Topics:

- Discrete Cosine Transform
- FFT
- Linear and Circular Convolution
- Rate Conversion

Includes review of Fourier transforms, properties of Fourier transforms, convolution, sampling

**Discrete Cosine Transform**

The Discrete Cosine Transform is widely used for audio and image compression and is the basis for MPEG (Motion Picture Experts Group) and JPEG compression standards.

- It is based on the DFT with imposed even symmetry through reflection so that the DCT is real valued.
- The symmetry created by reflection is also a better model for finite duration sequences than the DFT model of replication which considers sharp difference between the first and last sample the same as sharp differences in the middle of the sequence. The DCT is well matched to modeling the difference between the beginning and the end of a finite length sequence.
- The DCT can take advantage of the computational efficiency of fast Fourier transform algorithms, but it does not have the shift property of the DFT.

Consider a length N sequence \( x[n] \). It can be used to form a symmetric sequence of length 2N as the sum of the original sequence extended by N zeros and a reflected sequence preceded by zeros.

\[
\begin{align*}
x_1[n] &= x[0] \ x[1] \ldots \ x[N-1] \ 0 \ldots \ 0 \ 0 \\
x_2[n] &= 0 \ 0 \ldots \ 0 \ x[N-1] \ldots \ x[1] \ x[0] \\
x_3[n] &= x[0] \ x[1] \ldots \ x[N-1] \ x[N-1] \ldots \ x[1] \ x[0] = x_1[n] + x_2[n]
\end{align*}
\]

From the properties of the Discrete Fourier Transform:

- If \( x[n] \) is real, its transform has conjugate symmetry, i.e. \( X[k] = X^*[N-k] \).
- If \( x[n] \) has even symmetry, \( X[k] \) is real.

The transform of the complex conjugate of a time reversed sequence is the complex conjugate of the transform of the original sequence.

\[
X_1[k] = X_1^*[k] + e^{j\pi k/(2N)} X_1^*[k] = e^{j\pi k/(2N)} \left( e^{-j\pi k/(2N)} X_1[k] + e^{j\pi k/(2N)} X_1^*[k] \right)
\]
Although the DFT of this sequence will be of length 2N, except for the linear phase term for a half sample shift, it will be real valued and symmetric because the expression in parentheses is real valued and symmetric. Because of this, half of the transform values are redundant and only the first half of the 2N transform values are needed to define the transform. The equations for the transform values $C(k)$ are shown here. Note that there is a scale factor that depends on the index $k$.

$$C(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

$$x(n) = \sum_{k=0}^{N-1} \alpha(k) C(k) \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

$$\alpha(k) = \begin{cases} \sqrt{1/N} & k = 0 \\ \sqrt{2/N} & 1 \leq k \leq N - 1 \end{cases}$$

The basis functions of the DCT are real valued samples of the cosine function $\cos\left(\frac{\pi(2n+1)k}{2N}\right)$.

Discrete Cosine Transform basis functions for $N=8$ are shown below as bar plots. The left column has the basis functions for even values of $k$ with 0, 1, 2, and 3 cycles in the cosine. The right column has the basis functions for odd values of $k$ with 1/2, 1 1/2, 2 1/2, and 3 1/2 cycles in the cosine. The basis functions with the odd index values model the differences between the beginning and the end of a finite sequence.

In contrast, the Discrete Fourier Transform basis vectors assume periodic replication of the finite sequence and differences between the beginning and end of a finite sequence have the same effect as similar difference in the middle of the sequence. The transform equations are shown here.
### Analysis

\[ X(k) = \sum_{n=0}^{N-1} x(n)e^{-\frac{2\pi j nk}{N}} \]

### Synthesis

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

The real and imaginary part of the basis functions of the DFT are samples of the cosine and sine functions

\[
\cos \left( \frac{2\pi nk}{N} \right) \quad \text{and} \quad \sin \left( \frac{2\pi nk}{N} \right)
\]

The real part of the Discrete Fourier Transform basis functions for \( N=8 \) are shown below as bar plots. The left column has the basis functions for even values of \( k \) and the right column has the basis functions for odd values of \( k \). The \( k^{th} \) basis function will have \( k \) cycles in the cosine, and for \( k>N/2 \), aliasing will cause the basis functions to be the complex conjugates of functions with indices \( < N/2 \). That is for \( k> N/2 \), the index value \( k-N \) could be used and that value would be between \(-N/2 \) and 0. All of the basis function model periodic replication and none can model the differences between the beginning and the end of a finite sequence.

The Discrete Cosine Transform basis function values for \( N=4 \) are shown here as an array with \( n \) specifying the column and \( k \) specifying the row.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6533</td>
<td>0.2706</td>
<td>-0.2706</td>
<td>-0.6533</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5000</td>
<td>-0.5000</td>
<td>-0.5000</td>
<td>0.5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2706</td>
<td>-0.6533</td>
<td>0.6533</td>
<td>-0.2706</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In some applications an integer approximation also can work well for compression with a much lower computation requirement. For example, consider the approximation below.

\[
\begin{bmatrix}
13 & 13 & 13 & 13 \\
17 & 7 & -7 & -17 \\
13 & -13 & -13 & 13 \\
7 & -17 & 17 & -7
\end{bmatrix}
\]

**FFT**

The FFT computes the same values as the DFT, but the computation is structured to reuse partial results in an efficient manner. Direct implementation of DFT requires \(N^2\) multiplications, but using an FFT when \(N\) is a power of 2 reduces the number of multiplications to \(N(\log_2(N))\). Minimizing the number of multiplications has greater significance when the relative computational cost of multiplication exceeds other computational costs. The FFT has increased costs with respect to branching, nested loops, and data access.

There are many effective factorizations of the DFT leading to FFT algorithms. We will look at radix 2 (\(N=2^3\)) decimation in time structures to demonstrate the FFT approach. Radix 4 implementations or decimation in frequency are used often also. Note that:

- If \(N=2\), a DFT requires no multiplication because the transform equations only use 1 and -1 to multiply the data.
- If \(N\) is even, the sequence \(x[n]\) can be divided into two sequences of length \(N/2\) by putting the samples with an even index in a sequence \(x_0[m]=x[2m]\) and putting the samples with an odd index in a sequence \(x_1[m]=x[2m+1]\). Here \(0 \leq m < N/2\).
- The DFT of each sequence can be computed with \((N/2)^2\) multiplications and an additional \(N\) multiplication are needed to combine the two transforms to get the \(N\)-length DFT of the original sequence \(x[n]\).
- Thus, instead of \(N^2\) multiplications, this structure of computation requires only \(2((N/2)^2 + N)\) multiplications.
- If \(N/2\) is even, this process can be repeated to further reduce computation to \(2\{2(N/4)^2 +N/2\} + N\) multiplications.
- If \(N\) is a power of 2, this can be continued until all the DFTs are of length 2 which requires no multiplication.
- In summary
  - DFT of length \(N=2^q\) requires \(N^2\) multiplications.
  - One stage: Using two length \(N/2\) DFTs requires \(2((N/2)^2 + N) = N + N^2/2\) multiplications
  - Two stages: Using four length \(N/4\) DFTs requires \(2\{2(N/4)^2 +N/2\} + N = 2N + N^2/4\) multiplications
  - Three stages: Using eight length \(N/8\) DFTs requires \(2\{2[2(N/8)^2 +N/4]+N/2]\} + N = 3N + N^2/8\) multiplications
  - Using \(q\) stages requires \(qN = N(\log_2(N))\) multiplications.
One stage:

Two stages:
Linear and Circular Convolution

Convolution and Modulation Properties of the Discrete Time Fourier Transform and the Discrete Fourier Transform are similar. Convolution in time corresponds to multiplication in frequency, and vice versa. However there are some significant differences because the Discrete Time Fourier Transform is a relationship between a possibly infinite sequence of samples data and a frequency domain representation that is a periodic function of a continuous frequency variable while the DFT is a relationship between sampled periodic representations in both time and frequency.

TIME

Discrete Time Fourier Transform

Linear Convolution

\[ y[n] = h[n] * x[n] \]

\[ = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \]

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

Modulation or windowing

\[ g[n] = x_i[n]x[n] \]

FREQUENCY

Table 2.2, p58

\[ Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}) \]

Discrete Time Fourier Transform

Circular Convolution of length \( N \): \( 0 \leq n \leq N-1 \)

\[ y_c[n] = h[n] \otimes_N x[n] \]

\[ = \sum_{k=0}^{N-1} x[k]h[\langle n-k \rangle_N] \]

\[ = \sum_{m=0}^{N-1} h[m]x[\langle n-m \rangle_N] \]

Modulation or windowing

\[ g_c[n] = x_i[n]x[n] \]

Table 8.2 p660

\[ Y_c[k] = H[k]X[k] \]

\[ G_c[k] = \frac{1}{N} X_1[k] \otimes_N X_2[k] \]

\[ = \frac{1}{N} \sum_{\ell=0}^{N-1} X_1[\ell]X_2[\langle k-\ell \rangle_N] \]

Note: The circles “\( x \)” with subscript \( N \) is used to indicate a length \( N \) circular convolution.

Computation of linear convolution \( y[n] \) in the frequency domain with DTFT

- Compute \( H(e^{j\omega}) \) with DTFT
- Compute \( X(e^{j\omega}) \) with DTFT
- Compute product \( Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}) \)
- Compute \( y[n] \) from IDTFT of \( Y(e^{j\omega}) \)
For a practical implementation for arbitrary inputs, we can’t compute the DTFT, which is a function of the continuous frequency variable defined by the infinite duration sampled input sequence. We would need to sample the DTFT to compute \( y[n] \) over a finite range, so we should consider using the DFT instead.

But the DFT computes circular convolution, not linear convolution. How could we use it to compute the linear convolution? Consider the convolution of two finite sequences \( x[n] \) and \( h[n] \) where the sequence \( x[n] \) is defined for \( L \) samples and is otherwise zero and the sequence \( h[n] \) is defined for \( P \) samples and is otherwise zero.

- We can’t take an \( L \)-length DFT of \( x[n] \) and a \( P \)-length DFT of \( h[n] \) and then multiply them. When we multiply in frequency, the index \( k \) must represent the same frequency in both transforms, so the length of the transforms must be the same.
- We could append zeros to the shorter sequence to make it the same length as the longer sequence. If, for example, \( L>P \), we could compute \( X[k] \) as the \( L \)-length DFT of \( x[n] \) and \( H[k] \) as the \( L \)-length DFT of \( h[n] \) with \( (L-P) \) appended zeros. But the \( L \)-length IDFT of the product \( X[k]H[k] \) would be the \( L \)-length circular convolution of \( x[n] \) and \( h[n] \), not the \( L+P-1 \) length linear convolution we want.
- We could append \( (P-1) \) zeros to \( x[n] \) and \( (L-1) \) zeros to \( h[n] \) and compute \( (L+P-1) \) length DFTs of both. This would prevent overlap in the replications of \( y[n] \), so the circular and linear convolution would be the same.

**Example: \( L=4, P=3 \)**

Consider linear convolution, \( L \)-length circular convolution, and \( (L+P-1) \) length circular convolution.

- Linear convolution:
  \[
  y[n] = h[n] * x[n] = \sum_{k=0}^{3} x[k]h[n-k] = \sum_{m=0}^{2} h[m]x[n-m]
  \]
  \[
  \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( y[n] )</th>
<th>( x[n] )</th>
<th>( h[n] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( h[0]x[0] )</td>
<td>( h[0] )</td>
<td>( h[0] )</td>
</tr>
<tr>
<td>1</td>
<td>( h[0]x[1] )</td>
<td>( h[0] ) + ( h[1] ) + ( h[2] )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( h[0]x[2] )</td>
<td>( h[0] ) + ( h[1] ) + ( h[2] )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( h[0]x[3] )</td>
<td>( h[0] ) + ( h[1] ) + ( h[2] )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( h[0]x[4] )</td>
<td>( h[0] ) + ( h[1] ) + ( h[2] )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( h[0]x[5] )</td>
<td>( h[0] ) + ( h[1] ) + ( h[2] )</td>
<td></td>
</tr>
</tbody>
</table>

In the shaded areas the index for \( x[n] \) is less than zero or greater than 3, so the index is outside the range of nonzero values of \( x[n] \) and the products in the shaded areas are zero and do not contribute anything to the output sums.
Circular convolution of length 4 with h[n] zero-extended by one with h[3]=0:
Because x[n] is considered a periodic sequence with a period of 4, in the linear convolution output sums above we have
\[ x[-1] = x[4-1] = x[3] \]
\[ x[-2] = x[4-2] = x[2] \]
\[ x[4] = x[4-4] = x[0] \]
so
and only two of the four values of the circular convolution are the desired linear convolution values.

Circular convolution of length 6 with h[n] zero-extended by (L-1) = 3 zeros and x[n] zero extended by (P-1)=2 zeros:
Because x[n] is considered a periodic sequence with a period of 6, we have
\[ x[-1] = x[6-1] = x[5] = 0 \text{ from zero-entension} \]
\[ x[-2] = x[6-2] = x[4] = 0 \text{ from zero-entension} \]
\[ x[4] = 0 \text{ from zero-entension} \]
\[ x[5] = 0 \text{ from zero-entension} \]
so
\[ y_c[n] = y[n] \text{ for } 0 \leq n \leq 5 \]
and all 6 of the values of the circular convolution are the desired linear convolution values.

Using circular convolution to compute linear convolution.
Circular convolution can be used to compute linear convolution if enough zeros are appended so that replicas of the linear convolution do not overlap. This method can also be used convolve long data sequences using short segments or blocks of the input data x[n]. Refer to the linear convolution example above for the following discussion.

For the convolution of two finite sequences x[n] and h[n] where the sequence x[n] is defined for L samples and is otherwise zero and the sequence h[n] is defined for P samples and is otherwise zero, we have shown that the process to compute a linear convolution is:
- Append (P-1) zeros to x[n] and compute the length (L+P-1) DFT.
- Append (L-1) zeros to h[n] and compute the length (L+P-1) l DFT.
- Compute the point-by-point product of the two DFTs.
- Compute the IDFT of the product divided by (L+P-1) to get the linear convolution y[n].
For input sequences where $x[n]$ is much longer than $h[n]$, zero extension of $h[n]$ may not be reasonable. Instead the input sequence can be divided into blocks and methods for combining the block convolution outputs can be developed. There are two frequently used methods called overlap-add and overlap-save.

- **Overlap Add**
  - Use a block length of $N=L+P-1$
  - Append $(L-1)$ zeros to $h[n]$ and compute the $N$-length DFT $H[k]$ once. It will be used for all blocks.
  - Start the input block index at 0.
  - Repeat the following:
    - Get the next length $L$ sequence of input from $x[n]$ starting at the block index
    - Append $(P-1)$ zeros and compute the $N$-length DFT $X[k]$.
    - Compute the $N$-point IDFT of $(H[k]X[k]/N)$ to get a partial output sequence.
    - Overlap the last partial output sequence with the current output sequence by adding the last $(P-1)$ outputs of the last partial output sequence to the first $(P-1)$ outputs of the current partial output sequence.
    - Output the first $L$ outputs of the sum.
    - Save the remaining $(P-1)$ outputs for use with the next block of $L$ input values.

- **Overlap Save**
  - Use a block length of $N=L+P-1$
  - Append $(L-1)$ zeros to $h[n]$ and compute the $N$-length DFT $H[k]$ once. It will be used for all blocks.
  - Start the input block index at 0.
  - Initialize the current $x[n]$ to all zeros.
  - Repeat the following:
    - Get the next length $L$ sequence of input from $x[n]$ starting at the block index
    - Store it into the last $L$ locations of $x[n]$.
    - Compute the $N$-length DFT $X[k]$.
    - Compute the $N$-point IDFT of $(H[k]X[k]/N)$ to get a temporary output sequence with some valid results and some invalid results.
    - Output the $L$ valid results for linear convolution at the end of the temporary output sequence and discard the first $(P-1)$ invalid results. (Refer to linear convolution example above.)
    - Move the last $(P-1)$ values in $x[n]$ to the first $(P-1)$ entries to be used again in the next block computation.
Rate Conversion

There is often a need for sample rate conversion to match inputs and outputs of systems with different rates or to adjust after processing. For example, a received signal might contain 4 channels of information multiplexed into one signal and its sampling rate would be selected for the total bandwidth needed for the four channels. After demultiplexing to separate the channels, the sampling rate for each individual channel could be reduced by a factor or 4. The sampling rate may be increase to allow simpler design of analog output filters or to allow several signals to be multiplexed.

How can it be done? Theoretically there should be no problem if there is no aliasing at either the initial sampling rate or the new sampling rate. The continuous signal can theoretically be perfectly recovered from the original sampled signal. Then the recovered continuous signal can be resampled at the new desired rate.

In practice the conversion can be done more efficiently for the specific rates.

Up-Sampling by Factor L

This involves two steps.

- The original sample sequence has \((L-1)\) zeroes inserted between each sample. The effectively increases the sampling rate, but does not change the spectrum of the original sampled signal because only zeros have been added which change nothing.
- Because there are now spectral images from the old sampling rate that should be removed at the new sampling rate. An interpolation filter is needed to suppress them.

Two Interpolation Functions

\[
h_1[n] = \begin{cases} \frac{L-|n|}{L} & -L \leq n \leq L \\ 0 & \text{else} \end{cases}
\]

\[
h_2[n] = \sin(\frac{\pi n}{L}) \frac{\sin(\frac{\pi n}{L})}{\frac{\pi n}{L}}
\]

\[
H_1(e^{j\omega}) = \frac{1}{L} \left( \frac{\sin(\omega L/2)}{\sin(\omega/2)} \right)^2
\]

\[
H_2(e^{j\omega}) = \begin{cases} 1 & \text{if } 0 \leq |\omega| < \pi / L \\ 0 & \text{else} \end{cases}
\]

Effect of Interpolators

Interpolation with sinc completely suppresses \(L-1\) of \(L\) replicated spectral images due to up-sampling. This creates spectrum of original signal sampled at \(T/L\).

Linear interpolation attenuates higher frequencies of base-band image and leaves some parts of other images. It works well only if the maximum frequency is \(<< p/L\).
Down-Sampling by a factor of M

The sampled signal is resampled keeping every Mth sample.

This causes spectral replication and low pass filtering of the original sampled signal may be needed to avoid aliasing.

Noninteger Rate Conversions

x[n] is up-sampled by a factor of L and filtered by H_u(z), a low-pass interpolating filter with cutoff frequency $\pi/L$.

Effective sampling rate is $L/T$

The interpolated output is filtered by $H_d(z)$, a low-pass decimating filter with cutoff frequency $\pi / M$. Then the filtered signal is down-sampled by a factor of M.

Effective sampling rate is $L/MT$

Quantization;

6 dB per Bit

- Increasing the number of bits by one reduces the power of the error signal by 6 dB.
- Assuming uncorrelated quantization errors, the average power of $e[n]$ is $(1/12) \times 2^{-2B}$.
- $10\log_{10}( (1/12) \times 2^{-2B} ) = c +10(-2B)\log_{10}(2)$
  $= c - 6B$