Nonuniform Fast Fourier Transforms

Tony Liu

Arizona State University

October 31, 2016
Overview

- Fourier Transform
  - Background
  - Discrete Fourier Transform
  - Fast Fourier Transform
- Nonuniform Fourier Transform
- Examples
Overview

- Fourier Transform
  - Background
  - Discrete Fourier Transform
  - Fast Fourier Transform
- Nonuniform Fourier Transform
- Examples
Fourier Transform Background

- Many physical phenomena involving waves including radar, light, and sound can be described using smooth, periodic functions.
- The goal is to be able to approximate the behavior of such phenomena.

![Figure: Tone generated by the touch tone button 1 (Moler, [1])](image)

- This allows us to investigate frequency properties of sample signals.
- Many applications such as tomography, MRI, SAR.
In order to approximate, we need to choose a set of basis functions. Naturally, this basis should consist of smooth, periodic, wave-like functions.

1. **Trigonometric Polynomials**
   
   \[ f(x) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kx) + b_k \sin(kx) \]

2. **Complex Basis Functions**
   
   \[ f(x) = \sum_{k=-\infty}^{\infty} c_k \phi(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx} \]
In order to approximate, we need to choose a set of basis functions

Naturally, this basis should consist of smooth, periodic, wave-like functions

**Trigonometric Polynomials**

\[
f(x) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kx) + b_k \sin(kx)
\]
Fourier Transform Background

- In order to approximate, we need to choose a set of basis functions.
- Naturally, this basis should consist of smooth, periodic, wave-like functions.
- **Trigonometric Polynomials**

  \[ f(x) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kx) + b_k \sin(kx) \]

- **Complex Basis Functions**

  \[ f(x) = \sum_{k=-\infty}^{\infty} c_k \phi(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx} \]
To calculate the coefficients, $c_k$, we perform the Fourier Transform

$$c_k = \int_{-\infty}^{\infty} f(x) e^{-ikx} \, dx$$

In the discrete problem with $N$ samples ($x_j$) we use the Discrete Fourier Transform (DFT) to find the set of Fourier coefficients, $X$:

$$X_k = \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi i kj}{N}}$$

The inverse transformation follows:

$$x_k = \frac{1}{N} \sum_{j=0}^{N-1} X_j e^{\frac{2\pi i kj}{N}}$$
Discrete Fourier Transform

\[ X_k = \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi ikj}{N}} \]

- Computation of the Fourier Transform requires:
  - \( O(N) \) multiplications and additions for each \( k = 0, \ldots, N-1 \)
  - Total of \( O(N^2) \) operations
- For large dimension problems, computation becomes time consuming
  - Solution: Fast Fourier Transform (FFT)
  - Total of \( O(N \log_2 N) \) operations
Suppose \( N \) is a power of 2

Define: \( \omega_N = e^{-\frac{2\pi i}{N}} \)

\[
X_k = \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi ikj}{N}}
\]

\[
= \sum_{j=0}^{N-1} x_j \omega_N^{kj}
\]

\[
= \sum_{j \text{ even}} x_j \omega_N^{kj} + \sum_{j \text{ odd}} x_j \omega_N^{kj}
\]

\[
= \sum_{j=0}^{N/2-1} x_{2j} \omega_N^{k2j} + \sum_{j=0}^{N/2-1} x_{2j+1} \omega_N^{k(2j+1)}
\]
Fast Fourier Transform

\[ X_k = \sum_{j=0}^{\frac{N}{2}-1} x_{2j} \omega_N^{kj} + \sum_{j=0}^{\frac{N}{2}-1} x_{2j+1} \omega_N^{k(2j+1)} \]

\[ = \sum_{j=0}^{\frac{N}{2}-1} x_{2j} (\omega_N^2)^{kj} + \omega_N^k \sum_{j=0}^{\frac{N}{2}-1} x_{2j+1} (\omega_N^2)^{kj} \]

Notice that \( \omega_N^2 = e^{\frac{-2(2\pi i)}{N}} = e^{\frac{-2\pi i}{N/2}} = \omega_{N/2}^2 \)

\[ X_k = \sum_{j=0}^{\frac{N}{2}-1} x_{2j} \omega_{N/2}^{kj} + \omega_N^k \sum_{j=0}^{\frac{N}{2}-1} x_{2j+1} \omega_{N/2}^{kj} \]
Essentially one N DFT can be written as the sum of 2, \((\frac{N}{2})\) DFTs:

\[ X_k = X_k^{\text{even}} + \omega_N^k X_k^{\text{odd}} \]

Furthermore, we notice that:

\[ \omega_{\frac{N}{2}}^{(k+N/2)j} = e^{\frac{-2\pi i (k+N/2)j}{N/2}} = e^{\frac{-2\pi i kj}{N/2}} = \omega_N^{kj} \]

\[ \omega_{N}^{(k+N/2)} = e^{\frac{-2\pi i (k+N/2)}{N}} = -e^{\frac{-2\pi i k}{N}} = -\omega_N^k \]

Thus, finally:

\[
X_{k+N/2} = \sum_{j=0}^{N/2-1} x_{2n} \omega_{\frac{N}{2}}^{(k+N/2)j} + \omega_N^{(k+N/2)} \sum_{j=0}^{N/2-1} x_{2j+1} \omega_{\frac{N}{2}}^{kj} \\
= \sum_{j=0}^{N/2-1} x_{2n} \omega_{\frac{N}{2}}^{kj} - \omega_N^k \sum_{j=0}^{N/2-1} x_{2j+1} \omega_{\frac{N}{2}}^{kj}
\]
Fast Fourier Transform

\[
X_k = \sum_{j=0}^{N/2-1} x_{2n} \omega_{N/2}^{kj} + \omega_N^k \sum_{j=0}^{N/2-1} x_{2j+1} \omega_{N/2}^{kj}
\]

\[
X_{k+N/2} = \sum_{j=0}^{N/2-1} x_{2n} \omega_{N/2}^{kj} - \omega_N^k \sum_{j=0}^{N/2-1} x_{2j+1} \omega_{N/2}^{kj}
\]

More compactly,

\[
X_k = X_k^{even} + \omega_N^k X_k^{odd}
\]

\[
X_{k+N/2} = X_k^{even} - \omega_N^k X_k^{odd}
\]
Consider $N=8$

\[
\begin{align*}
X_0 &= X_0^{\text{even}} + \omega_8^0 X_0^{\text{odd}} \\
X_1 &= X_1^{\text{even}} + \omega_8^1 X_1^{\text{odd}} \\
X_2 &= X_2^{\text{even}} + \omega_8^2 X_2^{\text{odd}} \\
X_3 &= X_3^{\text{even}} + \omega_8^3 X_3^{\text{odd}} \\
X_4 &= X_0^{\text{even}} - \omega_8^0 X_0^{\text{odd}} \\
X_5 &= X_1^{\text{even}} - \omega_8^1 X_1^{\text{odd}} \\
X_6 &= X_2^{\text{even}} - \omega_8^2 X_2^{\text{odd}} \\
X_7 &= X_3^{\text{even}} - \omega_8^3 X_3^{\text{odd}} 
\end{align*}
\]

$X^{\text{even}}, X^{\text{odd}}$ are obtained from 2, $\frac{N}{2}$ DFTs. To obtain $X$ then requires 1 multiplication and 1 addition/subtraction for a total of $2 \left(\frac{N}{2}\right)^2 + 2N$ operations.
Fast Fourier Transform

So how does this help?
We continue this process, recall $N$ is a power of 2:

$$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \frac{N}{8} \ldots \rightarrow \frac{N}{2^p} \quad p = \log_2(N)$$

In terms of cost:

$$N : N^2$$

$$\frac{N}{2} : 2 \left( \frac{N}{2} \right)^2 + 2N = \frac{N^2}{2} + 2N$$

$$\frac{N}{4} : 2 \left( 2 \left( \frac{N}{4} \right)^2 + N \right) + 2N = \frac{N^2}{4} + 4N$$

$$\frac{N}{8} : 2 \left( 2 \left( \frac{N}{8} \right)^2 + \frac{N}{2} \right) \right) + 2N = \frac{N^2}{8} + 6N$$

$$\vdots$$

$$\frac{N}{2^p} : \frac{N^2}{2^p} + 2pN = N + 2N\log_2(N) \approx O(N\log_2 N)$$
Computation of the Fourier Transform requires:

- $O(N)$ multiplications and additions for each $k = 0, \ldots, N-1$
- Total of $O(N^2)$ operations

For large dimension problems, computation becomes time consuming

- Solution: Fast Fourier Transform (FFT)
- Total of $O(N \log_2 N)$ operations
Example:

Figure: Tone generated by the touch tone button 1 (Moler, [1])

Figure: FFT of generated tone (Moler, [1])
Fast Fourier Transform

Example:

![FFT of generated tone](image1)

**Figure:** FFT of generated tone (Moler, [1])

![Touch tone pad frequencies](image2)

**Figure:** Touch tone pad frequencies (Moler, [1])
Overview

- Fourier Transform
  - Background
  - Discrete Fourier Transform
  - Fast Fourier Transform
- Nonuniform Fourier Transform
- Examples
Nonuniform Fourier Transform

- In previous background, we always had uniformly spaced time or frequencies
- Sometimes, we wish to obtain information at nonuniformly spaced time/frequencies
- In SAR/MRI: We collect data in nonuniform frequency space and we wish to solve the inverse problem and fit in time space. This optimization requires multiple iterations, at each of which we need to obtain data in a nonuniform frequency space.
- We still want to do this in an efficient manner such as the FFT as opposed to directly computing the DFT
- Cannot implement FFT since it required uniform space in order to lower computational cost
- To do this, we use a Nonuniform Fourier Transform (NUFFT) algorithm [Fessler, Sutton 2]
Suppose we are given signal samples $x_n$ for $n = 0, 1, \ldots, N - 1$. We have the corresponding DFT:

$$X(\omega) = \sum_{n=0}^{N-1} x_n e^{-i\omega n}$$

We wish to calculate the FT at a collection of $M$ nonuniformly spaced frequencies $\omega_m$:

$$X_m = X(\omega_m) = \sum_{n=0}^{N-1} x_n e^{-i\omega_m n}$$

We can directly evaluate this Fourier Transform. This would cost $O(MN)$ operations. Instead, to solve this in a more expedient manner, we derive the NUFFT.
Step 1: Calculate a K-point FFT, where $K \geq N$

$$Y_k = \sum_{n=0}^{N-1} s_n x_n e^{-i\gamma kn} \quad k = 0, 1, \ldots, K - 1$$

where $\gamma = \frac{2\pi}{K}$ and $s_n$ are scaling factors

Step 2: Approximate each $X_m$ by interpolating the $Y_k$'s in the form:

$$\hat{X}(\omega_m) = \sum_{k=0}^{K-1} v_{mk}^* Y_k \quad m = 1, 2, \ldots, M$$

The design problem of the NUFFT then becomes choosing these interpolators
Nonuniform Fourier Transform

An 'ideal' interpolator recovers the given sample $x_n$ using the IFFT and computes the desired values:

$$X(\omega_m) = \sum_{n=0}^{N-1} x_n e^{-i\omega_m n}$$

$$= \sum_{k=0}^{K-1} Y_k I(\omega/\gamma - k)$$

where:

$$I(\kappa) = e^{-i\gamma\kappa\eta_0} \frac{N}{K} \delta_n(\kappa), \quad \eta_0 = \frac{N - 1}{2}$$

$$\delta_n(\kappa) = \frac{1}{N} \sum_{n=0}^{N-1} e^{\pm i\gamma\kappa(n-\eta_0)} = \begin{cases} \frac{\sin(\pi\kappa N/K)}{N\sin(\pi\kappa/K)} & \kappa/K \notin \mathbb{Z} \\ 1 & \kappa/K \in \mathbb{Z} \end{cases}$$
Figure: Dirichlet Kernel
This is still not an improvement compared to direct computation

For each $\omega_m$, we interpolate using all Fourier coefficients, $Y_k$

Using this ideal interpolator then requires $O(MK)$ operations

To contain computation requirements, we constrain each $v_m$ to have at most $J$ nonzero elements corresponding to the $J$ nearest neighbors of $\omega_m$ in $\Omega_k = \{\gamma k\}$ for $k = 0, 1\ldots, K - 1$

This restriction lowers the computational cost down to $O(JK)$, $J << K$

Ex: $J=5$, $K=1024$
We choose the J nearest neighbors by:

\[ k_m = k_0(\omega_m) = \begin{cases} 
\left( \text{argmin}_{k \in \mathbb{Z}} |\omega_m - \gamma k| \right) - \frac{J+1}{2} & \text{J odd} \\
\left( \text{max}\{k \in \mathbb{Z} : \omega_m \geq \gamma k\} \right) - \frac{J}{2} & \text{J even}
\end{cases} \]

The interpolation formula then becomes:

\[ \hat{X}(\omega_m) = \sum_{j=1}^{J} Y_{\{k_m+j\}} u_j^*(\omega_m) \]

To choose the JM interpolator coefficients, \( \{u_j(\omega_m)\} \), we apply a min-max criterion.
The min-max criterion is defined as:

\[
\min_{u(\omega_m) \in \mathbb{C}^J} \max_{x \in \mathbb{C}^N: \|x\| \leq 1} |\hat{X}(\omega_m) - X(\omega_m)|
\]

We can derive the solution to this:

\[
|\hat{X}(\omega_m) - X(\omega_m)| = \left| \sum_{j=1}^{J} Y_{\{k_m+j\}k} u_j^*(\omega_m) - X(\omega_m) \right| \\
= \left| \sum_{j=1}^{J} u_j^*(\omega_m) \left[ \sum_{n=0}^{N-1} s_n x_n e^{-i\gamma (k_m+j)n} \right] - \sum_{n=0}^{N-1} x_n e^{-i\omega_m n} \right|
\]
The min-max criterion is defined as:

$$\min_{u(\omega_m) \in \mathbb{C}^J} \max_{x \in \mathbb{C}^N : \|x\| \leq 1} \left| \hat{X}(\omega_m) - X(\omega_m) \right|$$

We can derive the solution to this:

$$\left| \hat{X}(\omega_m) - X(\omega_m) \right| = \left| \sum_{j=1}^{J} Y_{\{k_m+j\} \sum_{j=1}^{J} k u_j^* (\omega_m) - X(\omega_m)} \right|$$

$$= \left| \sum_{j=1}^{J} u_j^* (\omega_m) \left[ \sum_{n=0}^{N-1} s_n x_n e^{-i \gamma (k_m+j)n} \right] - \sum_{n=0}^{N-1} x_n e^{-i \omega_m n} \right|$$

$$= \sqrt{N} \langle x, g(\omega_m) \rangle$$
Nonuniform Fourier Transform

\[
\begin{align*}
&= \left| \sum_{j=1}^{J} u_j^* (\omega_m) \left[ \sum_{n=0}^{N-1} s_n x_n e^{-i \gamma (k_m+j)n} \right] - \left[ \sum_{n=0}^{N-1} x_n e^{-i \omega_m n} \right] \right| \\
&= \sqrt{N} \langle x, g (\omega_m) \rangle \\
\end{align*}
\]

\[
\begin{align*}
g_n(\omega) &= s_n^* \left[ \sum_{j=1}^{J} \frac{1}{\sqrt{n}} e^{i \gamma (k_0(\omega)+j)n} u_j (\omega) \right] - \left[ \frac{1}{\sqrt{N}} e^{i \omega n} \right] \\
g (\omega) &= D (\omega) \left[ S^* \Lambda (\omega) u (\omega) - b (\omega) \right] \\
\end{align*}
\]

\[
\begin{align*}
D_{nn} (\omega) &= e^{i \omega \eta_0} e^{i \gamma k_0(\omega)(n-\eta_0)} \\
C_{nj} &= e^{i \gamma j(n-\eta_0) / \sqrt{N}} \\
\Lambda_{jj} (\omega) &= e^{i [\omega - \gamma (k_0(\omega+j))] \eta_0} \\
b_n (\omega) &= e^{i(\omega - \gamma k_0(\omega)) (\eta-\eta_0) / \sqrt{N}} \\
\end{align*}
\]
The min-max criterion is now defined as:

$$\min_{u(\omega) \in \mathbb{C}^J} \max_{x \in \mathbb{C}^N : \|x\| \leq 1} \sqrt{N} \langle x, g(\omega) \rangle$$

From the Cauchy-Schwartz Inequality:

$$\max_{x \in \mathbb{C}^N : \|x\| = 1} \langle x, g(\omega) \rangle = \|g(\omega)\|$$

Thus giving us:

$$\min_{u(\omega) \in \mathbb{C}^J} \sqrt{N} \|g(\omega)\| = \min_{u(\omega) \in \mathbb{C}^J} \sqrt{N} \|[S^*C \Lambda(\omega) u(\omega) - b(\omega)]\|$$

Solution (Pseudoinverse):

$$u(\omega) = \Lambda^*(\omega) [C^*S S^*C]^{-1} C^* S b(\omega)$$
To save even more computation:

- Precompute $C^*SS^*C$ since it is independent of frequency sample location.
- In many instances (optimization), we look to compute the NUFFT several times for the same set of nonuniform frequencies, $\omega_m$, but for different signals, $x$.
- We can precompute and store the JM interpolation coefficients, $u_j(\omega)$ in the case.
- Each NUFFT after this precomputation requires only $O(JM)$.
Overview

- Fourier Transform
  - Background
  - Discrete Fourier Transform
  - Fast Fourier Transform
- Nonuniform Fourier Transform
- Examples
Examples

Given a sample signal with $N=128$ points, we calculated an oversampled FFT, $K=256$. 
Examples

Given a sample signal with $N=128$ points, we calculated an oversampled FFT, $K=256$.
We then generated $M=128$ random frequencies to obtain Fourier coefficients at, interpolating with $J=5$ nearest neighbors.
Examples

Given a sample signal with $N=128$ points, we calculated an oversampled FFT, $K=256$. We then generated $M=128$ random frequencies to obtain Fourier coefficients at, interpolating with $J=5$ nearest neighbors.
Examples

Since the signal size is small here, we also directly computed the Fourier coefficients:
Examples

Since the signal size is small here, we also directly computed the Fourier coefficients:
Examples

Computation Time for $N = 128, K = 256, M = 128, J = 5$ :
- Time for Precomputation: 14.8ms
- Time to directly compute using DFT: 19.1ms
- Time to compute using NUFFT: .8ms
- NUFFT was 24x faster

Computation Time for $N = 1024, K = 2048, M = 1024, J = 5$ :
- Time for Precomputation: 6.1ms
- Time to directly compute using DFT: 922ms
- Time to compute using NUFFT: 6.1ms
- NUFFT was 150x faster

Computation Time for $N = 2^{14}, K = 2 \times 2^{14}, M = 2^{14}, J = 5$ :
- NUFFT was 80,000x faster
Overview

- Fourier Transform
  - Background
  - Discrete Fourier Transform
  - Fast Fourier Transform
- Nonuniform Fourier Transform
- Examples
Thank You