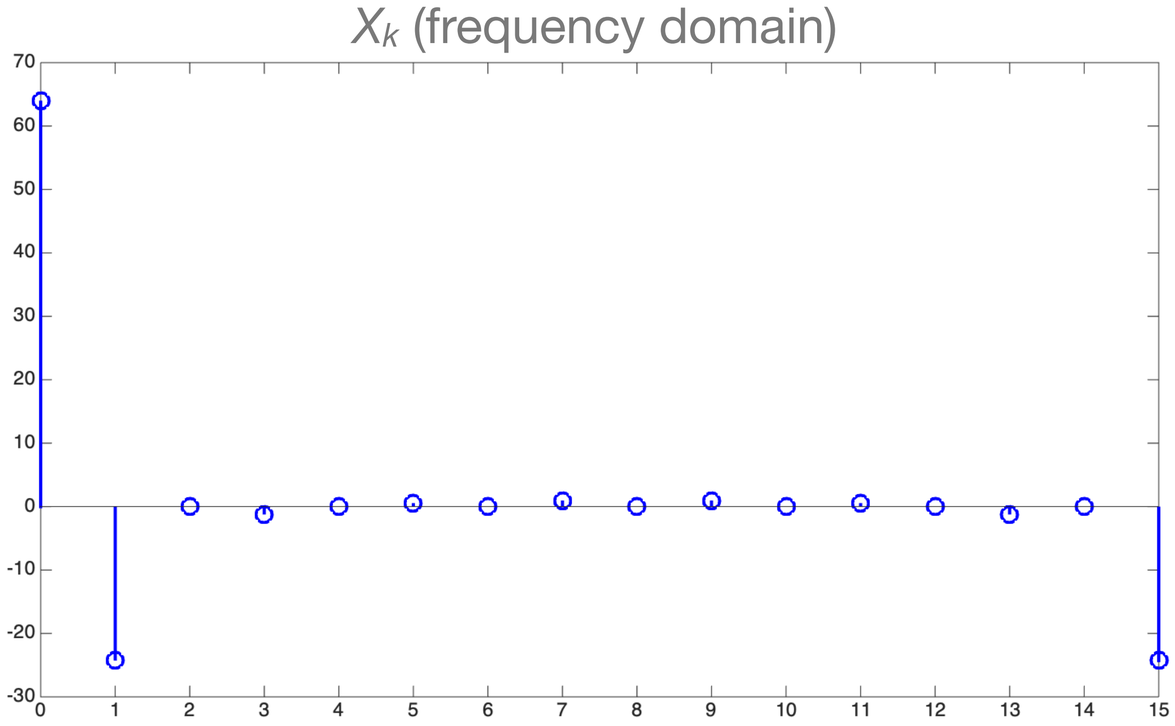
Frequency	Amplitude	Phase
0	4.00	0
π/8	1.64	-.875*π
3π/8	0.20	-.625*π
5π/8	0.09	-.375*π
7π/8	0.65	-.125*π

(other components are 0 or redundant)

The DFT calculates the amplitudes and phases of each frequency component

The Discrete Fourier Transform

- The DFT calculates the amplitudes and phases of each sinusoidal component
  - The frequencies are linearly spaced between 0 and 2π: $\frac{2\pi k}{N}$ for $0 \leq k < N$
  - Each amplitude/phase measurement is called a bin
  - Collectively, the N bins are called the frequency domain representation of the signal



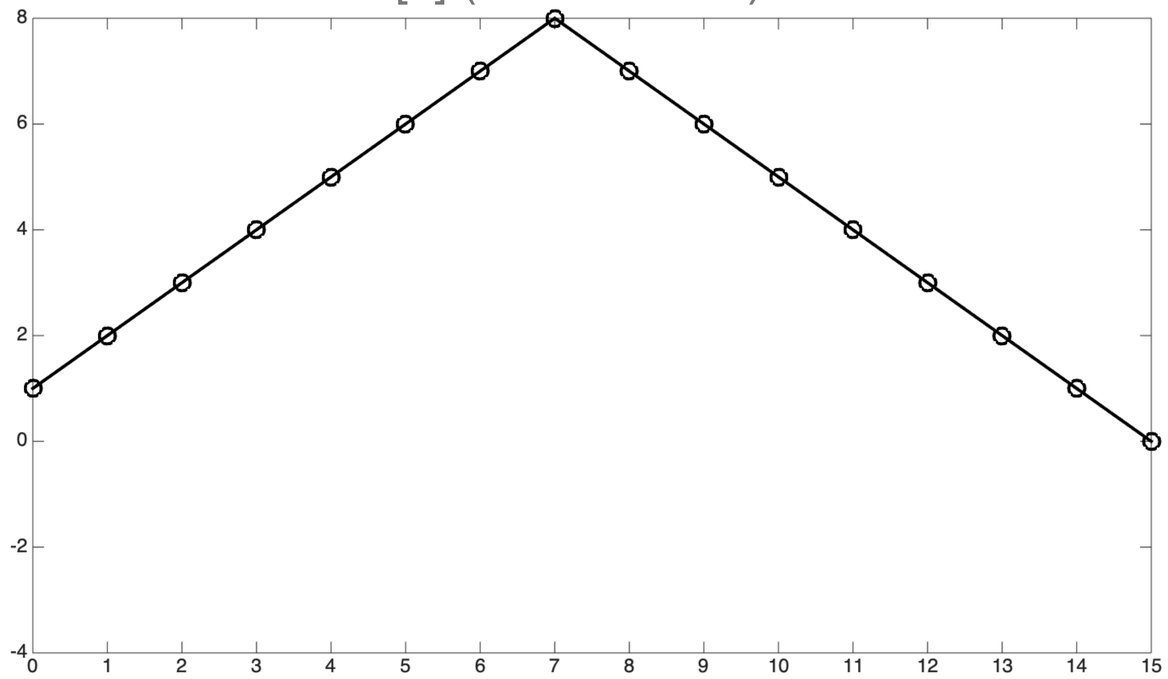
Frequency	Amplitude	Phase
0	4.00	0
π/8	1.64	-.875*π
3π/8	0.20	-.625*π
5π/8	0.09	-.375*π
7π/8	0.65	-.125*π

(other components are 0 or redundant)

The Discrete Fourier Transform

- Key point: N time samples ↔ N frequency samples
  - Two different mathematical perspectives on the same signal
  - The Inverse Discrete Fourier Transform (IDFT) converts from frequency domain to time domain
  - The DFT+IDFT is an exact reconstruction as long as signal length $M \leq$ DFT length N

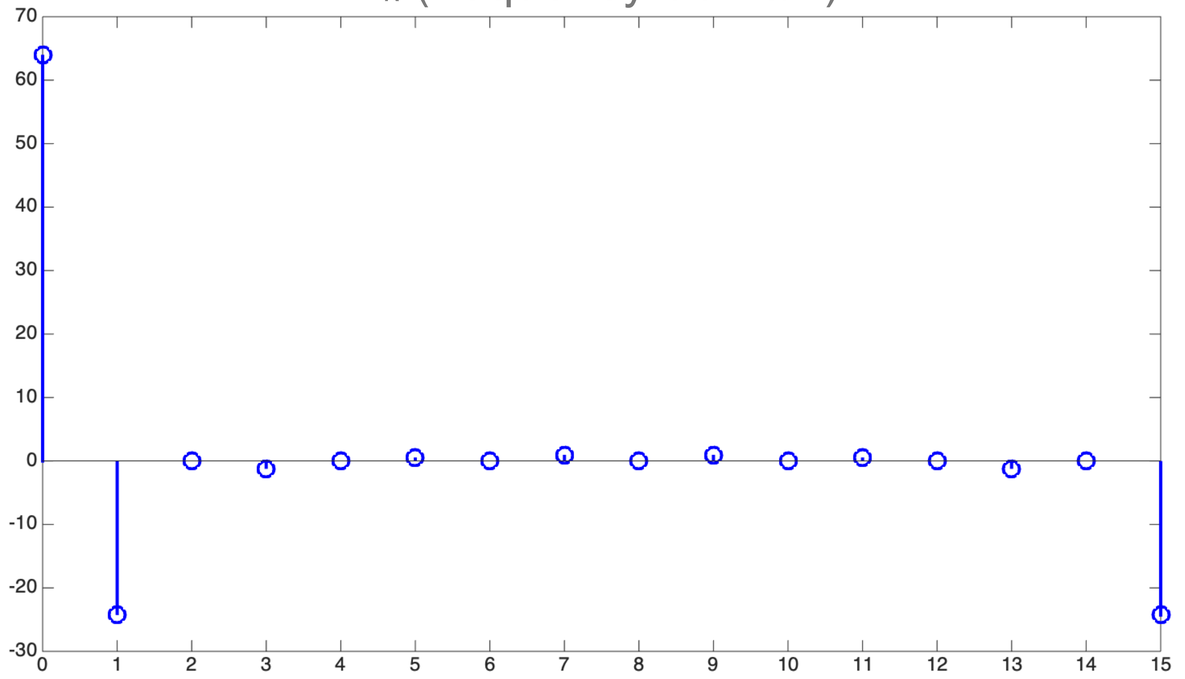
$x[n]$ (time domain)



DFT

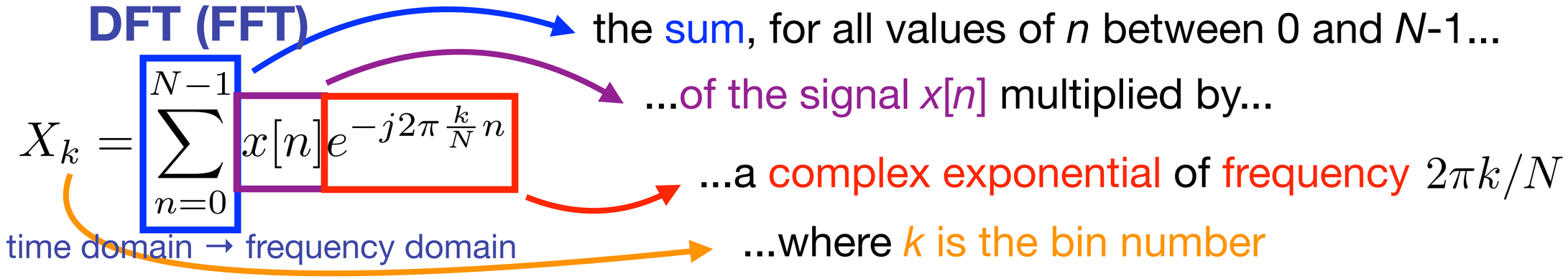
IDFT

X_k (frequency domain)



The Discrete Fourier Transform

- The mathematical basis of the Discrete Fourier Transform:



- N is the size of the DFT
  - For the FFT, N is almost always a power of 2 (the Cooley-Tukey algorithm)
- Each time sample *x*[*n*] is a real number
- Each frequency bin X_k is a complex number: $a + bj$ where $j = \sqrt{-1}$
  - Magnitude of the bin is given by: $\sqrt{a^2 + b^2}$
    - We usually use magnitude to plot a spectrum or spectrogram of a signal
  - Phase of the bin is given by: $\arctan(b/a)$

The Discrete Fourier Transform

- The mathematical basis of the Discrete Fourier Transform:

DFT (FFT)

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n}$$

time domain → frequency domain

IDFT (IFFT)

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{+j2\pi \frac{k}{N} n}$$

frequency domain → time domain

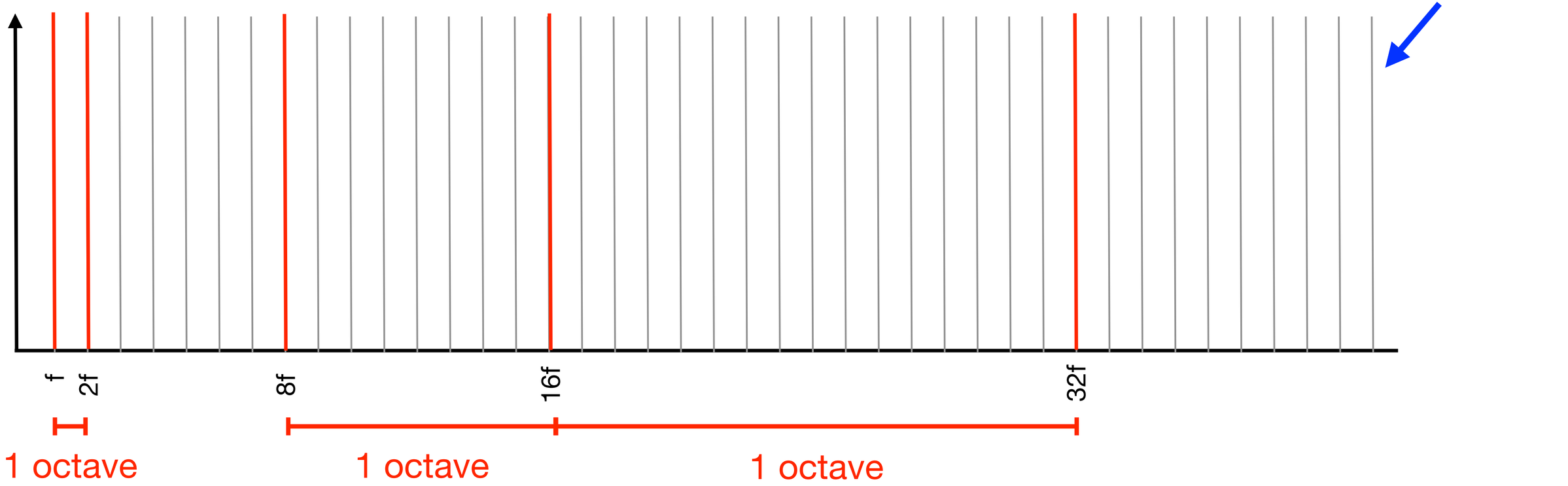
- N is the size of the DFT
  - For the FFT, N is almost always a power of 2 (the Cooley-Tukey algorithm)
- Each time sample *x*[*n*] is a real number
- Each frequency bin X_k is a complex number: $a + bj$ where $j = \sqrt{-1}$
  - Magnitude of the bin is given by: $\sqrt{a^2 + b^2}$
    - We usually use magnitude to plot a spectrum or spectrogram of a signal
  - Phase of the bin is given by: $\arctan(b/a)$

DFT magnitude and phase

- Each frequency bin X_k is a complex number: $a + bj$ where $j = \sqrt{-1}$
  - Magnitude of the bin is given by: $\sqrt{a^2 + b^2}$
    - We usually use magnitude to plot a spectrum or spectrogram of a signal
  - Phase of the bin is given by: $\arctan(b/a)$
- We can also turn the calculation around:
  - Given a magnitude M and a phase ϕ, calculate real and imaginary components
  - Real component: $a = M\cos(\phi)$ Imaginary component: $b = M\sin(\phi)$
- Remember, the frequency resolution depends on the DFT size
  - Bin frequencies are spaced 2π/N apart, where 2π corresponds to the sample rate
  - For example, N=1024, f_s = 44.1kHz: bin spacing is 43.1Hz

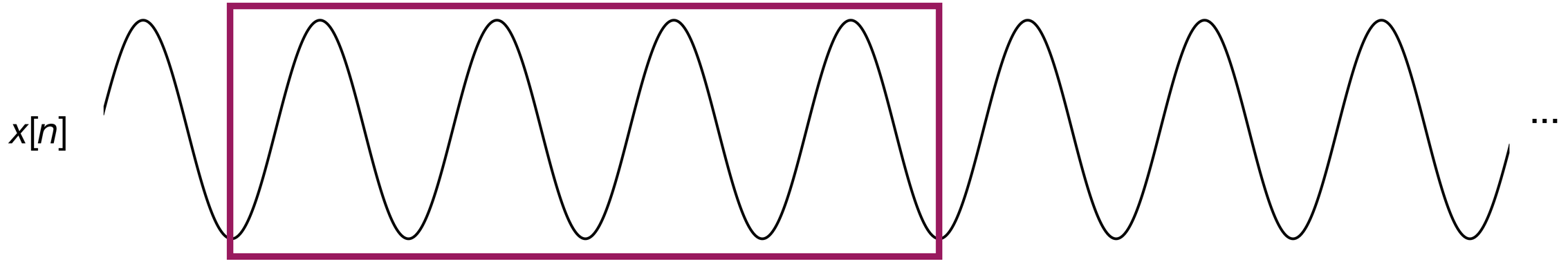
DFT frequency spacing

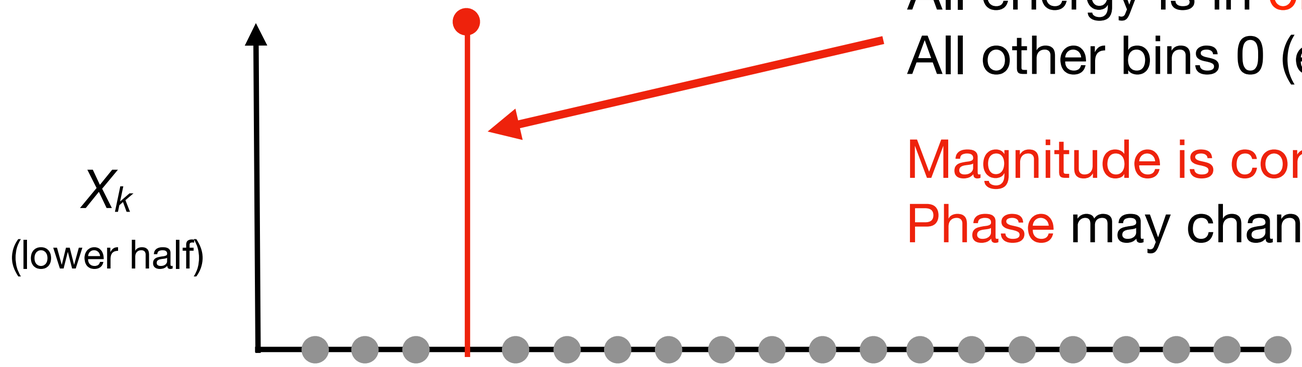
- Discrete Fourier Transform bins are spaced linearly in frequency
  - By contrast, musical intervals are defined as ratios in frequency (e.g. 1 octave = 2x)
  - This means that pitch resolution of the DFT is better for higher bins
    - This can be a headache for many musical applications



Sine wave in the frequency domain

- What does the FFT of a single sine wave look like?
  - Let's suppose its frequency exactly matches a bin: $2\pi k/N$ for some k
  - This means that an integer number of oscillations fit in the window



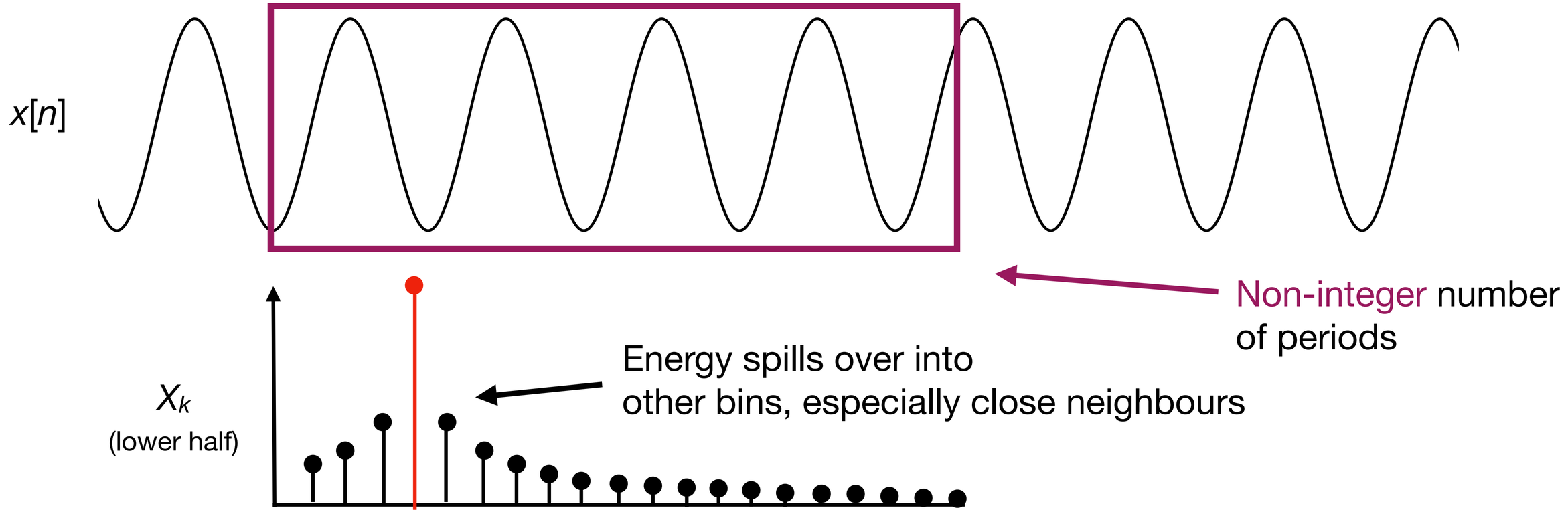


All energy is in one frequency bin
All other bins 0 (except mirror image above Nyquist)

Magnitude is constant no matter where the window starts
Phase may change depending on window location

Sine wave in the frequency domain

- What if the frequency doesn't exactly match any bin? We'll see two effects:
  1. Energy smears out across all the other frequency bins (we'll see why shortly)
  2. Phase is likely to be different on each successive hop
     - If we're clever, we can reconstruct the exact frequency based on the phase changes!



Sine wave in the frequency domain: task

- Task: run the example project fft-sine
  - This project takes the FFT of a sine wave and displays the spectrum in the GUI
    - Block-based processing in `render.cpp` follows Lecture 17
  - Adjust the frequency of the oscillator in the GUI: what happens to the spectrum?
    - By default, the sine is exactly on an FFT bin frequency
    - After you adjust it, the frequency will fall between FFT bins
  - You can change the FFT size at the top of `render.cpp`

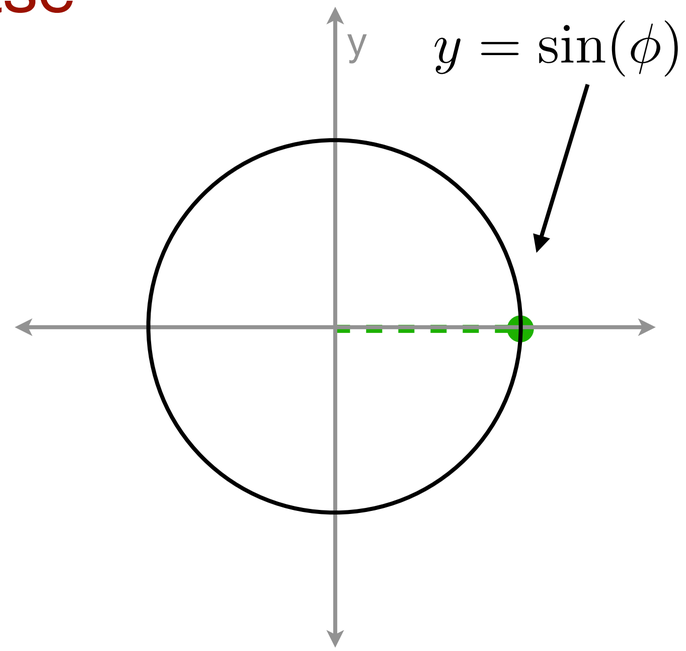
Reconstructing exact frequency

- We can estimate the exact frequency of a sine wave from the FFT
  - Even though the FFT only has bins at fixed, regularly spaced frequencies!
- To do this, we will need two hops
  - Two snapshots of the signal taken a known length of time apart
- Key point: frequency is the derivative of phase
  - $\omega(t) = \frac{d\phi}{dt}$ or in discrete time: $\omega[n] = \frac{\Delta\phi}{\Delta n}$
  - We can write out the first-order difference:

$$\omega[n] = \frac{\phi[n] - \phi[n-1]}{(n) - (n-1)} = \phi[n] - \phi[n-1]$$

  - Or generalising over a hop of *H* samples:

$$\omega[n] = \frac{\phi[n] - \phi[n-H]}{(n) - (n-H)} = \frac{\phi[n] - \phi[n-H]}{H}$$



Reconstructing exact frequency

- Calculating frequency from phase in discrete time:

$$\omega[n] = \frac{\phi[n] - \phi[n-H]}{(n) - (n-H)} = \frac{\phi[n] - \phi[n-H]}{H}$$

  - In discrete time, frequency ω is normalised between 0 and 2π;
  - Assumes unwrapped phase (i.e. always increasing, not constrained 0-2π)
- We can invert this formula to calculate phase shift based on frequency:

$$\phi[n] - \phi[n-H] = H\omega[n]$$

  - For a DFT of size *N*, we know bin *k* has a frequency of $2\pi k/N$
  - For a sine wave of exactly the bin frequency, this is the expected phase shift over 1 hop:

$$\phi[n] - \phi[n-H] = \frac{2\pi k H}{N}$$

  - By subtracting off this expected phase shift, the remainder will tell us whether the actual frequency is higher or lower than the bin frequency

Reconstructing exact frequency

- First we calculate the phase remainder: $\phi_r[n] = (\phi[n] - \phi[n-H]) - \frac{2\pi k H}{N}$

  measured **phase values** from bin *k* on **two successive hops**

  centre frequency of bin *k* multiplied by the **hop size**

- Then use remainder to calculate the deviation from the bin centre frequency:
  - We should wrap the phase to be between -π and π (i.e. the principal argument)

$$\omega[n] - \omega_k = \frac{\text{wrap}(\phi_r[n])}{H}$$

centre frequency of bin *k* (= 2π*k*/*N*)

- Finally, we can rearrange to get the exact frequency of our sine wave:

$$\omega[n] = \frac{\text{wrap}(\phi_r[n])}{H} + \omega_k$$

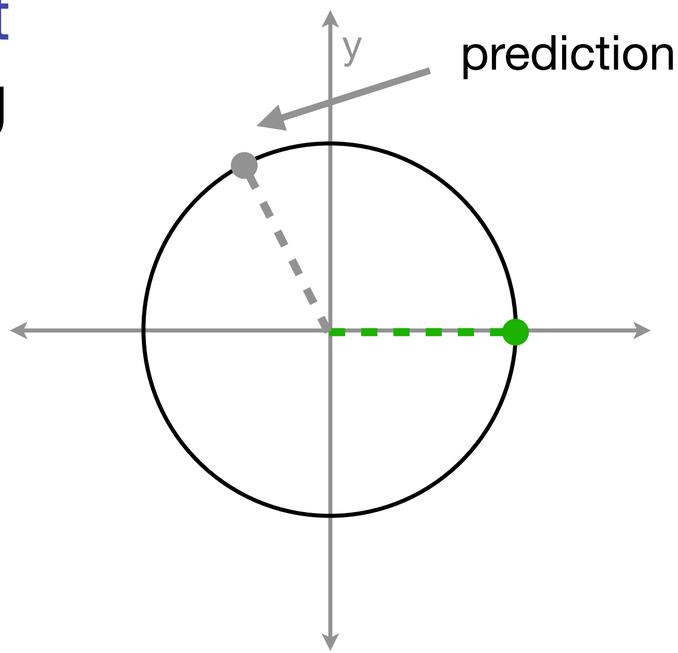
  - We could also calculate in terms of (fractional) FFT bins
    - Use the fact that bins are spaced 2π/*N* apart in frequency

fractional bin number of the sine wave

$$b[n] - k = \frac{\text{wrap}(\phi_r[n])N}{2\pi H} \rightarrow b[n] = \frac{\text{wrap}(\phi_r[n])N}{2\pi H} + k$$

Reconstructing exact frequency: code

- Effectively, we are predicting how far the phase ought to advance from one hop to the next, then comparing the actual measured values
  - Phase advancing farther means a higher frequency (compared to the bin frequency)
  - Phase advancing less far means a lower frequency
- What do we need to implement this in code?
  - A global (or static) variable to hold the phase of each bin
    - Saving each bin implies an array
    - This is very similar to saving previous values of $x[n]$ or $y[n]$ for a filter (see Lecture 8)
  - A function to wrap the phase to the range -π to π
  - Implement the calculations inside of our `process_fft()` function
    - Write a `for()` loop iterating across each bin



Reconstructing exact frequency: task

- Task: in fft-sine, write the code to work out the exact frequency of each bin
  - Implement these formulas:

$$\phi_r[n] = (\phi[n] - \phi[n-H]) - \frac{2\pi k H}{N}$$

bin phase this hop; bin phase last hop; FFT size; bin number; hop size

$$\omega[n] = \frac{\text{wrap}(\phi_r[n])}{H} + \omega_k$$

`wrapPhase()` written for you; bin centre frequency ($2\pi k/N$)

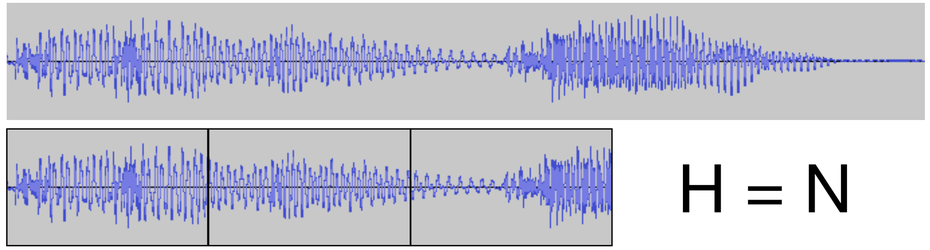
  - Remember you will need a global or static array to hold the previous phases
  - The GUI will display the frequency of the bin with highest magnitude
  - However, the same formula can apply to every bin regardless of magnitude
    - In practice, we only look at the first half of bins because the DFT is conjugate symmetric

Choosing an FFT size

- The previous formulas gave us one frequency component for each bin
  - An FFT of size *N* has (*N*/2+1) unique bins after accounting for symmetry
  - This tells us the maximum number of frequency components we can represent
- Hence, the larger the FFT, the greater the frequency resolution
  - Bins are spaced more closely together in frequency: $2\pi k/N$
  - Easier to resolve two closely-spaced frequency components of a signal
- What's the tradeoff? Poor time resolution
- N time-domain samples ↔ N frequency-domain bins
  - Resolving short transient events within a long time window is difficult
  - Frequency-domain processing tends to smear transients, distorting their shape
  - Another tradeoff to large FFTs: longer latency
  - These are fundamental mathematical limitations

Choosing a hop size

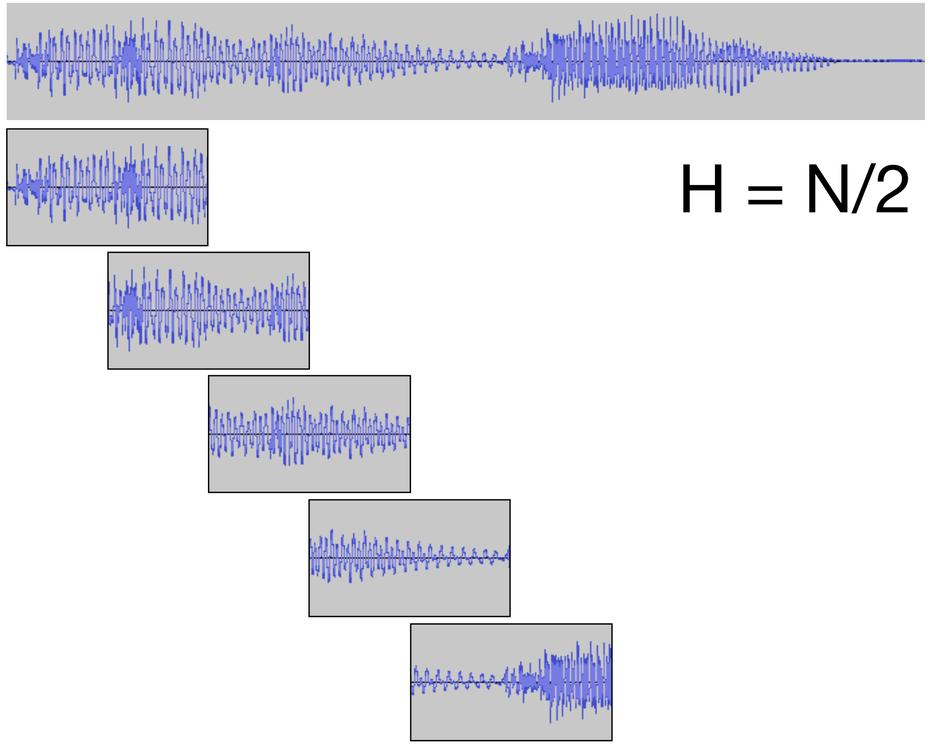
- How often should we take an FFT?
  - Assuming we want to reconstruct the signal, the hop size H must be no more than the FFT size N
- Advantages of smaller hop sizes
  - We get more frequent time snapshots of the signal
  - Phase of each bin advances less from one hop to the next, improving frequency estimation
  - Some types of window need a hop size of N/2, N/3 or even less to reconstruct the signal
  - Lower latency for phase vocoder effects
- Disadvantages of smaller hop sizes
  - More computationally expensive
  - Hop sizes that aren't an integer division of the FFT size can lead to artefacts on reconstruction



H = N

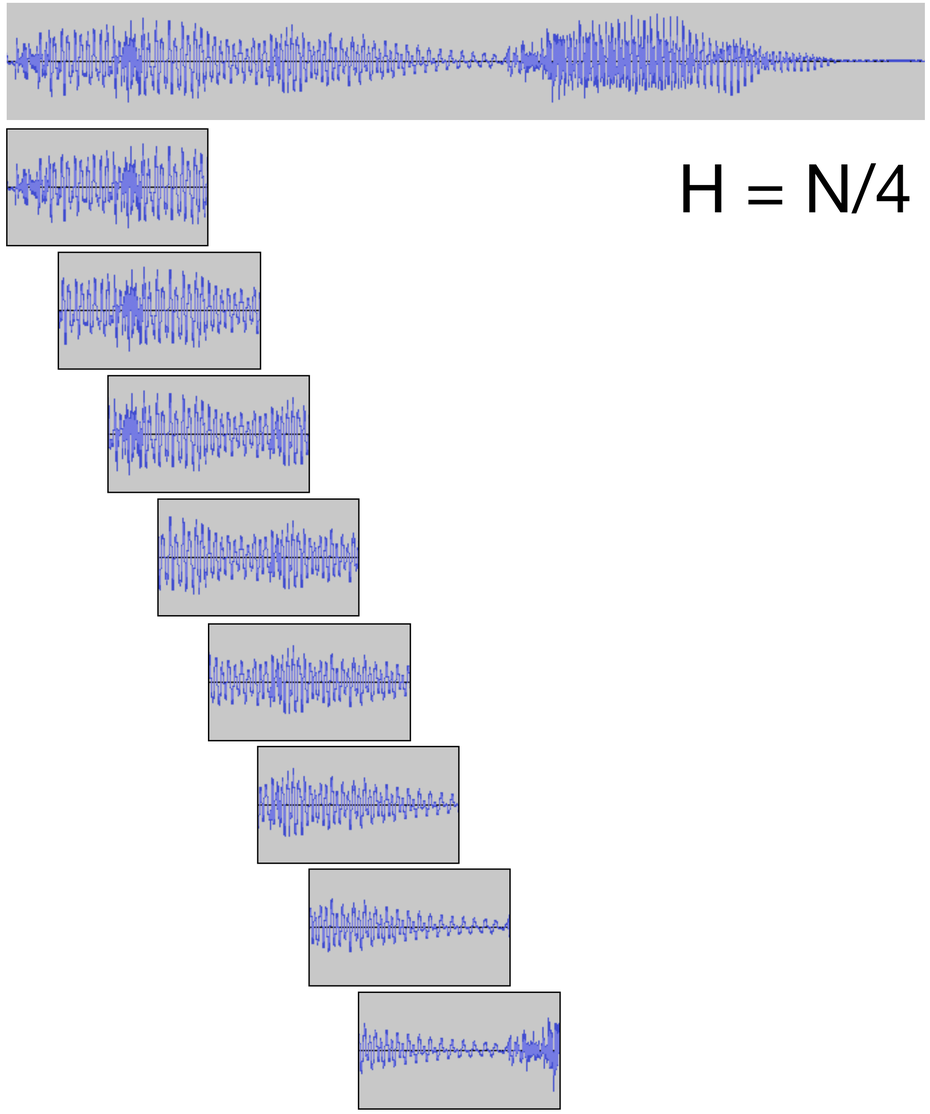
Choosing a hop size

- How often should we take an FFT?
  - Assuming we want to reconstruct the signal, the hop size H must be no more than the FFT size N
- Advantages of smaller hop sizes
  - We get more frequent time snapshots of the signal
  - Phase of each bin advances less from one hop to the next, improving frequency estimation
  - Some types of window need a hop size of N/2, N/3 or even less to reconstruct the signal
  - Lower latency for phase vocoder effects
- Disadvantages of smaller hop sizes
  - More computationally expensive
  - Hop sizes that aren't an integer division of the FFT size can lead to artefacts on reconstruction



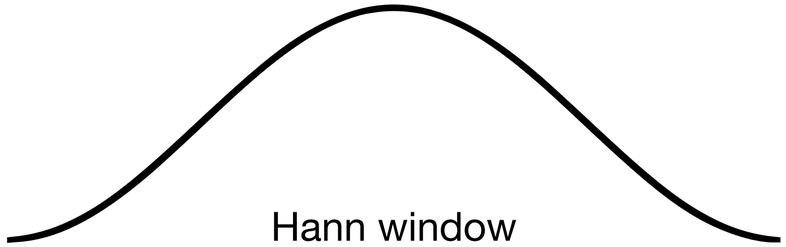
Choosing a hop size

- How often should we take an FFT?
  - Assuming we want to reconstruct the signal, the hop size H must be no more than the FFT size N
- Advantages of smaller hop sizes
  - We get more frequent time snapshots of the signal
  - Phase of each bin advances less from one hop to the next, improving frequency estimation
  - Some types of window need a hop size of N/2, N/3 or even less to reconstruct the signal
  - Lower latency for phase vocoder effects
- Disadvantages of smaller hop sizes
  - More computationally expensive
  - Hop sizes that aren't an integer division of the FFT size can lead to artefacts on reconstruction

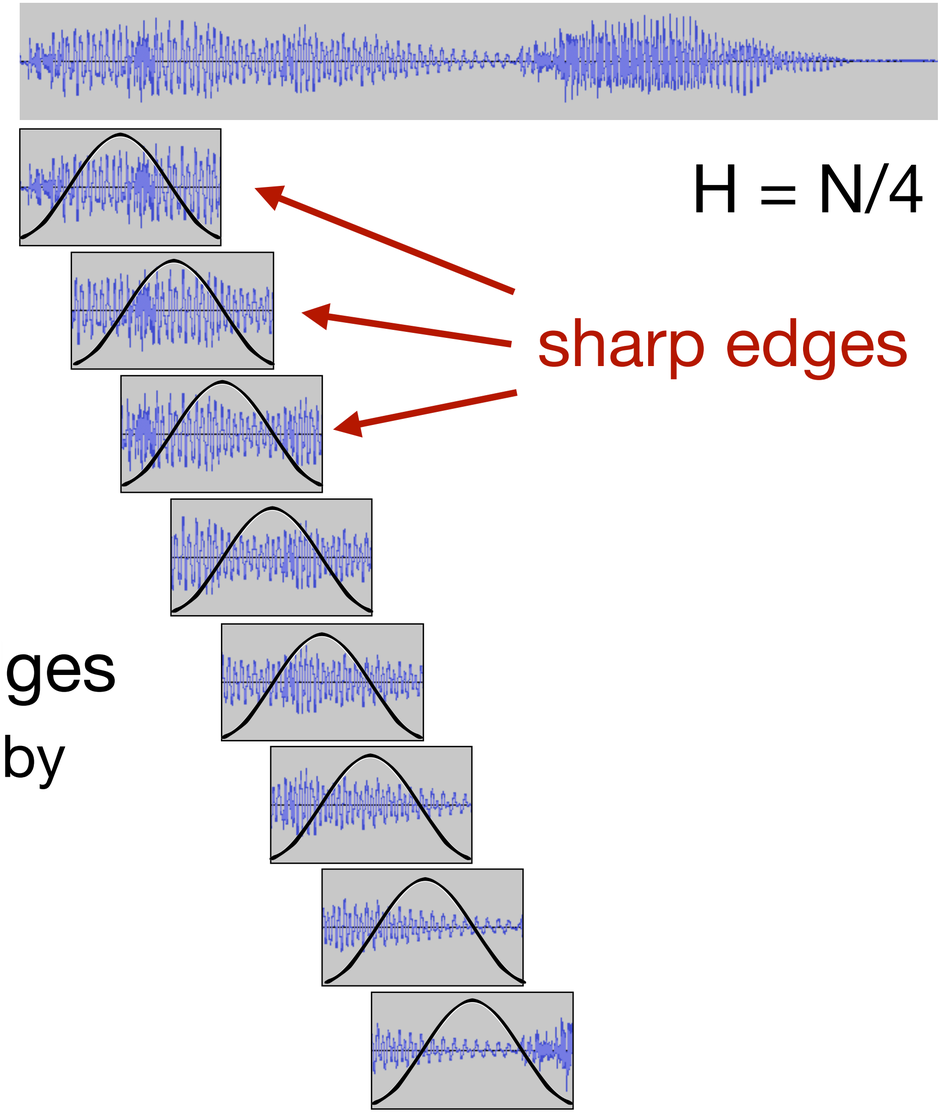


Windowing

- Overlap-Add process segments the signal into finite-length windows
  - These segments can have discontinuities at the edges (assuming 0 outside the window)
  - We might not notice this if we just reconstruct the signal with no frequency-domain processing
  - If we modify the signal, these sharp edges can lead to crackling or buzzing sounds
- We can use a window function to taper the edges
  - Pre-calculated shape that we multiply the segment by before FFT



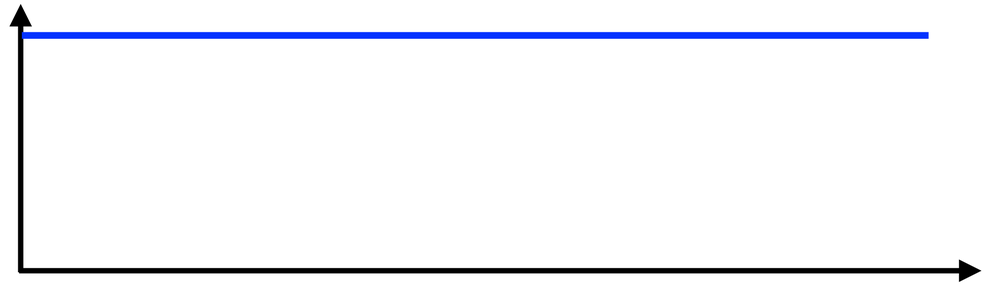
Hann window



Types of window function

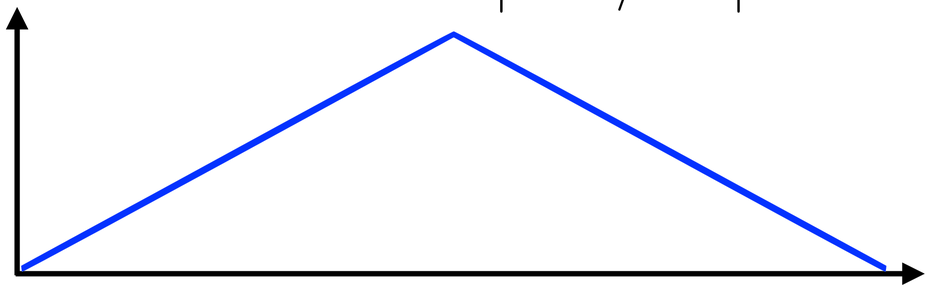
Rectangular

$$w[n] = 1 \quad \text{for } 0 \leq n < N$$



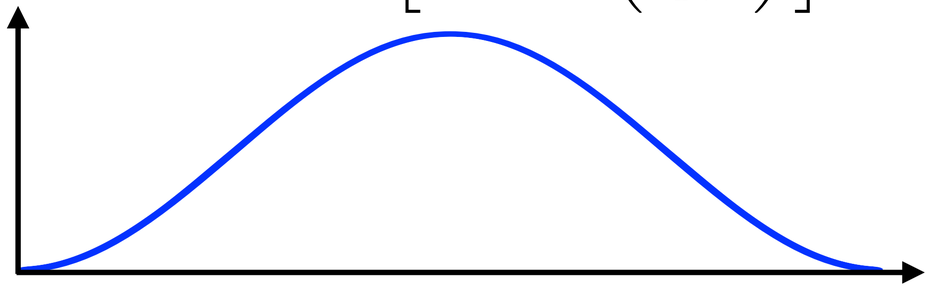
Triangular (Bartlett)

$$w[n] = 1 - \left| \frac{n - N/2}{N/2} \right|$$



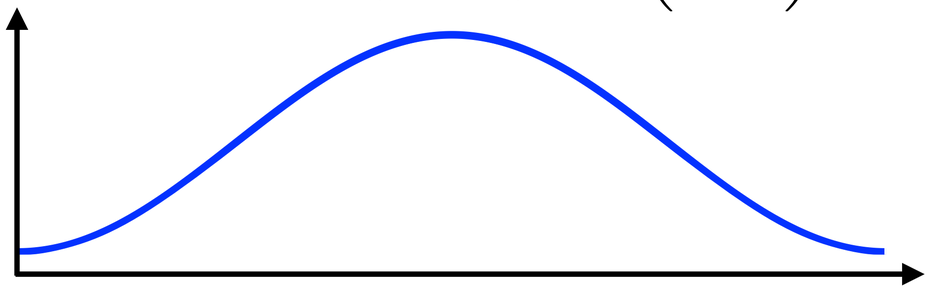
Hann

$$w[n] = 0.5 \left[1 - \cos\left(\frac{2\pi n}{N} \right) \right]$$



Hamming

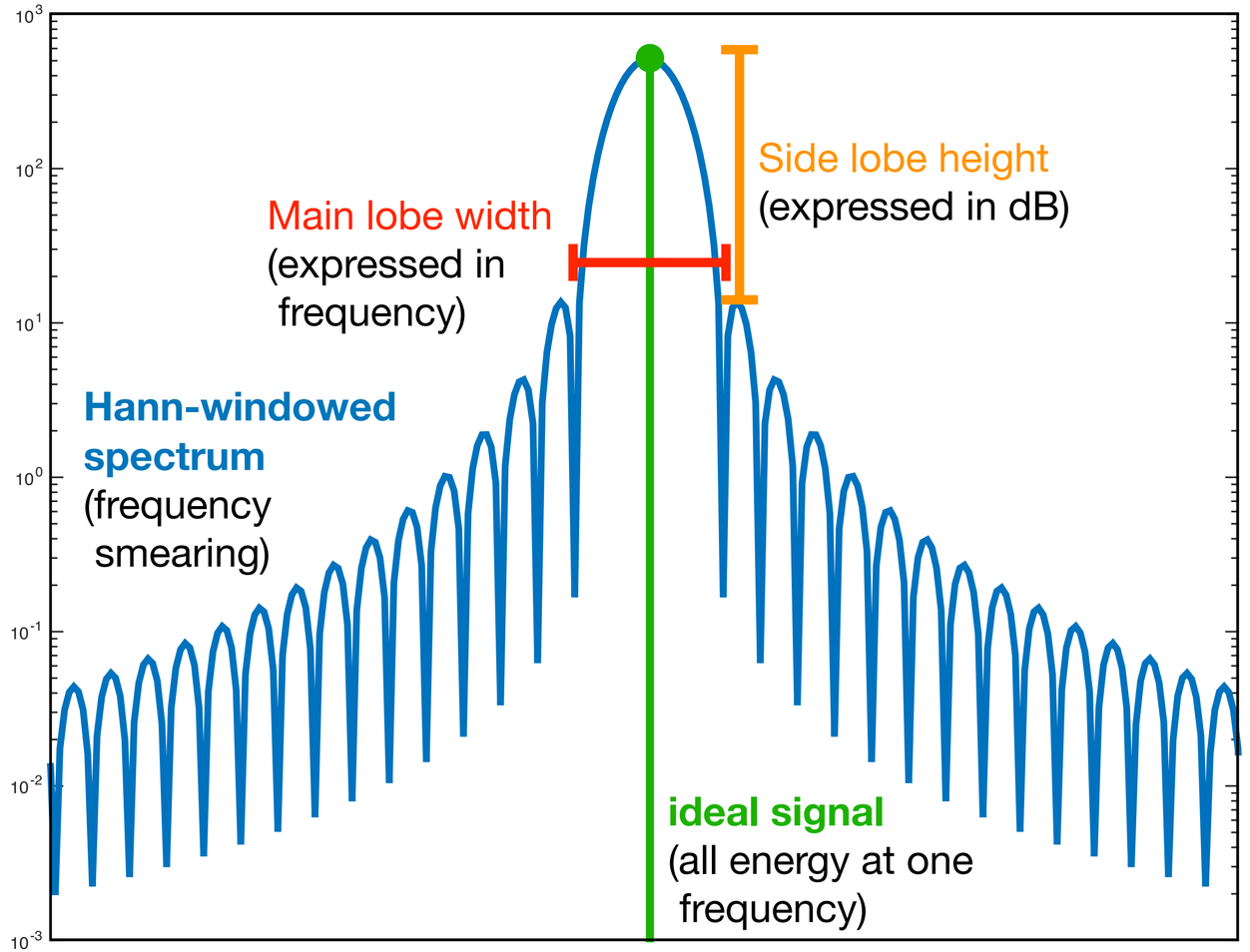
$$w[n] = \frac{25}{46} - \frac{21}{46} \cos\left(\frac{2\pi n}{N} \right)$$



And more types: Blackman, Kaiser, Flat Top, ...

Effects of windowing

- Windowing lets us isolate a particular time segment of our signal
- The cost is smearing of energy in the frequency domain
  - Energy appears at frequencies where it doesn't exist in the original signal
- Each frequency component of the signal appears as a series of lobes
  - Main lobe: a broad peak centred around the actual frequency
  - Side lobes: secondary peaks at other frequencies
- We typically want to minimise:
  - Main lobe width
  - Side lobe height

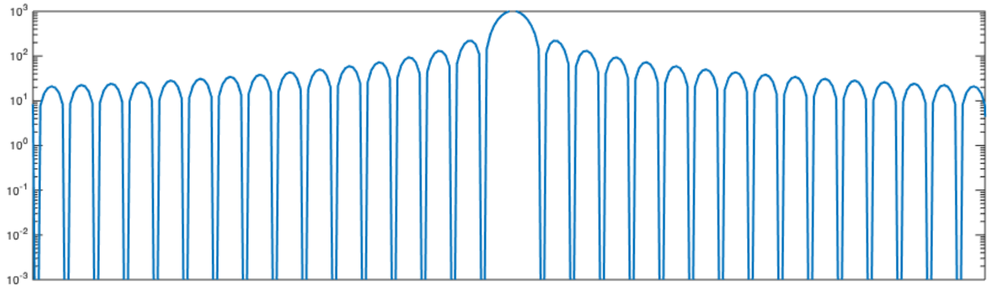


Window spectra

Rectangular

Main lobe width: 4π/N

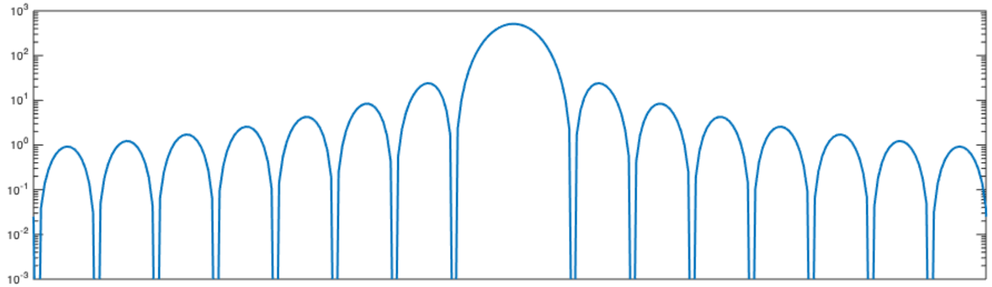
Side lobe height: -13dB



Triangular (Bartlett)

Main lobe width: 8π/N

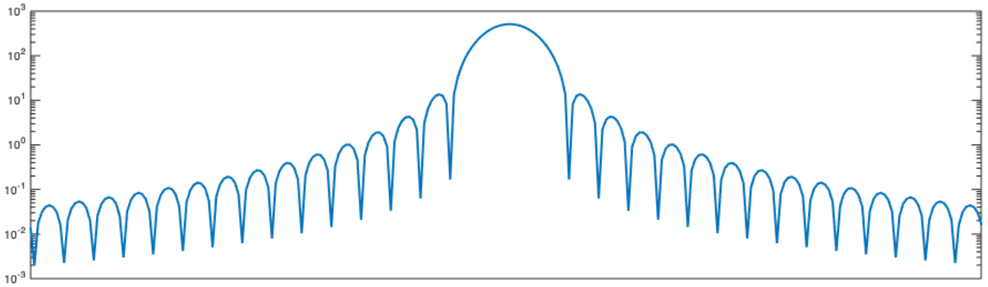
Side lobe height: -26.5dB



Hann

Main lobe width: 8π/N

Side lobe height: -31.5dB

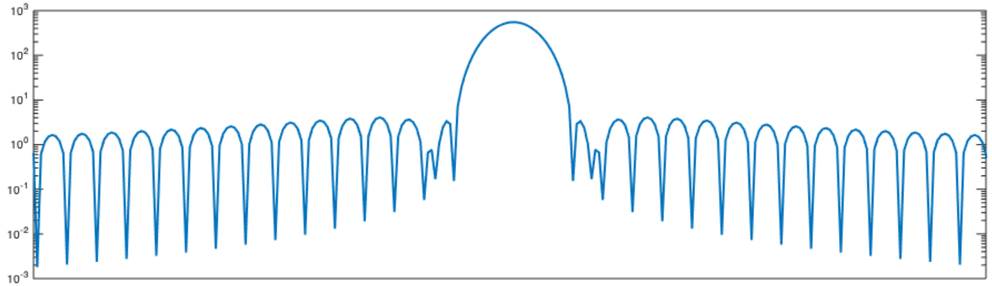


→ When in doubt: use a Hann window

Hamming

Main lobe width: 8π/N

Side lobe height: -42.8dB



Coding a window

- It is most efficient to pre-calculate the window when the program starts
  - Then recalculate when settings change: window size, window type etc.
- Store the window in an array (typically a global variable)
  - Length of the array is equal to the window size
- To calculate, use a `for()` loop to iterate over each sample of the window

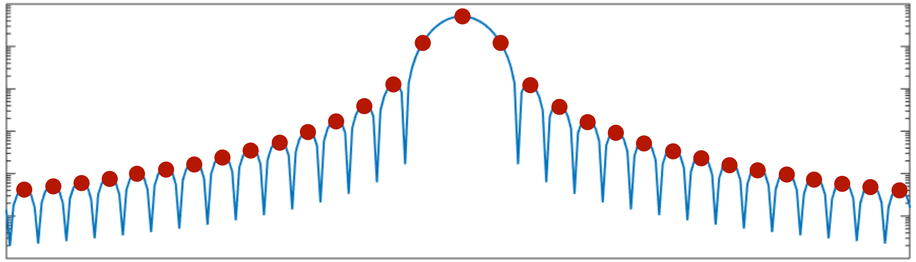
```
const int windowSize = 1024;
float windowBuffer[gWindowSize];

for(unsigned int n = 0; n < windowSize; n++) {
    windowBuffer[n] = equation for window
}
```

- To apply, multiply the signal by the window buffer frame-by-frame
  - Could do this while unwrapping a circular buffer (see Lecture 18)

```
for(unsigned int n = 0; n < windowSize; n++) {
    windowedSignal[n] = originalSignal[n] * windowBuffer[n];
}
```

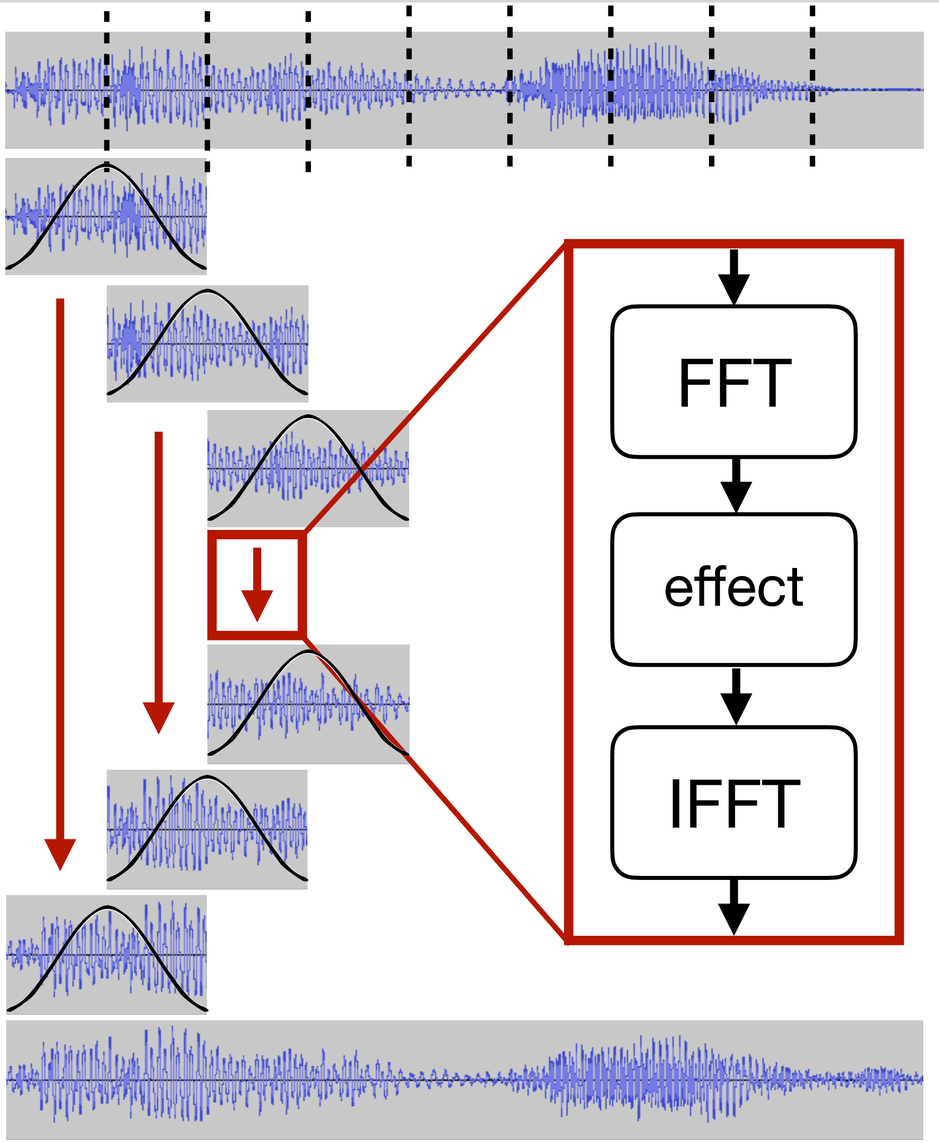
Visualising the effects of windowing

- Let's go back to our fft-sine project and see the window in action
- Task: in fft-sine, implement a Hann window
  - Change the frequency in the GUI so it doesn't align with any bin
  - What happens to the width of the main lobe?
  - What happens to the side lobes?
  - As a bonus, the Hann window should improve the frequency estimation!
- Why didn't we see the classic comb filter pattern?
  - Because the window size is the same as the FFT size
  - The FFT is effectively sampling the peaks of each lobe
- Task 2: in `setup()`, change the window size to be a fraction of the FFT length
  - Say 1/4 or 1/8 of the FFT length
  - Now the lobes should be visible



Next lecture: Phase vocoder, part 3

- Next time, we will bring everything together to create phase vocoder effects:
  - Robotisation (phase zeroing)
  - Whisperisation (phase randomisation)
  - Pitch shifting



Keep in touch!

Social media:

@BelaPlatform

forum.bela.io

blog.bela.io

More resources and contact info at:

learn.bela.io/resources