- Discrete Cosine Transform.
- Fast Fourier Transform.
- Convolution.
- Power spectrum.

- If the function $f(x)$ is even (i.e. symmetric) about the midpoint $(x = \frac{L}{2})$ then one can write the cosine series:

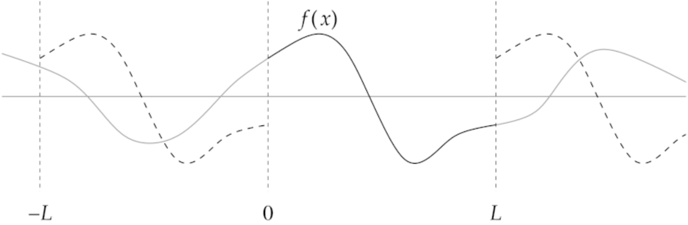
$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\pi k x}{L}\right)$$

- If the function $f(x)$ is even (i.e. symmetric) about the midpoint $(x = \frac{L}{2})$ then one can write the cosine series:

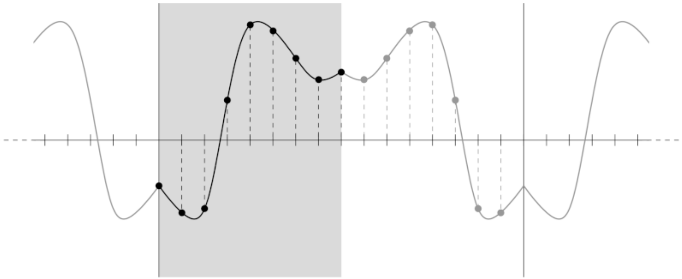
$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\pi k x}{L}\right)$$

- This might seem like a big limitation making the whole cosine transform virtually useless – but this is not the case.

- We had made any function periodic – say if we are only interested in a portion of this non periodic function over a finite interval, 0 to L, we can just take that portion and repeat it to create a periodic function.

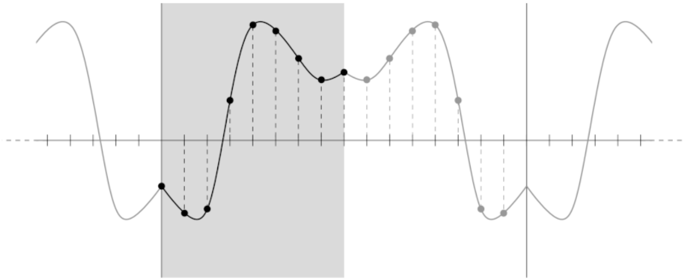


- If we are interested in the function in a finite region, we can make it symmetric by adding to it a mirror image of itself and then repeating it endlessly!

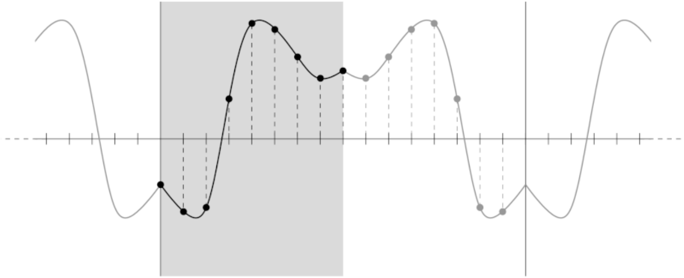


Discrete Cosine Transform – even functions

- If we are interested in the function in a finite region, we can make it symmetric by adding to it a mirror image of itself and then repeating it endlessly!
- In practice, this is how the cosine transform is always used.



- If we are interested in the function in a finite region, we can make it symmetric by adding to it a mirror image of itself and then repeating it endlessly!
- In practice, this is how the cosine transform is always used.
- This also implies that the number of samples in the transform is always even.



Discrete Cosine Transform (DCT)

Discrete Cosine Transform is a special case of Discrete Fourier Transform.

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

Discrete Cosine Transform (DCT)

Discrete Cosine Transform is a special case of Discrete Fourier Transform.

$$
\begin{aligned}
c_k &= \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \\
&= \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=\frac{1}{2}N+1}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)
\end{aligned}
$$

Discrete Cosine Transform is a special case of Discrete Fourier Transform.

$$\begin{aligned} c_k &= \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \\ &= \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=\frac{1}{2}N+1}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \\ &= \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=\frac{1}{2}N+1}^{N-1} y_{N-n} \exp\left(i\frac{2\pi k(N-n)}{N}\right) \end{aligned}$$

Because the function is symmetric $y_0 = y_N, y_1 = y_{N-1}, \ldots$ and $e^{i2\pi k} = 1$ for all $k \in \mathbb{Z}$

Changing variables $N - n \to n$ in the right hand expression:

$$c_k = \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=1}^{\frac{1}{2}N-1} y_n \exp\left(i\frac{2\pi kn}{N}\right)$$

Changing variables $N - n \to n$ in the right hand expression:

$$c_k = \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=1}^{\frac{1}{2}N-1} y_n \exp\left(i\frac{2\pi kn}{N}\right)$$

$$= y_0 + y_{N/2}\cos\left(\frac{2\pi k(N/2)}{N}\right) + 2\sum_{n=1}^{\frac{1}{2}N-1} y_n \cos\left(\frac{2\pi kn}{N}\right)$$

Changing variables $N - n \to n$ in the right hand expression:

$$
\begin{aligned}
c_k =& \sum_{n=0}^{\frac{1}{2}N} y_n \exp\left(-i\frac{2\pi kn}{N}\right) + \sum_{n=1}^{\frac{1}{2}N-1} y_n \exp\left(i\frac{2\pi kn}{N}\right) \\
=& y_0 + y_{N/2} \cos\left(\frac{2\pi k(N/2)}{N}\right) + 2\sum_{n=1}^{\frac{1}{2}N-1} y_n \cos\left(\frac{2\pi kn}{N}\right)
\end{aligned}
$$

Normally the cosine transform is applied to real samples, which implies that the coefficients c_k will all be real as well (as they are sums of real terms).

Discrete Cosine Transform (DCT)

As y_n and c_k are real, $c_{N-r} = c_r^* = c_r$. The inverse/ transform:

$$y_n = \frac{1}{N}\left[\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right]$$

Discrete Cosine Transform (DCT)

As y_n and c_k are real, $c_{N-r} = c_r^* = c_r$. The inverse/ transform:

$$
\begin{aligned}
y_n =& \frac{1}{N}\left[\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right]
\end{aligned}
$$

Discrete Cosine Transform (DCT)

As y_n and c_k are real, $c_{N-r} = c_r^* = c_r$. The inverse/ transform:

$$
\begin{aligned}
y_n =& \frac{1}{N}\left[\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_{N-k} \exp\left(-i\frac{2\pi(N-k)n}{N}\right.\right.
\end{aligned}
$$

As y_n and c_k are real, $c_{N-r} = c_r^* = c_r$. The inverse/ transform:

$$
\begin{aligned}
y_n =& \frac{1}{N}\left[\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_{N-k} \exp\left(-i\frac{2\pi(N-k)n}{N}\right)\right. \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=1}^{\frac{1}{2}N-1} c_k \exp\left(-i\frac{2\pi kn}{N}\right)\right]
\end{aligned}
$$

Discrete Cosine Transform (DCT)

As y_n and c_k are real, $c_{N-r} = c_r^* = c_r$. The inverse/ transform:

$$
\begin{aligned}
y_n =& \frac{1}{N}\left[\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=\frac{1}{2}N+1}^{N-1} c_{N-k} \exp\left(-i\frac{2\pi(N-k)n}{N}\right)\right] \\
=& \frac{1}{N}\left[\sum_{k=0}^{\frac{1}{2}N} c_k \exp\left(i\frac{2\pi kn}{N}\right) + \sum_{k=1}^{\frac{1}{2}N-1} c_k \exp\left(-i\frac{2\pi kn}{N}\right)\right] \\
=& \frac{1}{N}\left[c_0 + c_{N/2}\cos\left(\frac{2\pi(N/2)n}{N}\right) + 2\sum_{k=1}^{\frac{1}{2}N-1} c_k \cos\left(\frac{2\pi kn}{N}\right)\right]
\end{aligned}
$$

Discrete Cosine Transform (DCT)

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).

Discrete Cosine Transform (DCT)

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).
- Thus, one can say that this transform is its own inverse.

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).
- Thus, one can say that this transform is its own inverse.
- There is another commonly used form of this transform where the sample points are in the middle of the sample interval.

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).
- Thus, one can say that this transform is its own inverse.
- There is another commonly used form of this transform where the sample points are in the middle of the sample interval.
- A nice feature of this DCT is that unlike DFT, it does not assume that the samples are periodic.

Discrete Cosine Transform (DCT)

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).
- Thus, one can say that this transform is its own inverse.
- There is another commonly used form of this transform where the sample points are in the middle of the sample interval.
- A nice feature of this DCT is that unlike DFT, it does not assume that the samples are periodic.
- This is much better suited for non periodic functions as there is no discontinuity introduced.

Discrete Cosine Transform (DCT)

- The forward and reverse transforms are actually the same mathematical expression (but for the $1/N$ factor).
- Thus, one can say that this transform is its own inverse.
- There is another commonly used form of this transform where the sample points are in the middle of the sample interval.
- A nice feature of this DCT is that unlike DFT, it does not assume that the samples are periodic.
- This is much better suited for non periodic functions as there is no discontinuity introduced.
- In principle, the discrete sine transform can also be computed. However, the requirement of anti-symmetry forces the function to be zero at either end of the range. This does not happen often in real-world applications...

Fast Fourier Transform

- The Discrete Fourier Transform is defined as:

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

Fast Fourier Transform

- The Discrete Fourier Transform is defined as:

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

- The naive wave of doing the transform would involve: for each c_k, one has to perform N complex multiplications and $(N-1)$ additions – i.e. $2N-1$ complex operations – however as complex multiplications are much more expensive than complex additions, we only worry about complex multiplications.

Fast Fourier Transform

- The Discrete Fourier Transform is defined as:

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

- The naive wave of doing the transform would involve: for each c_k, one has to perform N complex multiplications and $(N-1)$ additions – i.e. $2N-1$ complex operations – however as complex multiplications are much more expensive than complex additions, we only worry about complex multiplications.
- Since there are N c_k's so the total number of operations is $\mathcal{O}(N^2)$.

Fast Fourier Transform

- The Discrete Fourier Transform is defined as:

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

- The naive wave of doing the transform would involve: for each c_k, one has to perform N complex multiplications and $(N-1)$ additions – i.e. $2N-1$ complex operations – however as complex multiplications are much more expensive than complex additions, we only worry about complex multiplications.
- Since there are N c_k's so the total number of operations is $\mathcal{O}(N^2)$.
- Gauss came up with a trick to reduce the number of operations. Often the FFT is attributed to Cooley and Tukey – but Gauss used it in 1805 (when he was 28 yrs).

- The Fast Fourier Transform algorithm is simplest to understand when one applies it to cases when the number of samples, $N = 2^m$.

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

- The Fast Fourier Transform algorithm is simplest to understand when one applies it to cases when the number of samples, $N = 2^m$.
- Divide the sum into two equal sized groups – first group containing terms where n is even and second group containing terms where n is odd.

$$
\begin{aligned}
c_k &= \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi k n}{N}\right) \\
&= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i\frac{2\pi k(2r)}{N}\right) + \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp\left(-i\frac{2\pi k(2r+1)}{N}\right)
\end{aligned}
$$

- The Fast Fourier Transform algorithm is simplest to understand when one applies it to cases when the number of samples, $N = 2^m$.
- Divide the sum into two equal sized groups – first group containing terms where n is even and second group containing terms where n is odd.

$$
\begin{aligned}
c_k &= \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \\
&= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i\frac{2\pi k(2r)}{N}\right) + \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp\left(-i\frac{2\pi k(2r+1)}{N}\right) \\
&= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i\frac{2\pi kr}{\frac{1}{2}N}\right) + e^{-i2\pi k/N} \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp\left(-i\frac{2\pi kr}{\frac{1}{2}N}\right)
\end{aligned}
$$

Fast Fourier Transform

- The Fast Fourier Transform algorithm is simplest to understand when one applies it to cases when the number of samples, $N = 2^m$.
- Divide the sum into two equal sized groups – first group containing terms where n is even and second group containing terms where n is odd.

$$
\begin{aligned}
c_k &= \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \\
&= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i\frac{2\pi k(2r)}{N}\right) + \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp\left(-i\frac{2\pi k(2r+1)}{N}\right) \\
&= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i\frac{2\pi kr}{\frac{1}{2}N}\right) + e^{-i2\pi k/N} \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp\left(-i\frac{2\pi kr}{\frac{1}{2}N}\right) \\
&\equiv E_k + e^{-i2\pi k/N} O_k
\end{aligned}
$$

However, thanks to the periodicity of the DFT:

$$E_{k+\frac{1}{2}N} = E_k \quad O_{k+\frac{1}{2}N} = O_k$$

However, thanks to the periodicity of the DFT:

$$E_{k+\frac{1}{2}N} = E_k \quad O_{k+\frac{1}{2}N} = O_k$$

Therefore, we can write:

$$c_k = \begin{cases} E_k + e^{-i2\pi k/N} O_k & \text{for } 0 \leq k < \frac{1}{2}N \\ E_{k-\frac{1}{2}N} + e^{-i2\pi k/N} O_{k-\frac{1}{2}N} & \text{for } \frac{1}{2}N \leq k < N \end{cases}$$

However, thanks to the periodicity of the DFT:

$$E_{k+\frac{1}{2}N} = E_k \quad O_{k+\frac{1}{2}N} = O_k$$

Therefore, we can write:

$$c_k = \begin{cases} E_k + e^{-i2\pi k/N} O_k & \text{for } 0 \leq k < \frac{1}{2}N \\ E_{k-\frac{1}{2}N} + e^{-i2\pi k/N} O_{k-\frac{1}{2}N} & \text{for } \frac{1}{2}N \leq k < N \end{cases}$$

However we also know:

$$\begin{aligned} e^{-i2\pi(k+\frac{1}{2}N)/N} &= e^{-i2\pi k/N - i\pi} \\ &= e^{-i\pi} e^{-i2\pi k/N} \\ &= -e^{-i2\pi k/N} \end{aligned}$$

Then for $0 \leq k < \frac{1}{2}N$:

$$\begin{aligned} c_k =& E_k + e^{-i2\pi k/N} O_k \\ c_{k+\frac{1}{2}N} =& E_k - e^{-i2\pi k/N} O_k \end{aligned}$$

Then for $0 \leq k < \frac{1}{2}N$:

$$\begin{aligned} c_k &= E_k + e^{-i2\pi k/N} O_k \\ c_{k+\frac{1}{2}N} &= E_k - e^{-i2\pi k/N} O_k \end{aligned}$$

- This result, expresses the DFT of length N recursively in terms of two DFTs of size $N/2$. In addition, there are N multiplications (one for each c_k).

Then for $0 \leq k < \frac{1}{2}N$:

$$\begin{aligned} c_k &= E_k + e^{-i2\pi k/N} O_k \\ c_{k+\frac{1}{2}N} &= E_k - e^{-i2\pi k/N} O_k \end{aligned}$$

- This result, expresses the DFT of length N recursively in terms of two DFTs of size $N/2$. In addition, there are N multiplications (one for each c_k).
- The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs.

Then for $0 \leq k < \frac{1}{2}N$:

$$c_k = E_k + e^{-i2\pi k/N} O_k$$
$$c_{k+\frac{1}{2}N} = E_k - e^{-i2\pi k/N} O_k$$

- This result, expresses the DFT of length N recursively in terms of two DFTs of size $N/2$. In addition, there are N multiplications (one for each c_k).
- The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs.
- From the original N^2 multiplications, now one has $2(\frac{N}{2})^2 = \frac{N^2}{2}$ multiplications.

Then for $0 \leq k < \frac{1}{2}N$:

$$\begin{aligned} c_k =& E_k + e^{-i2\pi k/N} O_k \\ c_{k+\frac{1}{2}N} =& E_k - e^{-i2\pi k/N} O_k \end{aligned}$$

- This result, expresses the DFT of length N recursively in terms of two DFTs of size $N/2$. In addition, there are N multiplications (one for each c_k).
- The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs.
- From the original N^2 multiplications, now one has $2(\frac{N}{2})^2 = \frac{N^2}{2}$ multiplications.
- This procedure can be recursively repeated – leading to a scaling of $\mathcal{O}(N \log_2 N)$.

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.

Fast Fourier Transform

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.
- $N \to 2 \times (2 \times (N/4)^2 + N/2) + N = N^2/4 + 2N$ operations.

Fast Fourier Transform

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.
- $N \to 2 \times (2 \times (N/4)^2 + N/2) + N = N^2/4 + 2N$ operations.
- $N \to 2\times(2\times(2\times(N/8)^2+N/4)+N/2)+N = N^2/8+3N$ operations.

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.
- $N \to 2 \times (2 \times (N/4)^2 + N/2) + N = N^2/4 + 2N$ operations.
- $N \to 2\times(2\times(2\times(N/8)^2+N/4)+N/2)+N = N^2/8+3N$ operations.
- $\dots$

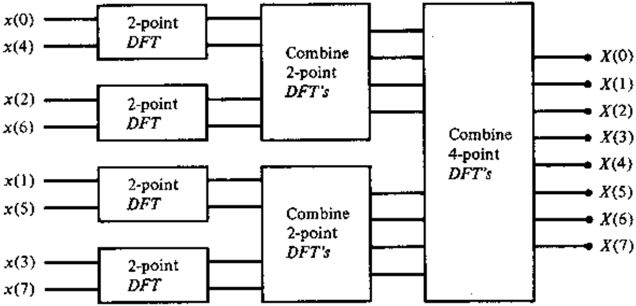
Fast Fourier Transform

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.
- $N \to 2 \times (2 \times (N/4)^2 + N/2) + N = N^2/4 + 2N$ operations.
- $N \to 2 \times (2 \times (2 \times (N/8)^2 + N/4) + N/2) + N = N^2/8 + 3N$ operations.
- $\ldots$
- $N \to N^2/2^m + mN$ where $m = \log_2 N$.

Fast Fourier Transform

- $N \to 2 \times (N/2)^2 + N = N^2/2 + N$ operations.
- $N \to 2 \times (2 \times (N/4)^2 + N/2) + N = N^2/4 + 2N$ operations.
- $N \to 2\times(2\times(2\times(N/8)^2+N/4)+N/2)+N = N^2/8+3N$ operations.
- $\ldots$
- $N \to N^2/2^m + mN$ where $m = \log_2 N$.
- $N \to N^2/N + N\log_2 N \sim \mathcal{O}(N\log_2 N)$.

Fast Fourier Transform



Discrete Fourier Transform

```
from numpy import zeros
from cmath import exp, pi

def dft(y):
    N = len(y)
    c = zeros(N//2 +1, complex)
    for k in range(N//2+1):
        for n in range(N):
            c[k] += y[n]*exp(-2j*pi*k*n/N)
return c
```

Discrete Fourier Transform

```python
import numpy as np
def DFT_slow(x):
    """Compute the discrete Fourier Transform"""
    x = np.asarray(x, dtype=float)
    N = x.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))
    M = np.exp(-2j * np.pi * k * n / N)
    return np.dot(M, x)
```

Fast Fourier Transform

```
def FFT(x):
    """A recursive implementation of the 1D Cooley-Tukey FFT"""
    x = np.asarray(x, dtype=float)
    N = x.shape[0]

    if N % 2 > 0:
        raise ValueError("size of x must be a power of 2")
    elif N <= 4:  # this cutoff should be optimized
        return DFT_slow(x)
    else:
        X_even = FFT(x[::2])
        X_odd = FFT(x[1::2])
        factor = np.exp(-2j * np.pi * np.arange(N) / N)
        return np.concatenate([X_even + factor[:N / 2] * X_odd,
                               X_even + factor[N / 2:] * X_odd])
```

Convolution

$$
\begin{aligned}
(f * g)(t) &= \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau \\
&= \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau
\end{aligned}
$$

The convolution theorem states that the Fourier transform of a convolution of two functions is the pointwise product of their Fourier transforms.

$$
\begin{aligned}
&\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \\
\implies\ & f * g = \mathcal{F}^{-1}\big\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\big\}
\end{aligned}
$$

Power Spectrum

- For a given signal, the power spectrum gives a plot of the portion of a signal's power (energy per unit time) falling within given frequency bins.

Power Spectrum

- For a given signal, the power spectrum gives a plot of the portion of a signal's power (energy per unit time) falling within given frequency bins.
- One method for estimating power spectral densities is based on using a function called the periodogram. The periodogram of an N-point sequence y_n is defined to be

$$I[k] = \frac{1}{N}\left|c_k\right|^2$$

Power Spectrum

- For a given signal, the power spectrum gives a plot of the portion of a signal's power (energy per unit time) falling within given frequency bins.
- One method for estimating power spectral densities is based on using a function called the periodogram. The periodogram of an N-point sequence y_n is defined to be

$$I[k] = \frac{1}{N}\left|c_k\right|^2$$

- It can be shown that the inverse transform of the periodogram is the sample autocorrelation function.

Power Spectrum

- For a given signal, the power spectrum gives a plot of the portion of a signal's power (energy per unit time) falling within given frequency bins.
- One method for estimating power spectral densities is based on using a function called the periodogram. The periodogram of an N-point sequence y_n is defined to be

$$I[k] = \frac{1}{N}\left|c_k\right|^2$$

- It can be shown that the inverse transform of the periodogram is the sample autocorrelation function.
- Parseval's theorem tells us:

$$\sum_{n=0}^{N-1} |y_n|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |c_k|^2$$