6.003: Signal Processing

Short-Time Fourier Transforms

- Processing streaming signals
- Computational cost
- Overlap-add method

- Quiz 2: November 2, 2-4pm, 50-340 (Walker)
  - Coverage up to and including all of week 7, including HW7.
  - Closed book except for two pages of notes (four sides total)
  - No electronic devices. (No headphones, cellphones, calculators, ...)
- No HW8 – a practice quiz is posted.

October 26, 2021

Streaming Music

One of the most important attributes of the DFT is that it can be computed from a finite part of a potentially very long signal.

The DFT is defined for $0 \le n < N$ while the DTFT has infinite limits.

DFT: $$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi k}{N}n}$$

DTFT: $$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega n}$$

Breaking a signal into pieces so that it can be processed in small chunks is especially important in **streaming** applications where the signal may be arbitrarily long.

Breaking a signal into pieces also affects the number of computations required to perform a signal processing task.

Computational Cost

How many multiplies (N_m) are needed to implement a filter $h[n]$ with length N_h=1024 for a 3 minute song, sampled at $44,100$ samples/second?

Compare three schemes:

- direct convolution
- using the DFT
- using the FFT

Which is most efficient? Which is least efficient?

Computational Cost of Convolution

How many multiplies (N_m) are needed to implement a filter $h[n]$ with length N_h=1024 for a 3 minute song, sampled at $44,100$ samples/second?

Start by finding the number of samples N_x in a 3 minute song:

$$N_x = 3\,\text{min.} \times 60\,\text{sec./min.} \times 44,100\,\text{samples/sec.} \approx 8 \times 10^6\,\text{samples}$$

Find the number of multiplies required to **convolve** an input of length N_x with a unit-sample response of length N_h.

$$y[n] = \sum_{m=\infty}^{\infty} h[m] x[n-m]$$

The range of m can be reduced since $h[m] = 0$ for $m < 0$ and for $m \ge N_h$.

$$y[n] = \sum_{m=0}^{N_h-1} h[m] x[n-m]$$

Ignoring end effects (for n near 0 and N_x), this requires

$$N_x \times N_h \approx 8 \times 10^9$$

multiplies (N_h for each sample in the input).

Computational Cost of DFT-based Convolution

Find the number of multiplies required to compute the convolution with DFTs. To compute the DFT of the input

$$X[k] = \sum_{n=0}^{N_x-1} x[n] e^{-j\frac{2\pi k}{N_x}n}$$

requires N_x multiplies for each value of k, which is N_x^2 in total.

This ignores the multiplies needed to compute the complex exponentials:

- efficient algorithms exist for computing trig functions and
- these coefficients can often be reused in subsequent calculations

Even though $N_h < N_x$, we need to compute $H[k]$ with the same resolution as $X[k]$ in order to multiply $H[k]$ times $X[k]$.

Therefore the total number of multiplies is

- N_x^2 to find $X[k]$ for all k,
- N_x^2 to find $H[k]$ for the same values of k,
- N_x to multiply $H[k]$ times $X[k]$ for all k, and
- N_x^2 to find $y[n]$ from $Y[k]$.

The total is approximately $3N_x^2 = 2 \times 10^{14}$ ($>> 8 \times 10^9$ for convolution!)

Computational Cost of FFT-based Convolution

The Fast Fourier Transform (FFT) is a highly efficient algorithm for computing the DFT. The FFT requires approximately $N \log_2 N$ multiplies (instead of N^2) to compute a DFT of length N.

Therefore the total number of multiplies using the FFT is

- $N_x \log_2 N_x$ to find $X[k]$ for all k,
- $N_x \log_2 N_x$ to find $H[k]$ for the same values of k,
- N_x to multiply $H[k]$ times $X[k]$ for all k, and
- $N_x \log_2 N_x$ to find $y[n]$ from $Y[k]$.

The total is approximately $3N_x \log_2 N_x = 6 \times 10^8$

method	number of multiplies
direct convolution	8×10^9
DFT	2×10^{14}
FFT	6×10^8

FFT is more than 300,000x faster than the DFT!

Computational Cost

It is easy to see why the FFT requires fewer multiplies than the DFT since the number of multiplies goes as $N\log_2 N$ intead of N^2.

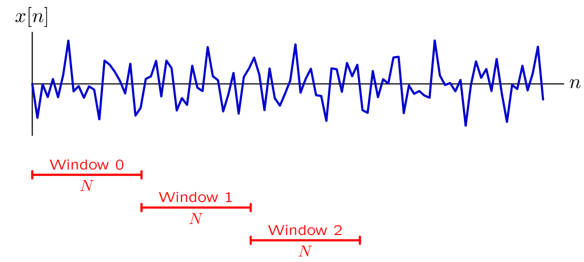
And there is still room for improvement!

Neither of the Fourier based schemes took advantage of the fact that the filter length N_h=1024 is small compared to that of the signal $N_x = 8 \times 10^6$.

We can improve the performance of the Fourier methods by using a short-time Fourier method, which allows us to use smaller transforms that are better matched to $h[n]$.

Short-Time Fourier Transforms

Short-time Fourier transforms are based on the analysis of a sequence of finite-length portions of an input signal.



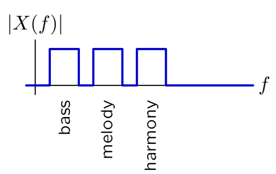
General procedure:

- Divide the input signal into a sequence of windows, each of length N.
- Process each window.
- Assemble the processed pieces together to form the output.

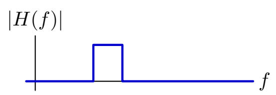
Example

Consider a musical piece that contains three simultaneous "voices," each playing a single sinusoidal tone (lab 7):

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

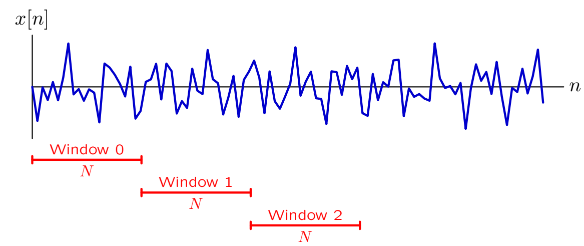


We would like to remove the bass and harmony voices, leaving just melody.

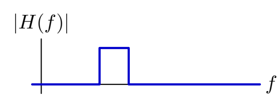


Algorithm 1

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



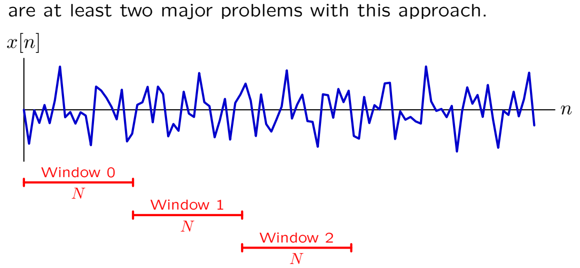
Filter data in each window by computing its DFT and zeroing components outside passband.



Assemble results for each window to form the output signal.

Algorithm 1

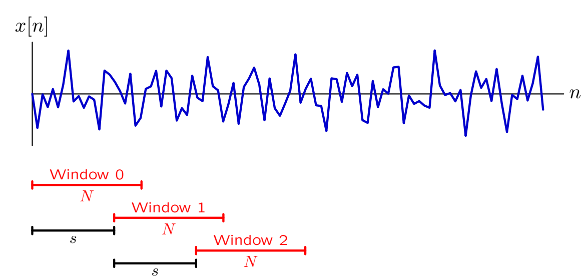
There are at least two major problems with this approach.



- The length of $(x * h)[n]$ is generally > length of $x[n]$ or $h[n]$. 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.

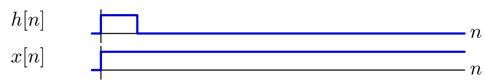


If the length of $h[n] \le N-s+1$, then the length of $h[n]$ convolved with s samples of the input will be less than N → no circular convolution artifact.

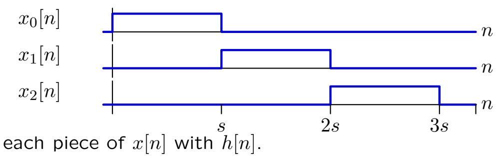
If the length of $h[n] \le N-s+1$, then the overlapping portions of adjacent windows will accomodate spill over between windows.

Overlap-Add: Graphical Depiction

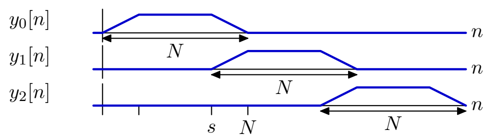
Convolve a square pulse with a signal that is 1 for all n.



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



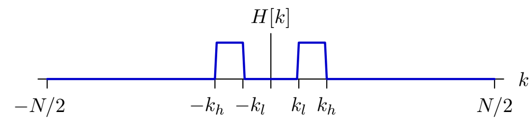
Convolve each piece of $x[n]$ with $h[n]$.



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

Filter Design

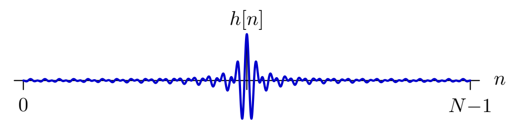
Design a filter to isolate the melody using the overlap-add method.



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

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

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



But the idea was to have a shorter $h[n]$ to prevent inter-window artifacts. This design leads to algorithm 1, and explains the clicking artifacts.

Filter Design

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

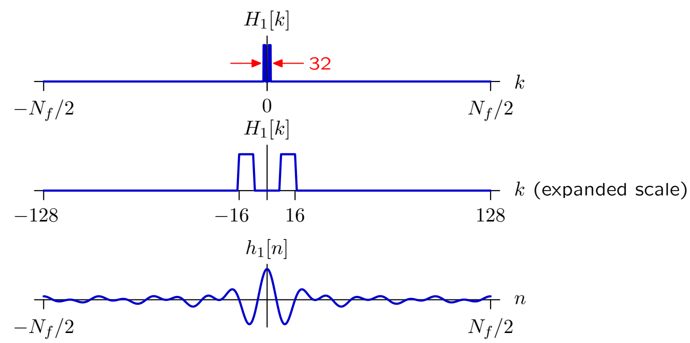
- Start by designing a filter $H_1[k]$ with length $N_f = 2048$. The filter should only pass frequencies in the range $f_l \le f \le f_h$.
- Convert $H_1[k]$ to the time domain using an inverse DFT. The length of the resulting $h_1[n]$ will be $N_f = 2048$.
- Define a new filter $h_2[n]$ which is a version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

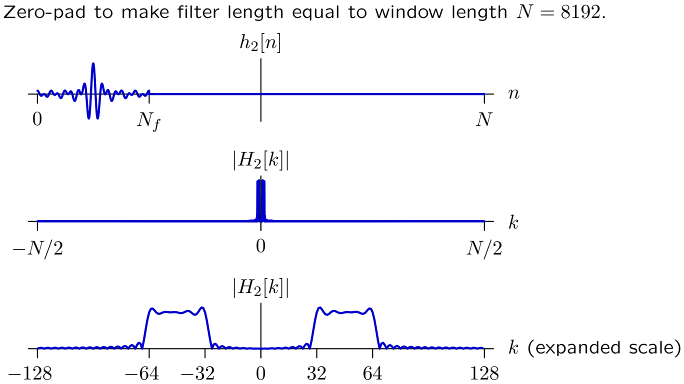
Design a bandpass filter to extract 170-340 Hz frequency region from signal sampled with $f_s = 44,100\,\text{Hz}$ with $N_f = 2048$.

$$\frac{170}{f_s} \times N \approx 8 \quad \le \quad k \quad \le \quad \frac{340}{f_s} \times N \approx 16$$



Filter Design

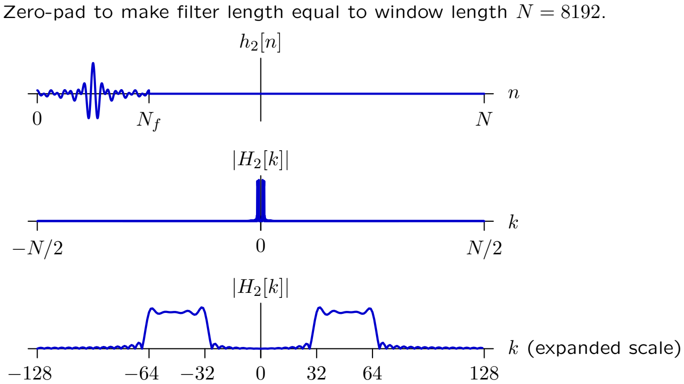
Zero-pad to make filter length equal to window length $N = 8192$.



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. Bass and harmony are faintly audible – probably because of deviations from ideal filters. Ripples are due to Gibb's phenomenon.

Gibb's Phenomenon

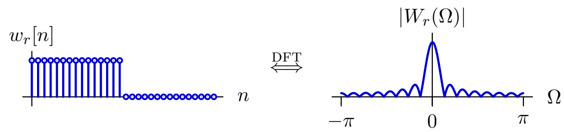
Ripples in frequency result from windowing in time.

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

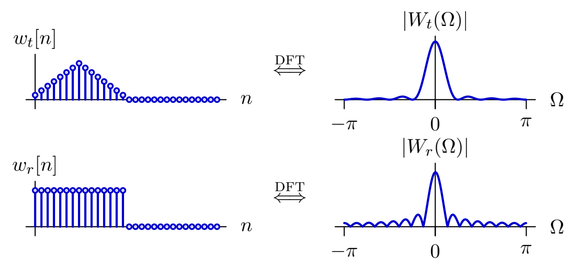
Triangular windows in time produce smaller ripples in frequency.

A triangular window in time

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

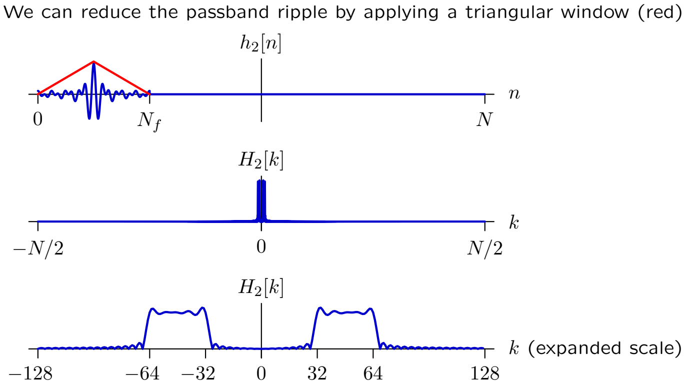
corresponds to a DT sinc squared in frequency.

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



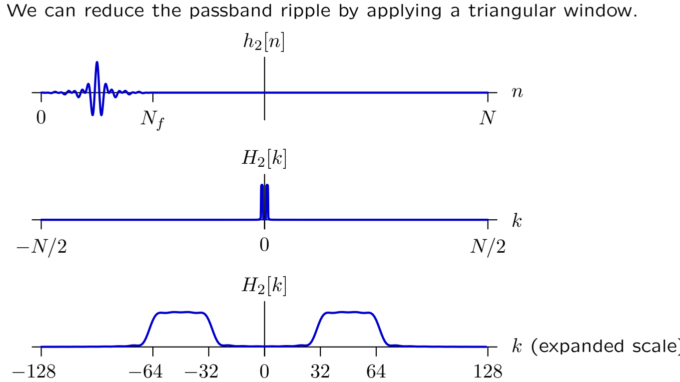
Filter Design

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`

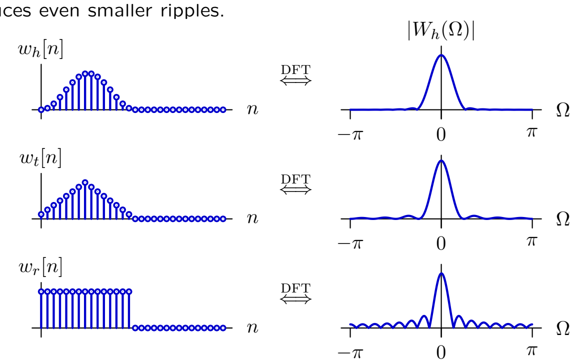
Gibb's Phenomenon

Ripples in frequency result from windowing in time.

A Hann window in time

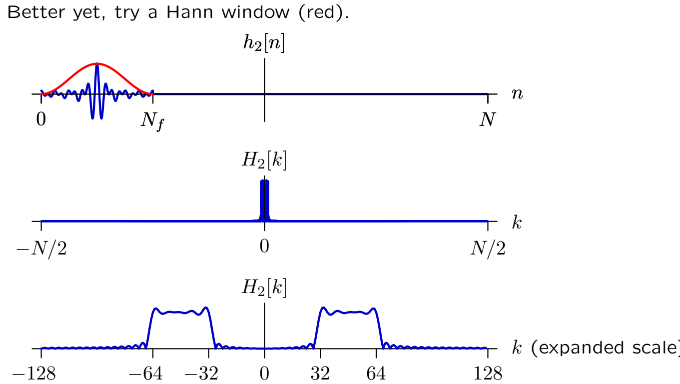
$$w_t[n] = \sin\left(\frac{\pi n}{N}\right)^2$$

produces even smaller ripples.



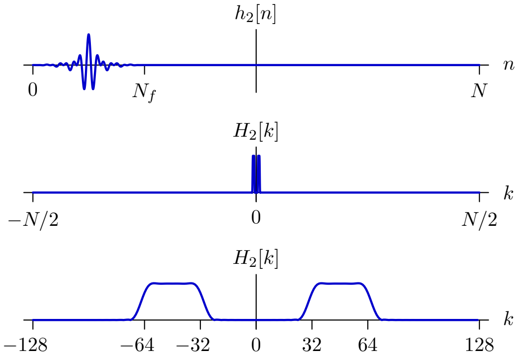
Filter Design

Better yet, try a Hann window (red).



Filter Design

Better yet, try a Hann window.

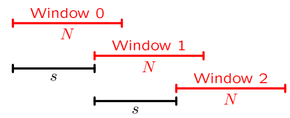


$H_2[k]$ is now even smoother.
Listen to result: `am_Hann.wav`

Computational Cost of Overlap-Add Method

Does breaking signal into windows increase or decrease computational cost?

Each FFT of length N contributes s samples to the output.



The total number of multiplies N_t to process a long signal (N_x samples) is the number of windows (N_x/s) times the number of multiplies per window ($2N\log_2 N$, since we only need to calculate frequency response once). For today's example, $N_x = 8\times 10^6$, $s = 6144$, $N = 8192$ so

$$N_t = \frac{8\times 10^6}{6144}\times 2\times 8192\times \log_2(8192)\approx 3\times 10^8$$

which is only half as many multiplies for full-length FFTs.

Even more importantly, we can process the first window without waiting for the entire song to be transmitted – very important for **streaming.**

Summary

Breaking a signal into pieces so that it can be processed in small chunks is important in **streaming** applications where the signal may be arbitrarily long.

Breaking a signal into pieces affects the **number of computations** required to perform a signal processing task.

- Implementing convolution with an FFT can be much more efficient that direct convolution.
- Implementing convolution with short-time Fourier transforms based on the FFT can be similarly efficient.

When processing a signal in chunks, care must be taken to avoid introducing **artifacts** between chunks.

There are a number of signal processing schemes (such as **overlap-add**) to enable seamless stitching of separately processed windows.