Efficient convolution using the DFT

Digital Signal Processing with a focus on audio signals

Fabio Antonacci

February 18, 2026

Background: convolution theorem

- The convolution operation is at the basis of many signal processing tasks
- For instance, it enables the computation of the output of digital systems, provided the input signal and the impulse response:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Background: convolution theorem

- The convolution operation is at the basis of many signal processing tasks
- For instance, it enables the computation of the output of digital systems, provided the input signal and the impulse response:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

- An important theorem states that convolution is equivalent to a product in the frequency domain:

$$Y(e^{j\omega}) = X(e^{j\omega}) \cdot H(e^{j\omega})$$

Circular convolution

- A similar theorem exists also for the DFT (sampled version of the DTFT)

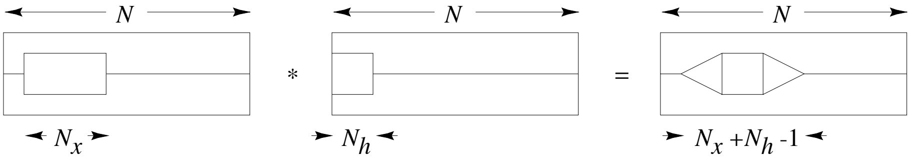
- As the DFT is implicitly periodic in both time and frequency domains, it turns that the convolution theorem involves the circular (cyclic) convolution. For N-points DFT we have, indeed:

$$Y(k) = X(k) \cdot H(k) \qquad \longleftrightarrow \qquad y(n) = \sum_{k=0}^{N-1} x(k) h(n-k)_N$$

Convolution of finite length signals

- In practical cases, performing the convolution in the frequency domain may be very efficient
- Let us consider the simplest case, i.e. that of convolving two finite-length signals:
  - $x(n)$, input signal with length N_x
  - $h(n)$, impulse response with length N_h

- The output signal $y(n) = x(n) * h(n)$ will be time-limited, too, with total length $N = N_x + N_h - 1$ samples, i.e.:



Convolution of finite length signals: computational complexity

- Computing the output signal means calculating the convolution sum:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Convolution of finite length signals: computational complexity

- Computing the output signal means calculating the convolution sum:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

- As the signals are time-limited, the total number of operations are finite. Specifically, we must compute a total of $(N_x + N_h - 1) \times N_h$ products

Convolution of finite length signals: computational complexity

- Computing the output signal means calculating the convolution sum:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

- As the signals are time-limited, the total number of operations are finite. Specifically, we must compute a total of $(N_x + N_h - 1) \times N_h$ products
- The complexity order of time-domain convolution can therefore be approximated as $O(N^2)$

Convolution of finite length signals: computational complexity

- Computing the output signal means calculating the convolution sum:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

- As the signals are time-limited, the total number of operations are finite. Specifically, we must compute a total of $(N_x + N_h - 1) \times N_h$ products
- The complexity order of time-domain convolution can therefore be approximated as $O(N^2)$
- **Idea**: what about computing the convolution in the frequency domain (simple product) instead of working in the time domain?

Convolution using the DFT: idea and motivations

- The DFT can be computed efficiently using the Fast Fourier Transform (FFT) algorithm

Convolution using the DFT: idea and motivations

- The DFT can be computed efficiently using the Fast Fourier Transform (FFT) algorithm
- Specifically, when the signal length N is a power of 2, the computational complexity of the FFT is $O(N \log N)$

Convolution using the DFT: idea and motivations

- The DFT can be computed efficiently using the Fast Fourier Transform (FFT) algorithm
- Specifically, when the signal length N is a power of 2, the computational complexity of the FFT is $O(N \log N)$
- We can therefore exploit the (circular) convolution theorem to speed-up the computation of convolution:
  - Compute FFT of $x(n)$ and $h(n)$: $O(N \log N)$
  - Compute the products $Y(k) = X(k)H(k)$: $O(N)$
  - Compute IFFT of $Y(k)$: $O(N \log N)$

Convolution using the DFT: idea and motivations

- The DFT can be computed efficiently using the Fast Fourier Transform (FFT) algorithm
- Specifically, when the signal length N is a power of 2, the computational complexity of the FFT is $O(N \log N)$
- We can therefore exploit the (circular) convolution theorem to speed-up the computation of convolution:
  - Compute FFT of $x(n)$ and $h(n)$: $O(N \log N)$
  - Compute the products $Y(k) = X(k)H(k)$: $O(N)$
  - Compute IFFT of $Y(k)$: $O(N \log N)$

Result: overall complexity of FFT-based convolution is $O(N \log N)$ (lower than in the time domain!)

Dealing with circular convolution

- The main issue about using the DFT is that cyclic convolution is involved

Dealing with circular convolution

- The main issue about using the DFT is that cyclic convolution is involved
- Suppose that the DFTs are of length N, the circular convolution can be rewritten as:

$$
\begin{aligned}
y(n) &= \sum_{k=0}^{N-1} x(k)h(n-k)_N \\
&= \sum_{k=0}^{n} x(k)h(n-k) + \sum_{k=n+1}^{N-1} x(k)h(n-k+N) \ , \quad \text{for } 0 \leq n \leq N-1
\end{aligned}
$$

- If we want the circular convolution to be equal to the linear a-cyclic convolution between $x(n)$ and $h(n)$, the second summation must be null, i.e.

$$
c(n) = \sum_{k=n+1}^{N-1} x(k)h(n-k+N) = 0 \ , \quad \text{for } 0 \leq n \leq N-1
$$

Dealing with circular convolution

- The main issue about using the DFT is that cyclic convolution is involved
- Suppose that the DFTs are of length N, the circular convolution can be rewritten as:

$$
\begin{aligned}
y(n) &= \sum_{k=0}^{N-1} x(k)h(n-k)_N \\
&= \sum_{k=0}^{n} x(k)h(n-k) + \sum_{k=n+1}^{N-1} x(k)h(n-k+N)\ , \quad \text{for } 0 \leq n \leq N-1
\end{aligned}
$$

Dealing with circular convolution

- The main issue about using the DFT is that cyclic convolution is involved
- Suppose that the DFTs are of length N, the circular convolution can be rewritten as:

$$
\begin{aligned}
y(n) &= \sum_{k=0}^{N-1} x(k)h(n-k)_N \\
&= \sum_{k=0}^{n} x(k)h(n-k) + \sum_{k=n+1}^{N-1} x(k)h(n-k+N)\ , \quad \text{for } 0 \leq n \leq N-1
\end{aligned}
$$

- If we want the circular convolution to be equal to the linear a-cyclic convolution between $x(n)$ and $h(n)$, the second summation must be null, i.e.

$$
c(n) = \sum_{k=n+1}^{N-1} x(k)h(n-k+N) = 0\ , \quad \text{for } 0 \leq n \leq N-1
$$

Dealing with circular convolution (cont')

- We know that:
  - $x(n)$ has duration N_x, so we set $x(n) = 0$ for $n \geq N_x$ and $n < 0$
  - $h(n)$ has duration N_h, so we set $h(n) = 0$ for $n \geq N_h$ and $n < 0$

Dealing with circular convolution (cont')

- We know that:
  - $x(n)$ has duration N_x, so we set $x(n) = 0$ for $n \geq N_x$ and $n < 0$
  - $h(n)$ has duration N_h, so we set $h(n) = 0$ for $n \geq N_h$ and $n < 0$

- Therefore, we can force the summation $c(n)$ to be identically zero if we select

$$N \geq N_x + N_h - 1$$

- Note that $N_x + N_h - 1$ is exactly the duration of the output signal $y(n) = x(n) * h(n)$

Dealing with circular convolution (cont')

- We know that:
  - $x(n)$ has duration N_x, so we set $x(n) = 0$ for $n \geq N_x$ and $n < 0$
  - $h(n)$ has duration N_h, so we set $h(n) = 0$ for $n \geq N_h$ and $n < 0$

- Therefore, we can force the summation $c(n)$ to be identically zero if we select

$$N \geq N_x + N_h - 1$$

- Note that $N_x + N_h - 1$ is exactly the duration of the output signal $y(n) = x(n) * h(n)$

- Considering N-length signals in place of their original lengths correspond to **zero-padding**, i.e. appending trailing zeros to reach the length N

Dealing with circular convolution (cont')

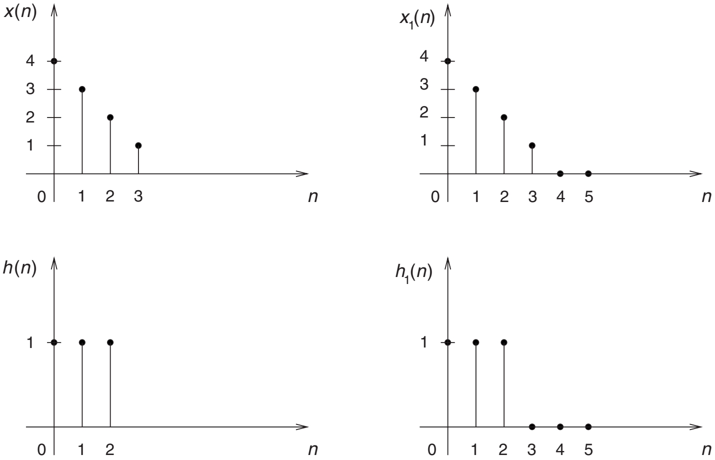
- We know that:
  - $x(n)$ has duration N_x, so we set $x(n) = 0$ for $n \geq N_x$ and $n < 0$
  - $h(n)$ has duration N_h, so we set $h(n) = 0$ for $n \geq N_h$ and $n < 0$
- Therefore, we can force the summation $c(n)$ to be identically zero if we select

$$N \geq N_x + N_h - 1$$

- Note that $N_x + N_h - 1$ is exactly the duration of the output signal $y(n) = x(n) * h(n)$
- Considering N-length signals in place of their original lengths correspond to **zero-padding**, i.e. appending trailing zeros to reach the length N
- Remark: to fully exploit the efficiency of the FFT algorithm, N should be selected as a power of 2

Zero-padding

Example of zero-padding of the two sequences to be convolved:



Zero-padding and time-domain aliasing

- Note that the condition $N \geq N_x + N_h - 1$ can be interpreted as an anti-aliasing one
- Indeed, if we add an insufficient number of zeros (i.e., the final signals length is $N < N_x + N_h - 1$), we have $c(n) \neq 0$ in general
- Recall that the DFT corresponds to sampling the DTFT: we must sample it at least at $N_x + N_h - 1$ points to avoid overlapping replicas in the time domain!
- If we don't add enough zeros, some of our convolution terms "wrap around" and add back upon others (due to modulo indexing): this generates **time domain aliasing**

Convolution using the DFT: algorithm

Convolution using the DFT: algorithm

1. Compute the N-points DFTs of $x(n)$ and $h(n)$:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \ldots N-1$$

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \ldots N-1$$

Convolution using the DFT: algorithm

1. Compute the N-points DFTs of $x(n)$ and $h(n)$:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \ldots N-1$$

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \ldots N-1$$

2. Compute the products (convolution in frequency domain):

$$Y(k) = X(k)H(k) \ , \quad k = 0, 1, \ldots N-1$$

Convolution using the DFT: algorithm

1. Compute the N-points DFTs of $x(n)$ and $h(n)$:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \dots N-1$$

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j\frac{2\pi kn}{N}} \ , \quad k = 0, 1, \dots N-1$$

2. Compute the products (convolution in frequency domain):

$$Y(k) = X(k)H(k) \ , \quad k = 0, 1, \dots N-1$$

3. Obtain the output signal via N-points IDFT:

$$y(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y(k) e^{j\frac{2\pi kn}{N}} \ , \quad n = 0, 1, \dots N-1$$

Convolution of indefinite length signals

- We saw that we can perform efficient linear convolution of two finite length sequences using Fourier based techniques

Convolution of indefinite length signals

- We saw that we can perform efficient linear convolution of two finite length sequences using Fourier based techniques
- There are some situations where it will not be practical to perform the convolution of two signals using a single FFT:
  - N_x is extremely large
  - Real time operation (we can't wait until the signal ends)

Convolution of indefinite length signals

- We saw that we can perform efficient linear convolution of two finite length sequences using Fourier based techniques
- There are some situations where it will not be practical to perform the convolution of two signals using a single FFT:
  - N_x is extremely large
  - Real time operation (we can't wait until the signal ends)
- Theoretically, there is no problem doing this with direct convolution: since $h(n)$ is finite-length, we only need to store the past N_h samples of the input signal
- However, this would require N_h products for each output sample

Overlap and add: overview

- For the above reason, filtering directly in the time domain can be extremely time-consuming when N_h is large
- Again, a more efficient solution is that of computing the convolution in the frequency domain

Overlap and add: overview

- For the above reason, filtering directly in the time domain can be extremely time-consuming when N_h is large
- Again, a more efficient solution is that of computing the convolution in the frequency domain
- **Idea**: use overlap-and-add to process the signal on a frame basis
  - processing one frame (segment) at a time
  - compute FFT convolution on each frame separately
  - put it all back together correctly (we'll see how to)

Overlap and add: overview

- For the above reason, filtering directly in the time domain can be extremely time-consuming when N_h is large
- Again, a more efficient solution is that of computing the convolution in the frequency domain

- **Idea**: use overlap-and-add to process the signal on a frame basis
  - processing one frame (segment) at a time
  - compute FFT convolution on each frame separately
  - put it all back together correctly (we'll see how to)

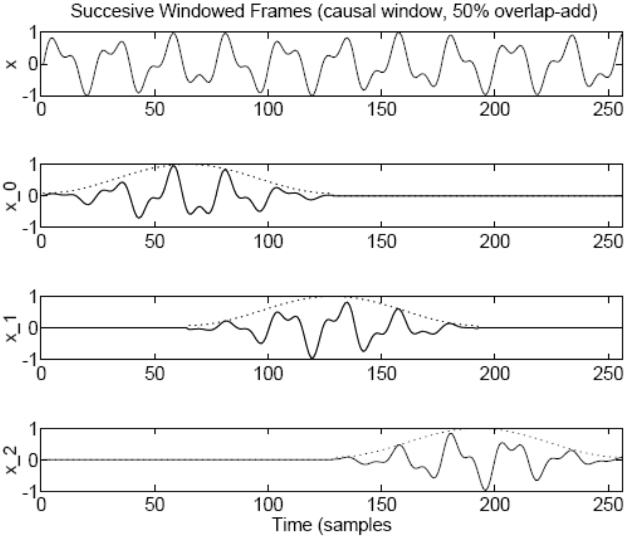
- **Frames** (or **segments**) from the input signal $x(n)$ are extracted as

$$x_m(n) \triangleq x(n)w(n - mR) \, , n \in (-\infty, +\infty) \, ,$$

  where:
  - $w(n)$ is a length M window (rectangular/Hamming/...)
  - m is the frame index
  - R is the hop-size

Overlap-and-add: overview (cont')

Example of frame extraction:



Constant overlap-and-add condition

- For this frame-by-frame spectral processing to work, we must be able to reconstruct $x(n)$ from the individual overlapping frames.

Constant overlap-and-add condition

- For this frame-by-frame spectral processing to work, we must be able to reconstruct $x(n)$ from the individual overlapping frames. This can be written as

$$x(n) = \sum_{m=-\infty}^{+\infty} x_m(n) = \sum_{m=-\infty}^{+\infty} x(n)w(n-mR) = x(n) \sum_{m=-\infty}^{+\infty} w(n-mR)$$

Constant overlap-and-add condition

- For this frame-by-frame spectral processing to work, we must be able to reconstruct $x(n)$ from the individual overlapping frames. This can be written as

$$x(n) = \sum_{m=-\infty}^{+\infty} x_m(n) = \sum_{m=-\infty}^{+\infty} x(n)w(n - mR) = x(n) \sum_{m=-\infty}^{+\infty} w(n - mR)$$

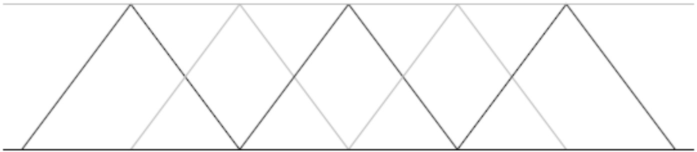
- It follows that

$$x(n) = \sum_{m=-\infty}^{+\infty} x_m(n) \iff \sum_{m=-\infty}^{+\infty} w(n - mR) = 1$$

- The last equality plays a key-role in overlap-and-add convolution, and it is called **Constant Overlap-And-Add** (COLA) **condition** on the analysis window $w(n)$

Constant overlap-and-add condition (cont')

- All windows which obey the COLA will yield perfect reconstruction of the original signal from the data frames by overlap-add
- There is no constraint on window type, only that the window overlap-adds to a constant for the hop size used
- Examples (where M is the generic window length):
  - Rectangular window at 0% overlap ($R = M$)
  - Rectangular window at 50% overlap ($R \approx M/2$, i.e., $R = M/2$ if M is even, $R = (M-1)/2$ if M is odd)
  - Hamming window at 50% overlap ($R \approx M/2$)
  - Hamming window at 75% overlap ($R \approx M/4$)
  - Triangular (Bartlett) window at 50% overlap (see figure below)
  - Any window with $R = 1$



Frequency-domain convolution via OLA

- Once a suitable window has been selected, the convolution process can be expressed as follows:

$$y(n) = x(n) * h(n) = \sum_{m=-\infty}^{\infty} x_m(n) * h(n)$$

- The last equation tells that the overall convolution operation can be separated into a summation of finite-length convolutions on signal segments
- The segment convolutions can be accomplished in the frequency domain, exploiting the efficiency of the FFT algorithm

Frequency-domain convolution via OLA (cont')

- Let's denote the result of the *m*th convolution as

$$y_m(n) = x_m(n) * h(n)$$

Frequency-domain convolution via OLA (cont')

- Let's denote the result of the *m*th convolution as

$$y_m(n) = x_m(n) * h(n)$$

- To compute the DFT, it is convenient to shift-back the frames to the origin. Let's define:

$$\tilde{x}_m(n) \triangleq \mathrm{SHIFT}_{mR}[x_m(n)] \triangleq x_m(n+mR)$$
$$\tilde{y}_m(n) \triangleq \mathrm{SHIFT}_{mR}[y_m(n)] \triangleq y_m(n+mR) = \tilde{x}_m(n) * h(n)$$

Frequency-domain convolution via OLA (cont')

- Let's denote the result of the mth convolution as

$$y_m(n) = x_m(n) * h(n)$$

- To compute the DFT, it is convenient to shift-back the frames to the origin. Let's define:

$$\tilde{x}_m(n) \triangleq \mathrm{SHIFT}_{mR}[x_m(n)] \triangleq x_m(n + mR)$$
$$\tilde{y}_m(n) \triangleq \mathrm{SHIFT}_{mR}[y_m(n)] \triangleq y_m(n + mR) = \tilde{x}_m(n) * h(n)$$

- This way, the signals are so that:
  - $\tilde{x}_m(n) = 0$ for $n < 0$ and $n \geq M$
  - $\tilde{y}_m(n) = 0$ for $n < 0$ and $n \geq N_h$

Frequency-domain convolution via OLA (cont')

- The mth convolution can be safely performed using N-points DFTs, paying attention to:
  - select $N \geq M + N_h - 1$ for avoiding aliasing problems
  - zero-pad both $\tilde{x}_m(n)$ and $h(n)$ to reach length N

Frequency-domain convolution via OLA (cont')

- The mth convolution can be safely performed using N-points DFTs, paying attention to:
  - select $N \geq M + N_h - 1$ for avoiding aliasing problems
  - zero-pad both $\tilde{x}_m(n)$ and $h(n)$ to reach length N
- The N-points DFTs are thus computed as

$$\tilde{X}_m(k) = \sum_{n=0}^{N-1} \tilde{x}_m(n) e^{-j\frac{2\pi kn}{N}}$$

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j\frac{2\pi kn}{N}}$$

Frequency-domain convolution via OLA (cont')

- The time-domain convolution corresponds to a multiplication in the frequency-domain:

$$\tilde{Y}_m(k) = \tilde{X}_m(k) \cdot H(k)$$

Frequency-domain convolution via OLA (cont')

- The time-domain convolution corresponds to a multiplication in the frequency-domain:

$$\tilde{Y}_m(k) = \tilde{X}_m(k) \cdot H(k)$$

- The time-domain segment is then recovered via N-points IDFT:

$$\tilde{y}_m(n) = \frac{1}{N} \sum_{n=0}^{N-1} \tilde{Y}_m(k) e^{j\frac{2\pi kn}{N}}$$

Frequency-domain convolution via OLA (cont')

- The time-domain convolution corresponds to a multiplication in the frequency-domain:

$$\tilde{Y}_m(k) = \tilde{X}_m(k) \cdot H(k)$$

- The time-domain segment is then recovered via N-points IDFT:

$$\tilde{y}_m(n) = \frac{1}{N} \sum_{n=0}^{N-1} \tilde{Y}_m(k) e^{j\frac{2\pi kn}{N}}$$

- The overall output signal is finally obtained by superposing all the filtered segments, remembering to shift back each segment to the right position:

$$y(n) = \sum_{m=-\infty}^{\infty} \text{SHIFT}_{-mR}\left[\tilde{y}_m(n)\right] = \sum_{m=-\infty}^{\infty} \tilde{y}_m(n - mR) = \sum_{m=-\infty}^{\infty} y_m(n)$$

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:
   - Obtain $\tilde{x}_m(n)$ by shifting $x_m(n)$ to the origin

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:
   - Obtain $\tilde{x}_m(n)$ by shifting $x_m(n)$ to the origin
   - Zero-pad $h(n)$ and $\tilde{x}_m(n)$ to length $N \geq M + N_h - 1$

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:
   - Obtain $\tilde{x}_m(n)$ by shifting $x_m(n)$ to the origin
   - Zero-pad $h(n)$ and $\tilde{x}_m(n)$ to length $N \geq M + N_h - 1$
   - Obtain $\tilde{y}_m(n)$ by performing alias-free convolution using the FFT

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:
   - Obtain $\tilde{x}_m(n)$ by shifting $x_m(n)$ to the origin
   - Zero-pad $h(n)$ and $\tilde{x}_m(n)$ to length $N \geq M + N_h - 1$
   - Obtain $\tilde{y}_m(n)$ by performing alias-free convolution using the FFT
   - Obtain $y_m(n)$ by shifting $\tilde{y}_m(n)$ back to the original position

Frequency-domain convolution via OLA: algorithm

Collecting the equations discussed before, we can summarize the overlap-and-add algorithm:

1. Initialize an output buffer $y(n)$ at zero
2. Divide $x(n)$ into length-N segments $x_m(n)$
3. For each block repeat:
   - Obtain $\tilde{x}_m(n)$ by shifting $x_m(n)$ to the origin
   - Zero-pad $h(n)$ and $\tilde{x}_m(n)$ to length $N \geq M + N_h - 1$
   - Obtain $\tilde{y}_m(n)$ by performing alias-free convolution using the FFT
   - Obtain $y_m(n)$ by shifting $\tilde{y}_m(n)$ back to the original position
   - Overlap and add $y_m(n)$ to the output buffer $y(n)$

Overlap-and-add example