6.300: Signal Processing

Short-Time Fourier Transform

- spectrograms
- overlap-add

Quiz 2: November 10, 2-4pm, room 50-340 (Walker).

- Closed book except for two pages of notes (8.5"x11" four sides total).
- No electronic devices. (No headphones, cellphones, calculators, ...)
- Coverage up to and including classes on November 5 and HW 8.

There is no HW 9.

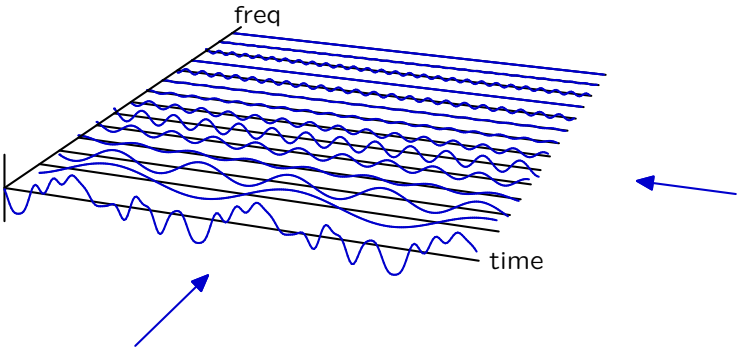
If you have personal or medical difficulties, please contact S^3 and/or `6.300-instructors@mit.edu` for accommodations.

November 03, 2026

Time versus Frequency

Fourier methods provide an alternative (frequency) view of a signal.

Two views: as a **function of time** or as a **function of frequency**.

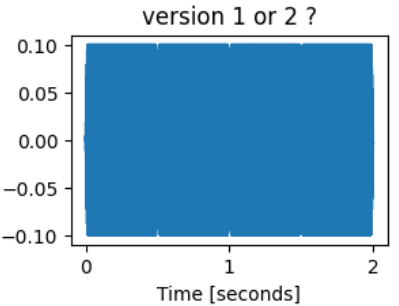


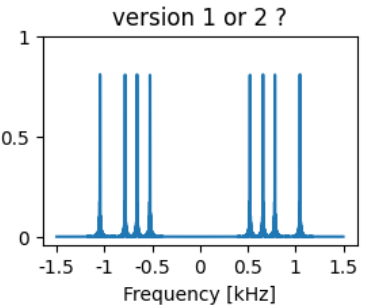
While each view is useful, many real-world signals are more naturally described in a middle ground: with **frequency content that varies with time**.

Time-Varying Frequency Content

Frequency content of signals such as speech and music varies with time.

Example: **tones** – listen to **version 1** and **version 2**



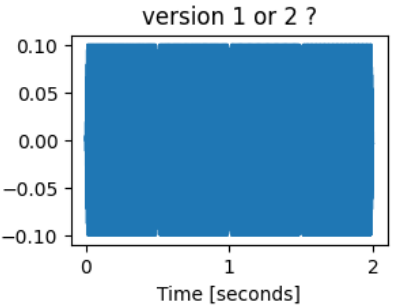


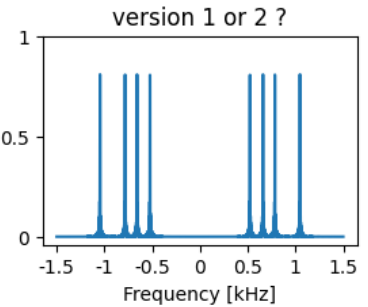
Time-Varying Frequency Content

Frequency content of signals such as speech and music varies with time.

Example: **tones** – listen to **version 1** and **version 2**

The important information for distinguishing these signals is not obvious in either the time or the frequency view.

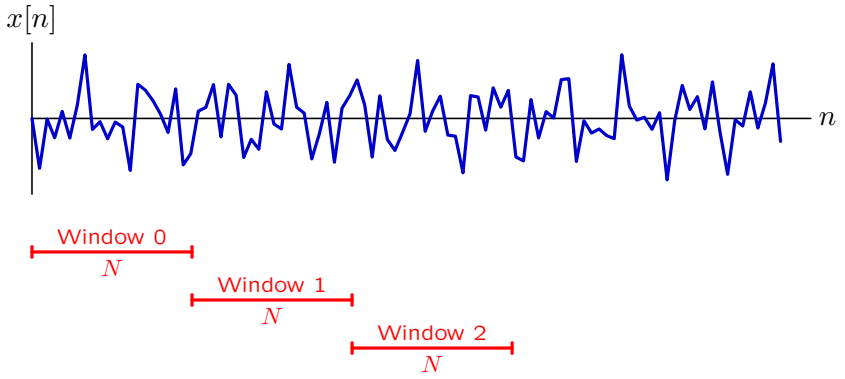




Difficult to discern frequencies in time view (left).
Difficult to discern times in frequency view (right).

Short-Time Fourier Transform

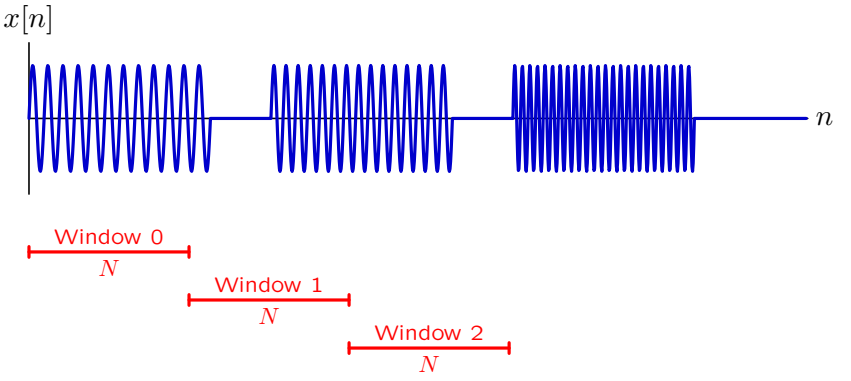
Short-time Fourier transforms (STFTs) represent the frequency content of a long signal by a sequence of shorter DFTs.



- Each DFT is computed for a time interval of length N.
- Successive time intervals begin at increasingly later times.

Short-Time Fourier Transform

Each window highlights frequencies from a different part of time.



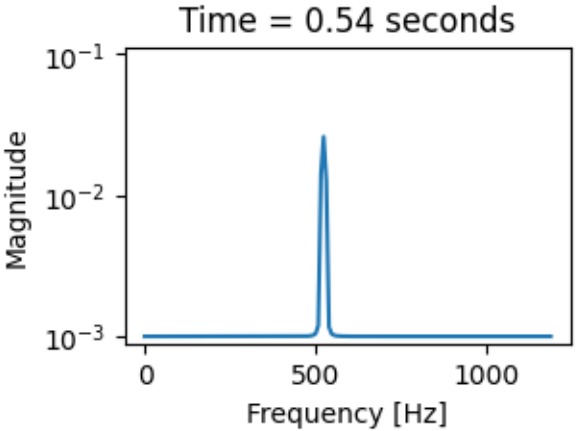
Window 0 highlights the frequency of the first tone.

Window 1 contains contributions from the first two tones.

Window 2 ...

Short-Time Fourier Transform

View the results as a sequence of DFTs: one every 0.03 seconds.

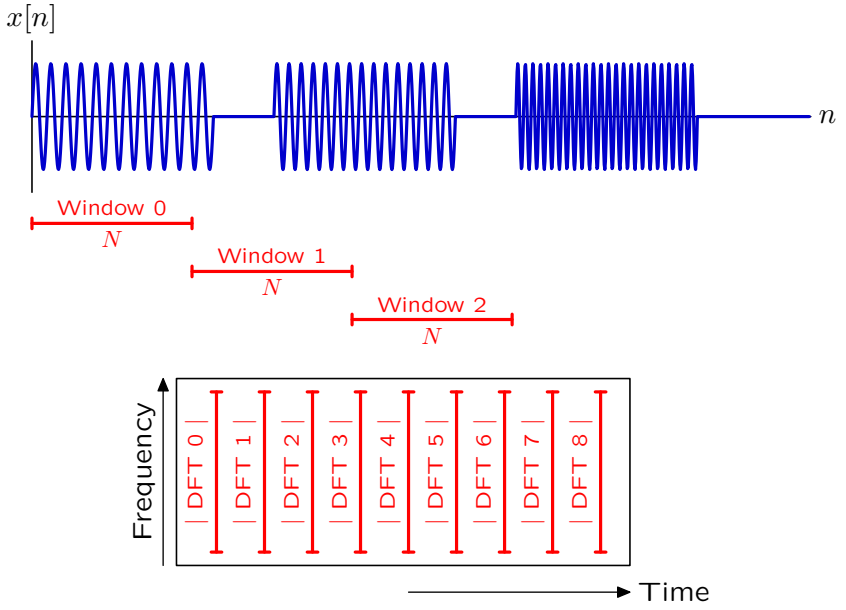


This representation illustrates both frequency and time – clearly representing an increase in tone frequency with time.

But this representation as video is neither compact nor easy to work with.

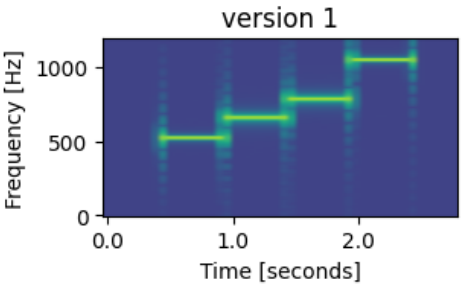
Spectrogram

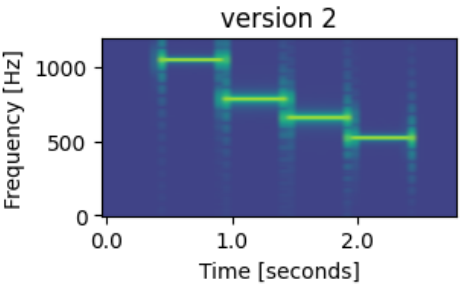
A spectrogram displays the successive DFT magnitudes as columns in a two-dimensional representation.



Spectrogram

The following images show spectrograms for the previous tone sequences. Large magnitudes are shown in bright green, small magnitudes are shown in dark blue.

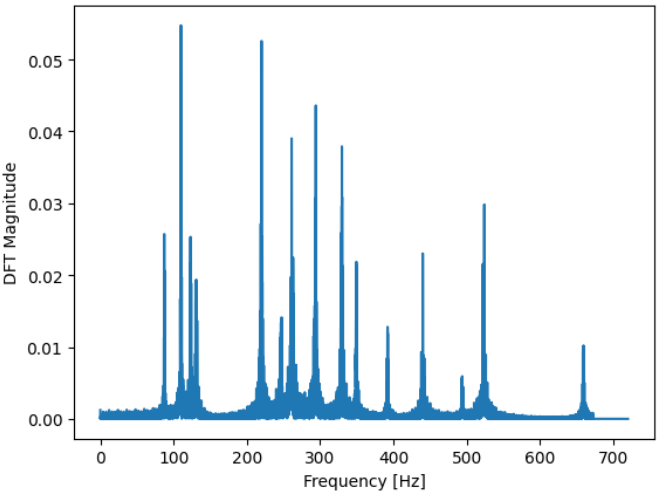




Version 1 clearly represents a sequence of tones with increasing frequency, while version 2 represents a sequence with decreasing frequency.

Frequency Representations of Music

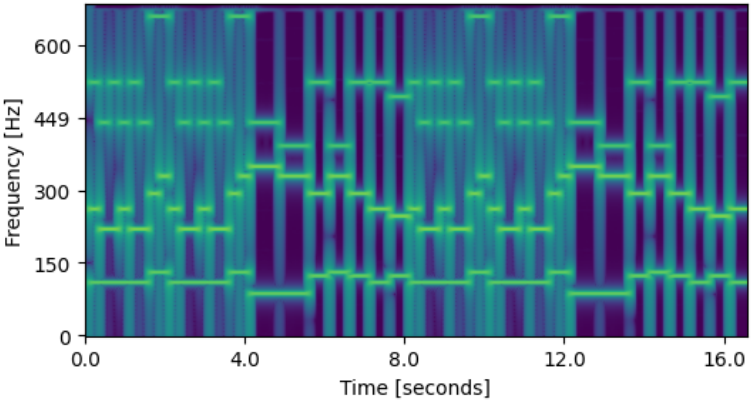
The DFT coefficients contain a great deal of information about music,
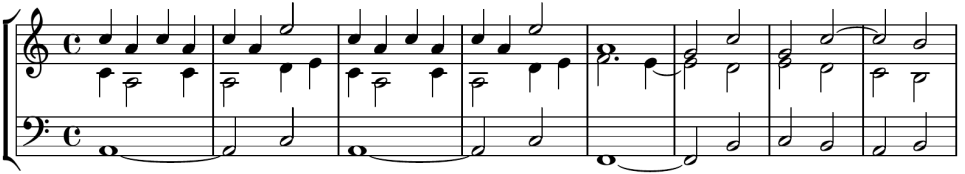


but specrograms are much easier to interpret.

Spectrogram

The spectrogram shows three parts: bass, melody, and harmony.



Structure is similar to that of musical score.

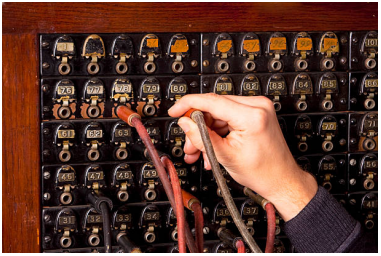


This score was created from the spectrogram using Lilypond (GNU project).

Spectrograms in Telephony

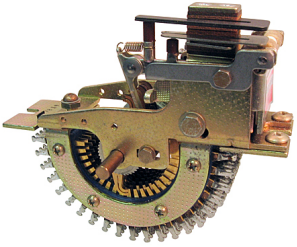
In early telephone systems (circa 1880) clients were connected manually through a switchboard.





The first automated systems (circa 1900) used pulses from a rotary dial to route calls via complicated systems of relays.





Dual Tone Multi Frequency Signaling

In the 1950s, Bell Labs pioneered a system to route phone calls using tones.

		High-Group Frequencies			
		1209Hz	1336Hz	1477Hz	1633Hz
Low-Group Frequencies	697Hz	1	ABC 2	DEF 3	A
	770Hz	GHI 4	JKL 5	MNO 6	B
	852Hz	PRS 7	TUV 8	WXY 9	C
	941Hz	*	OPER 0	#	D

This method is still in use today, especially as a way to collecting data:

"Please enter your 16-digit account number followed by the # key."

Dual Tone Multi Frequency Signaling

In the 1950s, Bell Labs pioneered a system to route phone calls using tones.

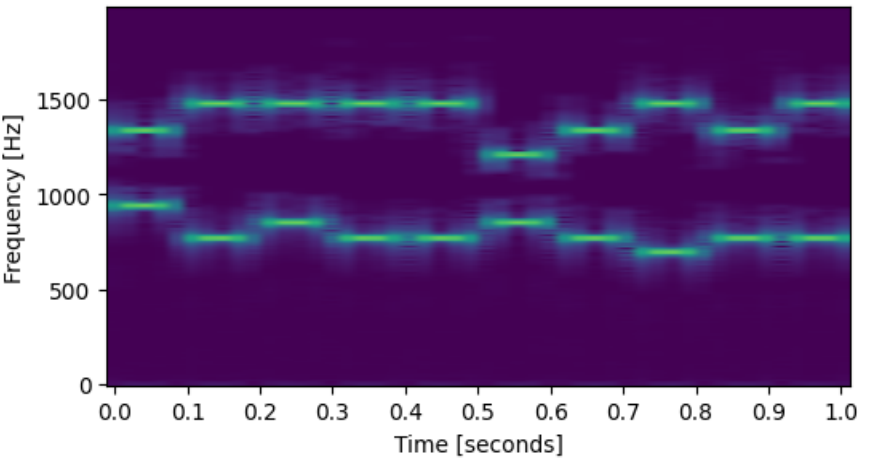
		High-Group Frequencies			
		1209Hz	1336Hz	1477Hz	1633Hz
Low-Group Frequencies	697Hz	1	ABC 2	DEF 3	A
	770Hz	GHI 4	JKL 5	MNO 6	B
	852Hz	PRS 7	TUV 8	WXY 9	C
	941Hz	*	OPER 0	#	D

Pressing a button transmits two frequencies: one representing the row number and one representing the column number (the last column is rarely used today).

Decoding the sequence of tones requires recognizing those two frequencies.

Spectrogram

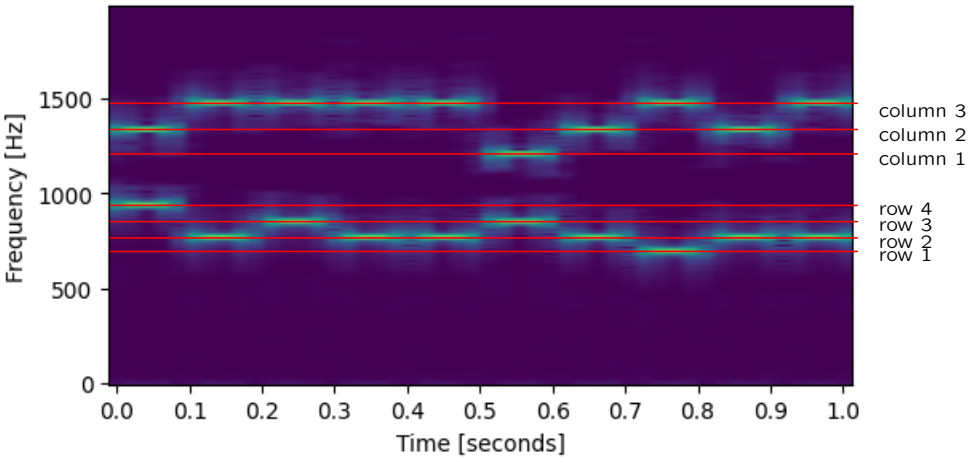
Here is a spectrogram for the previous signal.



It clearly shows a sequence of frequency pairs.

Spectrogram

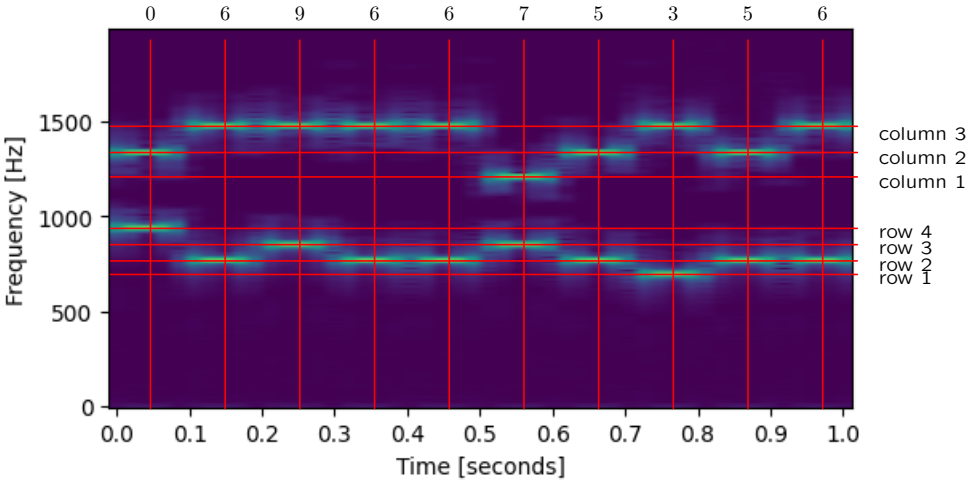
Here is a spectrogram for the previous signal.



The upper trace corresponds to the columns and the lower trace corresponds to the rows.

Spectrogram

Here is a spectrogram for the previous signal.



The pair uniquely identifies the pushbutton number.

1	2	3
4	5	6
7	8	9
*	0	#

Check Yourself

Choosing parameters of a spectrogram for analysis of music.

Determine values of N that can be used to construct a spectrogram to separate musical notes based on DFT analysis of a recording at 44,100 samples/sec.

Errors in frequency should be less than or equal to 5 Hz, so that we can resolve middle C (261.63 Hz) from C# (277.18 Hz).

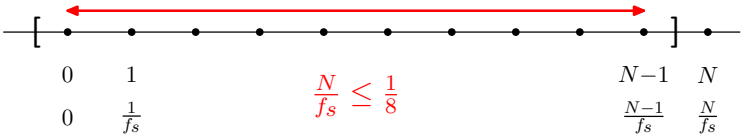
Frequencies should be computed separately for 1/8 second windows, so that we can tell if two notes are sounding together or separately.

What DFT analysis length N will work best?

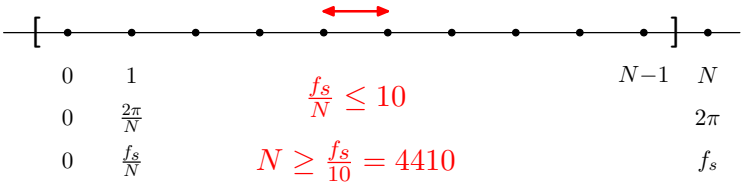
1. $N > 4410$
2. $N < 4410$
3. $N < 5512$
4. $4410 < N < 5512$
5. none of the above

Resolving Time and Frequency

The time window is divided into N samples numbered $n = 0$ to $N-1$.



Discrete frequencies are similarly numbered as $k = 0$ to $N-1$.



N determines both the length of the window in time and the frequency resolution of the result.

Check Yourself

Choosing parameters of a spectrogram for analysis of music.

To make frequency errors less than 5 Hz, we need to analyze frequencies into bins of 10 Hz width, or smaller.

A DFT breaks the full range of frequencies (44,100 Hz) into N bins, so the bin width is 44,100/N, and this bin width should be less than 10 Hz. So $N \geq 4410$.

A DFT breaks time into chunks of length $N\Delta = N/f_s$. To keep the chunks smaller than 1/8 second, $N/f_s < 1/8$ so N should be less than 44100/8=5512.

So N should be between 4410 and 5512.

Check Yourself

Choosing parameters of a spectrogram for analysis of music.

Determine values of N that can be used to construct a spectrogram to separate musical notes based on DFT analysis of a recording at 44,100 samples/sec.

Errors in frequency should be less than or equal to 5 Hz, so that we can resolve middle C (261.63 Hz) from C# (277.18 Hz).

Frequencies should be computed separately for 1/8 second windows, so that we can tell if two notes are sounding together or separately.

What DFT analysis length N will work best? 4

1. $N > 4410$
2. $N < 4410$
3. $N < 5512$
4. $4410 < N < 5512$
5. none of the above

Short-Time Fourier Transforms

Short-Time Fourier transforms are useful for constructing **spectrograms** to visualize the frequency content of a signal as a function of time.

– convert important information into a single picture, process as image.

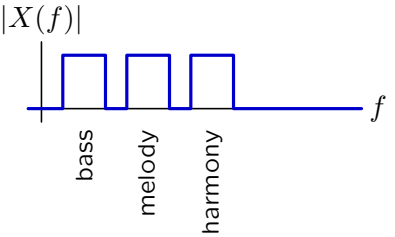
– next lecture, how spectrograms are used in analyzing speech.

Short-Time Fourier transforms are also useful for processing long signals, such as those that are common in **streaming** applications.

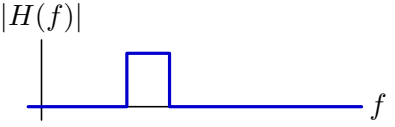
Example

Consider a musical piece that contains three simultaneous "voices," each playing a single sinusoidal tone (lab 6):

Voice 1 (bass):	40-170 Hz
Voice 2 (melody):	170-340 Hz
Voice 3 (harmony):	340-750 Hz

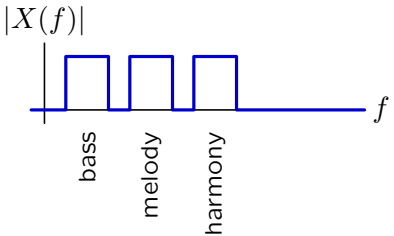


We would like to remove the bass and harmony voices, leaving just melody.

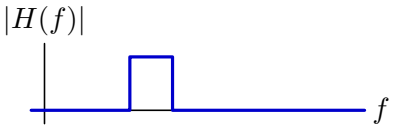


Example

Consider a musical piece that contains three simultaneous "voices," each playing a single sinusoidal tone (lab 6:



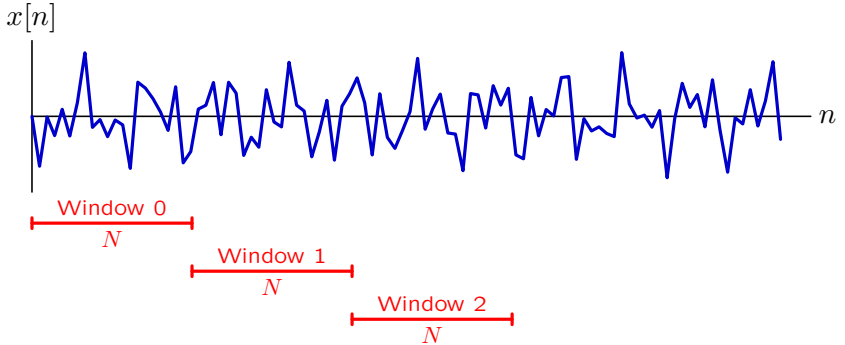
We would like to remove the bass and harmony voices, leaving just melody.



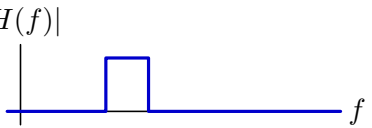
The straightforward approach is to filter the DFT of the piece, passing only the frequencies of interest. This straightforward approach requires accessing the entire piece before any part is ready to play. This approach is problematic for **streaming** applications.

Streaming Algorithm 1

Divide a signal $x[n]$ into a sequence of **shorter** signals of length N.



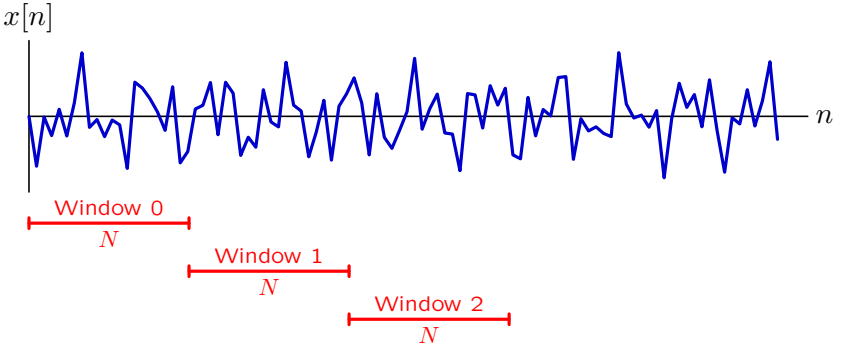
Filter data **in each window** by computing its DFT and zeroing components outside passband.



Assemble results for each window to form the output signal.

How Effective is Algorithm 1?

Divide a signal $x[n]$ into a sequence of **shorter** signals of length N.



Compare the **original:**

- `am_resynth.wav`

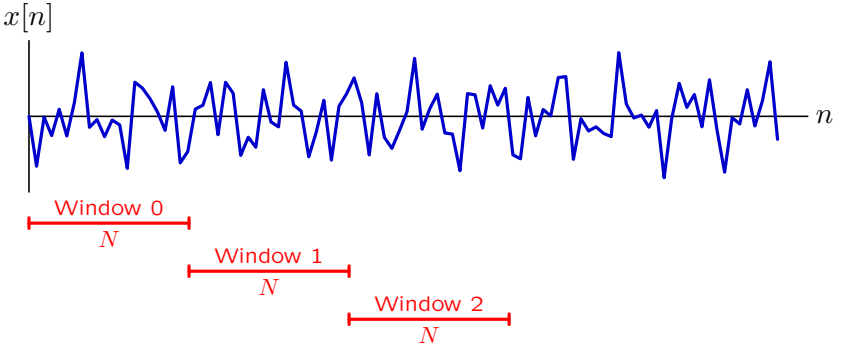
with a version that is processed **one window at a time:**

- `am_algorithm1.wav`

It isolated the melody, but also added clicks!

Streaming Algorithm 1

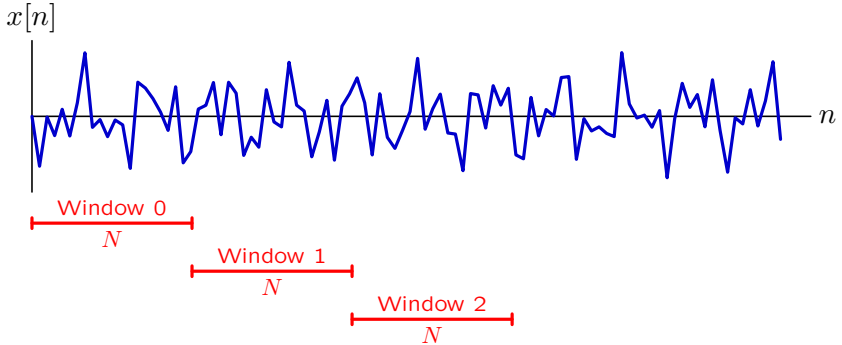
There are at least two major problems with this approach.



Can you identify the problems?

Streaming Algorithm 1

There are at least two major problems with this approach.

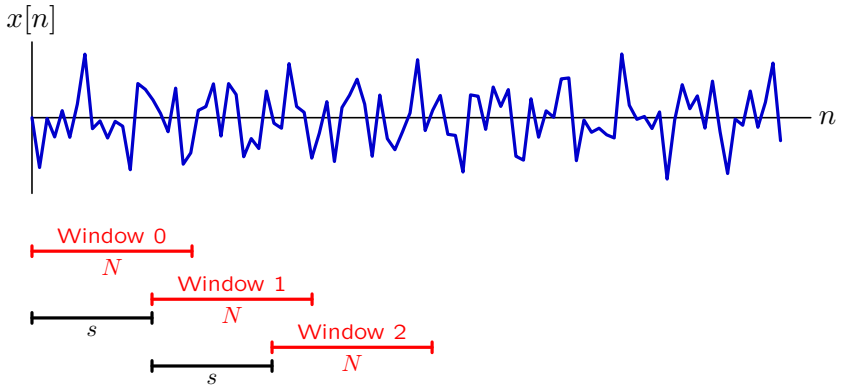


- The length of $(x * h)[n] = \text{len}(x) + \text{len}(h) - 1$.
  Part of result from each window should fall into an adjacent window(s).

- Even worse, the convolution will be circular if implemented with a DFT.
  Results from window 1 that should fall into window 2 will alias back to the beginning of window 1!

Overlap-Add Method

Avoid circular convolution artifacts and spill over problems by filling each window with just $s < N$ input samples and then zero-padding.

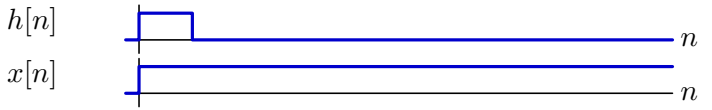


If the length of $h[n] \leq N-s+1$, then the length of $h[n]$ convolved with s samples of the input will be $\leq N \rightarrow$ no circular convolution artifact.

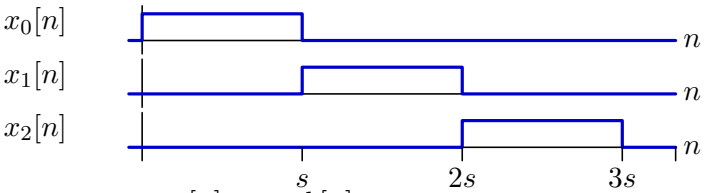
If the length of $h[n] \leq N-s+1$, then the overlapping portions of adjacent windows will accomodate spill over between windows.

Overlap-Add: Graphical Depiction

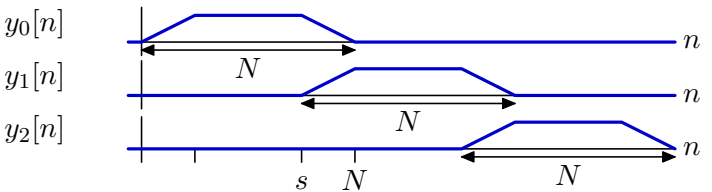
Convolve a square pulse with a signal that is 1 for all $n \geq 0$.



Divide the input $x[n]$ into pieces that are each of length s.



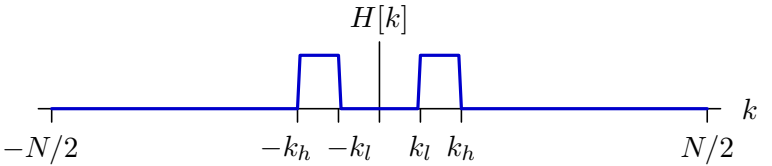
Convolve each piece of $x[n]$ with $h[n]$.



Then the output $y[n] = y_0[n] + y_1[n] + y_2[n] + \cdots$ Hence overlap-**add**.

Filter Design

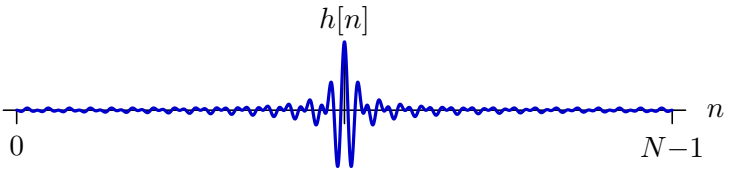
Design a filter to isolate the melody using the overlap-add method.



The filter should pass frequencies in the range $f_l \leq f \leq f_h$.

$$k_l = \frac{f_l}{f_s} N\,; \qquad k_h = \frac{f_h}{f_s} N$$

If we take the window length $N = 8192$, then $h[n]$ has that same length.



This design leads to algorithm 1, and explains the clicking artifacts.
We need a shorter $h[n]$ to prevent circular convolution artifacts.

Filter Design

How can we design a filter with 2048 points in time (n) but 8192 points in frequency (k)?

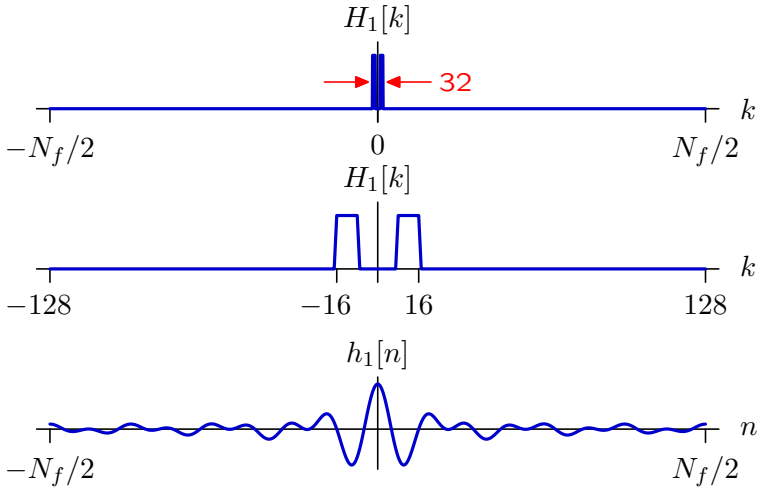
- Start by designing a filter $H_1[k]$ with length $N_f = 2048$. The filter should only pass frequencies in the range $f_l \leq f \leq f_h$.
- Convert $H_1[k]$ to the time domain using an inverse DFT. The length of the resulting $h_1[n]$ will be $N_f = 2048$.
- Define a new filter $h_2[n]$ which is a version of $h_1[n]$ that is zero-padded to a new length of $N = 8192$.
- Convert $h_2[n]$ to the frequency domain to get $H_2[k]$.

The filter $H_2[k]$ will have 8192 values of k but its time-domain representation $h_2[n]$ will have just 2048 non-zero values.

Filter Design

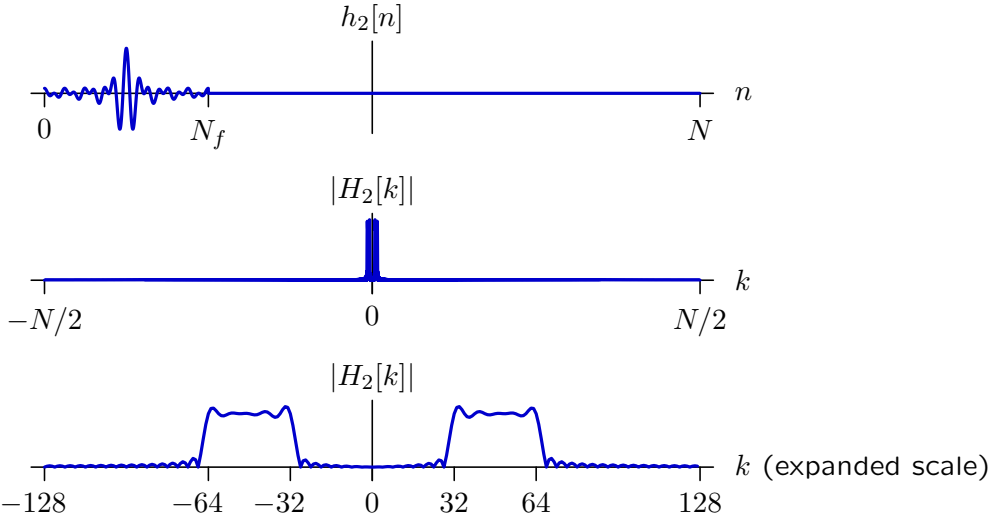
Design a bandpass filter to extract 170-340 Hz frequency region from signal sampled with $f_s = 44,100$ Hz with $N_f = 2048$.

$$\frac{170}{f_s} \times N_f \approx 8 \quad \leq \quad k \quad \leq \quad \frac{340}{f_s} \times N_f \approx 16$$



Filter Design

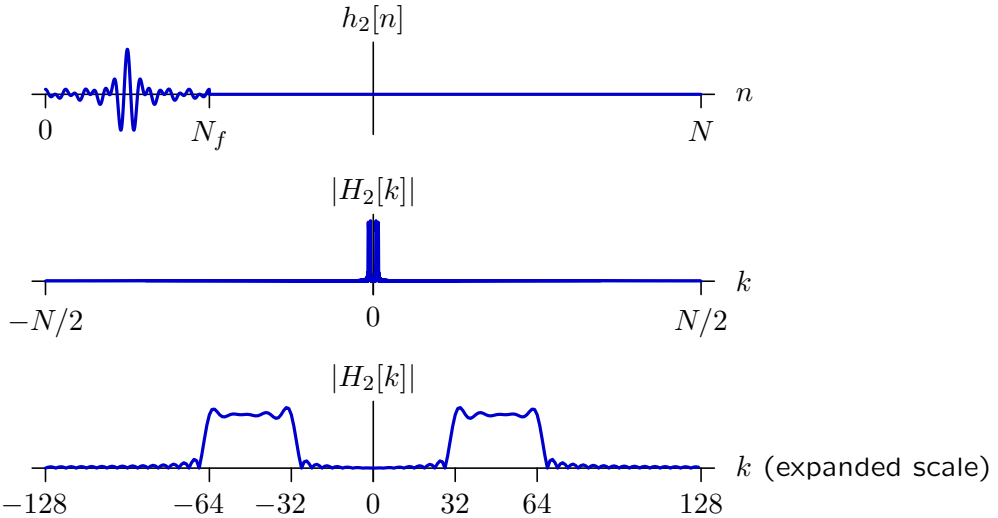
Zero-pad to make filter length equal to window length $N = 8192$.



Listen to result: `am_filtered.wav`

Filter Design

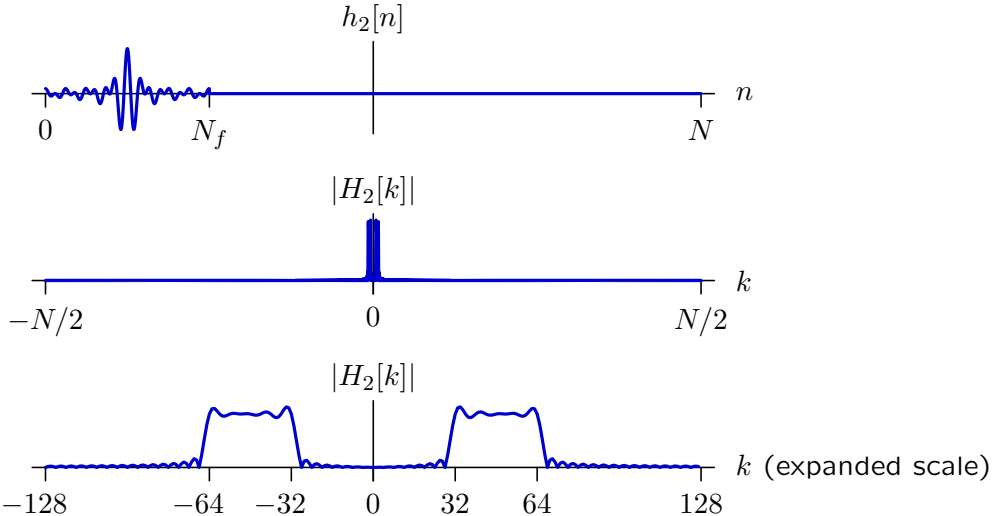
Zero-pad to make filter length equal to window length $N = 8192$.



Listen to result: `am_filtered.wav` Significant improvement. No clicks.

Filter Design

Zero-pad to make filter length equal to window length $N = 8192$.



Listen to result: `am_filtered.wav` Significant improvement. No clicks. Bass and harmony are faintly audible – probably because of deviations from ideal filters. Ripples are due to Gibb's phenomenon.

Gibb's Phenomenon

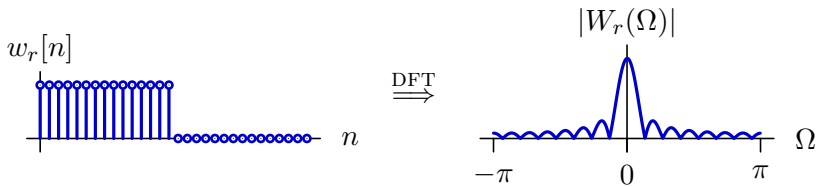
Ripples in frequency result from windowing in time.

A rectangular window in time

$$w_r[n] = \begin{cases} 1 & \text{if } 0 \le n < N \\ 0 & \text{otherwise} \end{cases}$$

corresponds to a DT sinc in frequency.

$$W_r(\Omega) = \sum_{n=-\infty}^{\infty} w[n] e^{-j\Omega n} = \sum_{n=0}^{N-1} e^{-j\Omega n} = \frac{1 - e^{-j\Omega N}}{1 - e^{-j\Omega}} = \frac{\sin \frac{\Omega N}{2}}{\sin \frac{\Omega}{2}} \, e^{-j\Omega \frac{(N-1)}{2}}$$



Multiplying the unit sample response $h[n]$ by a window function, convolves the desired bandpass shape with the DT sinc – generating ripples.

Gibb's Phenomenon

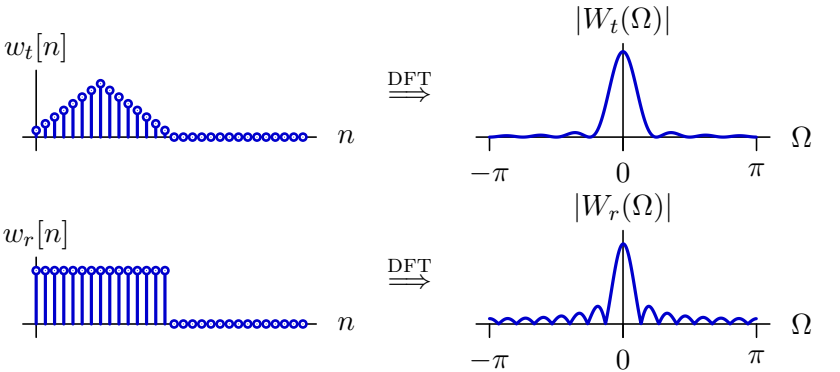
Triangular windows in time produce smaller ripples in frequency.

A triangular window in time

$$w_t[n] = w_r[n] * w_r[n]$$

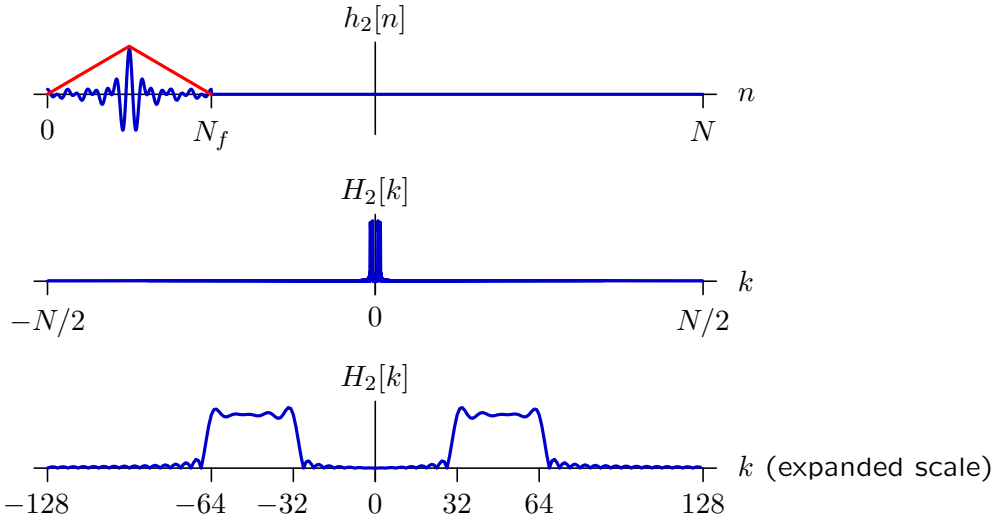
corresponds to a DT sinc squared in frequency.

$$W_t(\Omega) = W_r^2(\Omega)$$



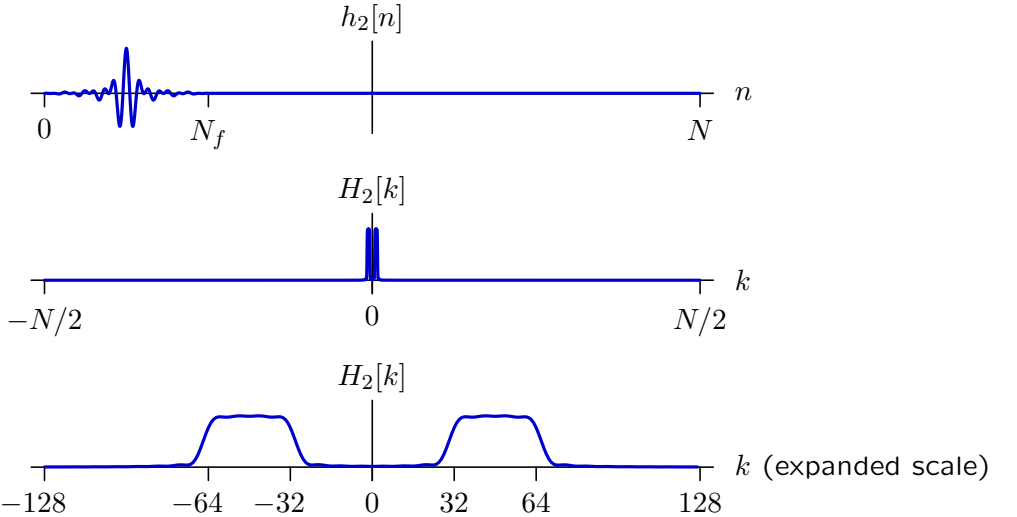
Filter Design

We can reduce the passband ripple by applying a triangular window (red).



Filter Design

We can reduce the passband ripple by applying a triangular window.

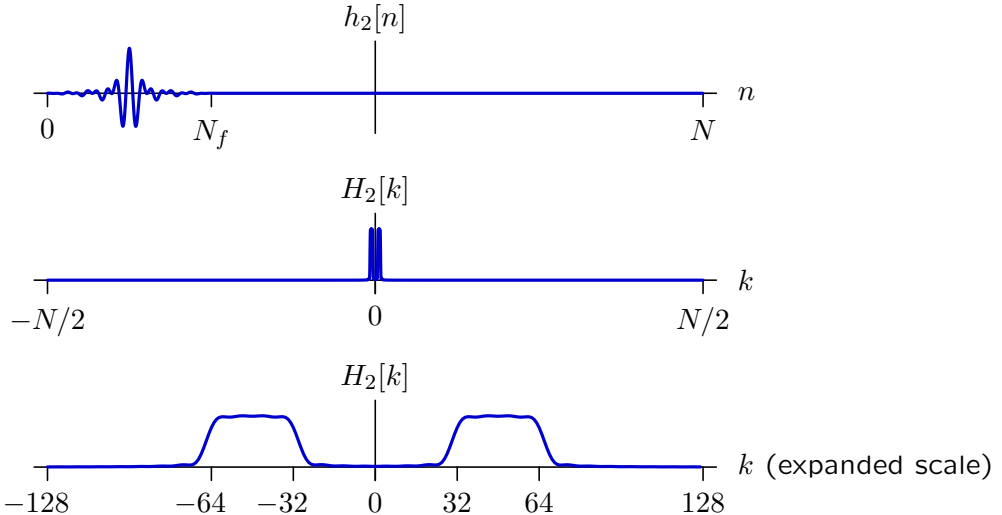


$H_2[k]$ is now a smoother function of k, rippling is greatly reduced.

Listen to result: `am_triangular.wav`

Filter Design

We can reduce the passband ripple by applying a triangular window.



$H_2[k]$ is now a smoother function of k, rippling is greatly reduced.

Listen to result: `am_triangular.wav` Bass and harmony are still faintly audible (although not as audible as for rectangular window).

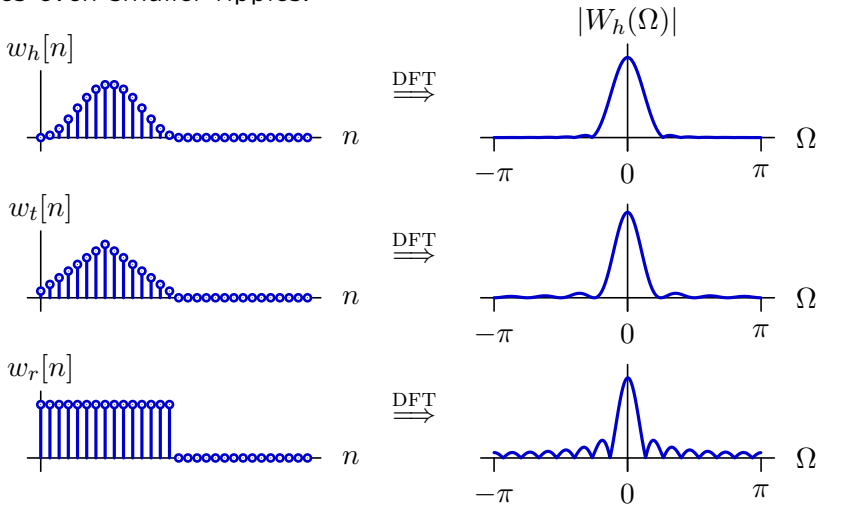
Gibb's Phenomenon

Ripples in frequency result from windowing in time.

A Hann window in time

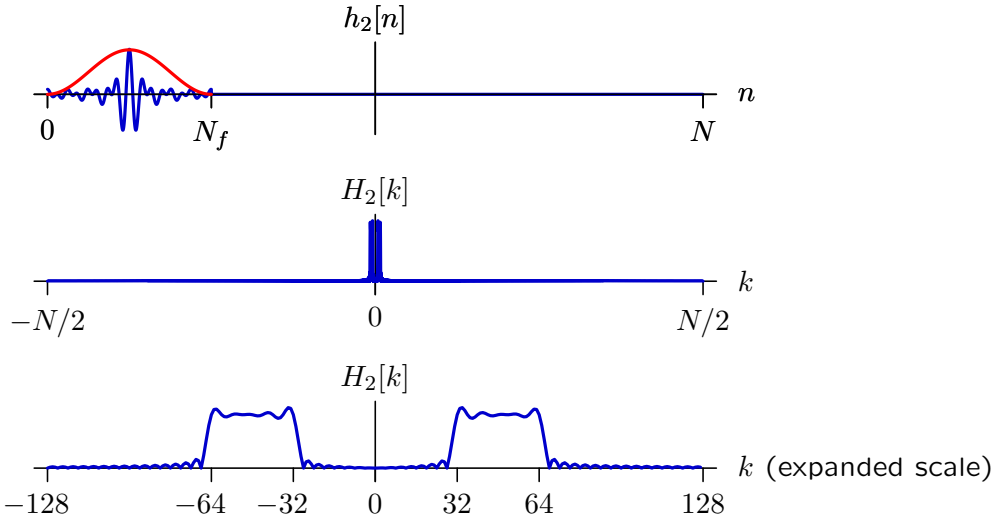
$$w_h[n] = \sin\left(\frac{\pi n}{N}\right)^2$$

produces even smaller ripples.



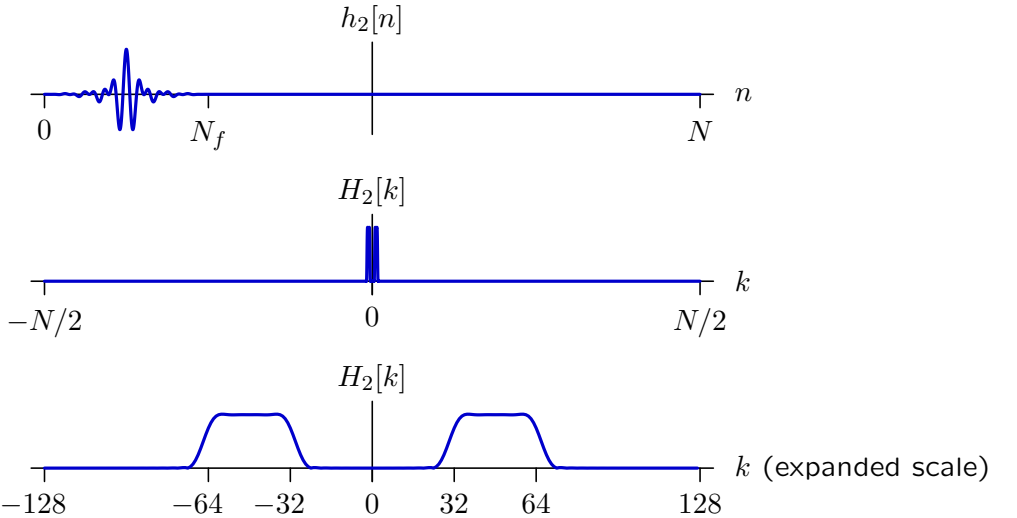
Filter Design

Better yet, try a Hann window (red).



Filter Design

Better yet, try a Hann window.

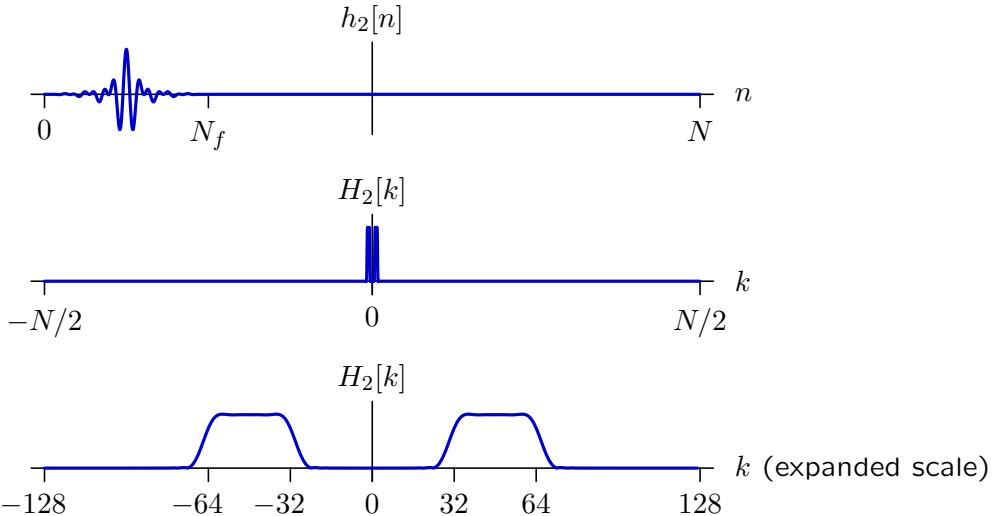


$H_2[k]$ is now even smoother.

Listen to result: `am_Hann.wav`

Filter Design

Better yet, try a Hann window.



$H_2[k]$ is now even smoother.

Listen to result: `am_Hann.wav` Bass and harmony no longer audible.

Summary

A key feature of the DFT is that it can be applied to parts of a signal

- to visualize the change in frequency with time, and/or
- to process long signals, one chunk at a time.