Numerical Methods I
Trigonometric Polynomials and the FFT

Aleksandar Donev
Courant Institute, NYU\textsuperscript{1}
donev@courant.nyu.edu

\textsuperscript{1}MATH-GA 2011.003 / CSCI-GA 2945.003, Fall 2014

Nov 13th, 2014
Consider now interpolating / approximating **periodic functions** defined on the interval $I = [0, 2\pi]$:

$$\forall x \quad f(x + 2\pi) = f(x),$$

as appear in practice when analyzing signals (e.g., sound/image processing).

Also consider only the space of complex-valued **square-integrable functions** $L^2_{2\pi}$,

$$\forall f \in L^2_{w} : \quad (f, f) = \|f\|^2 = \int_0^{2\pi} |f(x)|^2 \, dx < \infty.$$ 

Polynomial functions are not periodic and thus basis sets based on orthogonal polynomials are not appropriate.

Instead, consider sines and cosines as a basis function, combined together into **complex exponential functions**

$$\phi_k(x) = e^{ikx} = \cos(kx) + i \sin(kx), \quad k = 0, \pm 1, \pm 2, \ldots$$
It is easy to see that these are orthogonal with respect to the continuous dot product

$$\langle \phi_j, \phi_k \rangle = \int_{x=0}^{2\pi} \phi_j(x) \phi_k^*(x) dx = \int_0^{2\pi} \exp \left[ i(j - k)x \right] dx = 2\pi \delta_{ij}$$

The complex exponentials can be shown to form a complete trigonometric polynomial basis for the space $L^2_{2\pi}$, i.e.,

$$\forall f \in L^2_{2\pi} : \quad f(x) = \sum_{k=-\infty}^{\infty} \hat{f}_k e^{ikx},$$

where the Fourier coefficients can be computed for any frequency or wavenumber $k$ using:

$$\hat{f}_k = \frac{\langle f, \phi_k \rangle}{2\pi} = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx.$$
For a general interval \([0, X]\) the **discrete frequencies** are

\[ k = \frac{2\pi}{X} \kappa \quad \kappa = 0, \pm 1, \pm 2, \ldots \]

For non-periodic functions one can take the limit \(X \to \infty\) in which case we get **continuous frequencies**.

Now consider a **discrete Fourier basis** that only includes the first \(N\) basis functions, i.e.,

\[
\begin{align*}
    k &= -(N - 1)/2, \ldots, 0, \ldots, (N - 1)/2 & \text{if } N \text{ is odd} \\
    k &= -N/2, \ldots, 0, \ldots, N/2 - 1 & \text{if } N \text{ is even,}
\end{align*}
\]

and for simplicity we focus on \(N\) odd.

The least-squares **spectral approximation** for this basis is:

\[
f(x) \approx \phi(x) = \sum_{k=-(N-1)/2}^{(N-1)/2} \hat{f}_k e^{ikx}.
\]
Now also discretize a given function on a set of $N$ equi-spaced nodes

$$x_j = jh \text{ where } h = \frac{2\pi}{N}$$

where $j = N$ is the same node as $j = 0$ due to periodicity so we only consider $N$ instead of $N + 1$ nodes.

We also have the **discrete dot product** between two discrete functions (vectors) $f_j = f(x_j)$:

$$f \cdot g = h \sum_{j=0}^{N-1} f_j g_j^*$$

The discrete Fourier basis is **discretely orthogonal**

$$\phi_k \cdot \phi_{k'} = 2\pi \delta_{k,k'}$$
Proof of Discrete Orthogonality

The case $k = k'$ is trivial, so focus on

$$\phi_k \cdot \phi_{k'} = 0 \text{ for } k \neq k'$$

$$\sum_j \exp (ikx_j) \exp (-ik'x_j) = \sum_j \exp [i(\Delta k)x_j] = \sum_{j=0}^{N-1} [\exp (ih(\Delta k))]^j$$

where $\Delta k = k - k'$. This is a geometric series sum:

$$\phi_k \cdot \phi_{k'} = \frac{1 - z^N}{1 - z} = 0 \text{ if } k \neq k'$$

since $z = \exp (ih(\Delta k)) \neq 1$ and

$$z^N = \exp (ihN(\Delta k)) = \exp (2\pi i(\Delta k)) = 1.$$
The **Fourier interpolating polynomial** is thus easy to construct

\[ \phi_N(x) = \sum_{k=-(N-1)/2}^{(N-1)/2} \hat{f}_k e^{ikx} \]

where the **discrete Fourier coefficients** are given by

\[ \hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) \exp(-ikx_j) \]

Simplifying the notation and recalling \( x_j = jh \), we define the the **Discrete Fourier Transform** (DFT):

\[ \hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \exp \left( -\frac{2\pi ijk}{N} \right) \]
The set of discrete Fourier coefficients $\hat{f}$ is called the **discrete spectrum**, and in particular,

$$S_k = \left| \hat{f}_k \right|^2 = \hat{f}_k \hat{f}_k^*,$$

is the **power spectrum** which measures the frequency content of a signal.

If $f$ is real, then $\hat{f}$ satisfies the **conjugacy property**

$$\hat{f}_{-k} = \hat{f}_k^*,$$

so that half of the spectrum is redundant and $\hat{f}_0$ is real.

For an even number of points $N$ the largest frequency $k = -N/2$ does not have a conjugate partner.
Fourier Spectral Approximation

- **Discrete Fourier Transform (DFT):**

  \[
  \text{Forward } f \rightarrow \hat{f} : \quad \hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \exp \left( -\frac{2\pi ijk}{N} \right)
  \]

  \[
  \text{Inverse } \hat{f} \rightarrow f : \quad f(x_j) \approx \phi(x_j) = \sum_{k=-\frac{N-1}{2}}^{\frac{(N-1)/2}{}} \hat{f}_k \exp \left( \frac{2\pi ijk}{N} \right)
  \]

  - There is a very fast algorithm for performing the **forward and backward DFTs (FFT).**

  - There is **different conventions** for the DFT depending on the interval on which the function is defined and placement of factors of \( N \) and \( 2\pi \).

  Read the documentation to be consistent!
The Fourier interpolating polynomial $\phi(x)$ has **spectral accuracy**, i.e., exponential in the number of nodes $N$

$$\|f(x) - \phi(x)\| \sim e^{-N}$$

for **sufficiently smooth functions**.

Specifically, what is needed is sufficiently **rapid decay of the Fourier coefficients** with $k$, e.g., exponential decay $|\hat{f}_k| \sim e^{-|k|}$.

Discontinuities cause slowly-decaying Fourier coefficients, e.g., power law decay $|\hat{f}_k| \sim k^{-1}$ for **jump discontinuities**.

Jump discontinuities lead to slow convergence of the Fourier series for non-singular points (and no convergence at all near the singularity), so-called **Gibbs phenomenon** (ringing):

$$\|f(x) - \phi(x)\| \sim \begin{cases} N^{-1} & \text{at points away from jumps} \\ \text{const.} & \text{at the jumps themselves} \end{cases}$$
Gibbs Phenomenon

Approximation of a square wave timing signal ($f_0 = 20$ MHz)
Reconstruction of the periodic square waveform with 1, 3, 5, 7, 9 sinusoids
If we sample a signal at too few points the Fourier interpolant may be wildly wrong: **aliasing** of frequencies $k$ and $2k, 3k, \ldots$
Recall the transformation from real space to frequency space and back:

\[ f \rightarrow \hat{f} : \quad \hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \exp \left( -\frac{2\pi ijk}{N} \right), \quad k = -\frac{(N-1)}{2}, \ldots, \frac{(N-1)}{2} \]

\[ \hat{f} \rightarrow f : \quad f_j = \sum_{k=-\frac{(N-1)}{2}}^{\frac{(N-1)}{2}} \hat{f}_k \exp \left( \frac{2\pi ijk}{N} \right), \quad j = 0, \ldots, N - 1 \]

We can make the forward-reverse **Discrete Fourier Transform (DFT)** more symmetric if we shift the frequencies to \( k = 0, \ldots, N \):

**Forward** \( f \rightarrow \hat{f} : \quad \hat{f}_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} f_j \exp \left( -\frac{2\pi ijk}{N} \right), \quad k = 0, \ldots, N - 1 \)

**Inverse** \( \hat{f} \rightarrow f : \quad f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}_k \exp \left( \frac{2\pi ijk}{N} \right), \quad j = 0, \ldots, N - 1 \)
We can write the transforms in matrix notation:

\[
\hat{f} = \frac{1}{\sqrt{N}} U_N f \\
f = \frac{1}{\sqrt{N}} U_N^* \hat{f},
\]

where the **unitary Fourier matrix** \( \text{fft(eye}(N)) \) in MATLAB) is an \( N \times N \) matrix with entries

\[
u_{jk}^{(N)} = \omega_N^{jk}, \quad \omega_N = e^{-2\pi i / N}.
\]

A **direct** matrix-vector multiplication algorithm therefore takes \( O(N^2) \) multiplications and additions.

Is there a faster way to compute the **non-normalized**

\[
\hat{f}_k = \sum_{j=0}^{N-1} f_j \omega_N^{jk} \quad ?
\]
For now assume that $N$ is even and in fact a power of two, $N = 2^n$. The idea is to split the transform into two pieces, **even and odd** points:

$$
\sum_{j=2j'} f_j \omega_{N}^{jk} + \sum_{j=2j'+1} f_j \omega_{N}^{jk} = \sum_{j'=0}^{N/2-1} f_{2j'} (\omega_{N}^{2})^{j'k} + \omega_{N}^{k} \sum_{j'=0}^{N/2-1} f_{2j'+1} (\omega_{N}^{2})^{j'k}
$$

Now notice that

$$
\omega_{N}^{2} = e^{-4\pi i/N} = e^{-2\pi i/(N/2)} = \omega_{N/2}
$$

This leads to a **divide-and-conquer algorithm**:

$$
\hat{f}_k = \sum_{j'=0}^{N/2-1} f_{2j'} \omega_{N/2}^{jk} + \omega_{N}^{k} \sum_{j'=0}^{N/2-1} f_{2j'+1} \omega_{N/2}^{jk}
$$

$$
\hat{f}_k = U_N f = \left( U_{N/2} f_{\text{even}} + \omega_{N}^{k} U_{N/2} f_{\text{odd}} \right)
$$
FFT Complexity

- The **Fast Fourier Transform** algorithm is recursive:

  \[ \text{FFT}_N(f) = \text{FFT}_{N/2}(f_{\text{even}}) + w \boxdot \text{FFT}_{N/2}(f_{\text{odd}}), \]

  where \( w_k = \omega_N^k \) and \( \boxdot \) denotes element-wise product. When \( N = 1 \) the FFT is trivial (identity).

- To compute the whole transform we need \( \log_2(N) \) steps, and at each step we only need \( N \) multiplications and \( N/2 \) additions at each step.

- The total **cost of FFT** is thus much better than the direct method’s \( O(N^2) \): **Log-linear**

  \[ O(N \log N). \]

- Even when \( N \) is not a power of two there are ways to do a similar **splitting** transformation of the large FFT into many smaller FFTs.

- Note that there are different **normalization conventions** used in different software.
In MATLAB

- The forward transform is performed by the function \( \hat{f} = \texttt{fft}(f) \) and the inverse by \( f = \texttt{ifft}(\hat{f}) \). Note that \( \texttt{ifft(fft(f))} = f \) and \( f \) and \( \hat{f} \) may be complex.

- In MATLAB, and other software, the frequencies are not ordered in the “normal” way \(-\frac{N-1}{2}\) to \(\frac{N-1}{2}\), but rather, the nonnegative frequencies come first, then the positive ones, so the “funny” ordering is

\[
0, 1, \ldots, \frac{N-1}{2}, -\frac{N-1}{2}, -\frac{N-1}{2} + 1, \ldots, -1.
\]

This is because such ordering (shift) makes the forward and inverse transforms symmetric.

- The function \texttt{fftshift} can be used to order the frequencies in the “normal” way, and \texttt{ifftshift} does the reverse:

\[
\hat{f} = \texttt{fftshift(fft(f))} \text{ (normal ordering)}.
\]
Fast Fourier Transform

FFT-based noise filtering (1)

Fs = 1000; % Sampling frequency
dt = 1/Fs; % Sampling interval
L = 1000; % Length of signal
t = (0:L−1)*dt; % Time vector
T=L*dt; % Total time interval

% Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t)); % Sinusoids plus noise

figure(1); clf; plot(t(1:100),y(1:100),'b--'); hold on
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time')
if (0)
    N=(L/2)*2; % Even N
    y_hat = fft(y(1:N));
    % Frequencies ordered in a funny way:
    f_funny = 2*pi/T* [0:N/2−1, −N/2:−1];
    % Normal ordering:
    f_normal = 2*pi/T* [−N/2 : N/2−1];
else
    N=(L/2)*2−1; % Odd N
    y_hat = fft(y(1:N));
    % Frequencies ordered in a funny way:
    f_funny = 2*pi/T* [0:(N−1)/2, −(N−1)/2:−1];
    % Normal ordering:
    f_normal = 2*pi/T* [−(N−1)/2 : (N−1)/2];
end
Fast Fourier Transform

FFT-based noise filtering (3)

```matlab
figure(2); clf; plot(f_funny, abs(y_hat), 'ro'); hold on;

y_hat=fftshift(y_hat);
figure(2); plot(f_normal, abs(y_hat), 'b-');

title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('Power')

y_hat(abs(y_hat)<250)=0; % Filter out noise
y_filtered = ifft(ifftshift(y_hat));
figure(1); plot(t(1:100), y_filtered(1:100), 'r-')
```
Fast Fourier Transform

**FFT results**

Signal Corrupted with Zero-Mean Random Noise

Single-Sided Amplitude Spectrum of \( y(t) \)

A. Donev (Courant Institute)

Lecture X

11/2014 23 / 41
Applications of FFTs

• Because FFT is a very fast, almost linear algorithm, it is used often to accomplish things that are not seemingly related to function approximation.

• Denote the Discrete Fourier transform, computed using FFTs in practice, with

\[
\hat{f} = \mathcal{F}(f) \quad \text{and} \quad f = \mathcal{F}^{-1}\left(\hat{f}\right). 
\]

• Plain FFT is used in signal processing for digital filtering: Multiply the spectrum by a filter \(\hat{S}(k)\) discretized as \(\hat{s} = \left\{\hat{S}(k)\right\}_k\):

\[
f_{\text{filt}} = \mathcal{F}^{-1}\left(\hat{s} \odot \hat{f}\right) = f \ast s,
\]

where \(\ast\) denotes convolution, to be described shortly.

• Examples include low-pass, high-pass, or band-pass filters. Note that aliasing can be a problem for digital filters.
Convolution

- For continuous function, an important type of operation found in practice is **convolution** of a (periodic) function $f(x)$ with a (periodic) **kernel** $K(x)$:

\[
(K \ast f)(x) = \int_0^{2\pi} f(y)K(x - y)\,dy = (f \ast K)(x).
\]

- It is not hard to prove the **convolution theorem**:

\[
\mathcal{F}(K \ast f) = \mathcal{F}(K) \cdot \mathcal{F}(f).
\]

- Importantly, this remains true for **discrete convolutions**:

\[
(K \ast f)_j = \frac{1}{N} \sum_{j'=0}^{N-1} f_{j'} \cdot K_{j-j'} \quad \Rightarrow
\]

\[
\mathcal{F}(K \ast f) = \mathcal{F}(K) \cdot \mathcal{F}(f) \quad \Rightarrow \quad K \ast f = \mathcal{F}^{-1}(\mathcal{F}(K) \cdot \mathcal{F}(f))
\]
Proof of Discrete Convolution Theorem

Assume that the normalization used is a factor of $N^{-1}$ in the forward and no factor in the reverse DFT:

$$\mathcal{F}^{-1} (\mathcal{F}(K) \cdot \mathcal{F}(f)) = K \otimes f$$

$$[\mathcal{F}^{-1} (\mathcal{F}(K) \cdot \mathcal{F}(f))]_k = \sum_{k=0}^{N-1} \hat{f}_k \hat{K}_k \exp \left( \frac{2\pi ijk}{N} \right) =$$

$$N^{-2} \sum_{k=0}^{N-1} \left( \sum_{l=0}^{N-1} f_l \exp \left( -\frac{2\pi ilk}{N} \right) \right) \left( \sum_{m=0}^{N-1} K_m \exp \left( -\frac{2\pi imk}{N} \right) \right) \exp \left( \frac{2\pi ijk}{N} \right) =$$

$$N^{-2} \sum_{l=0}^{N-1} f_l \sum_{m=0}^{N-1} K_m \sum_{k=0}^{N-1} \exp \left[ \frac{2\pi i (j - l - m) k}{N} \right]$$
Recall the key discrete orthogonality property

\[ \forall \Delta k \in \mathbb{Z} : \quad N^{-1} \sum_j \exp \left[ i \frac{2\pi}{N} j \Delta k \right] = \delta_{\Delta k} \quad \Rightarrow \]

\[ N^{-2} \sum_{l=0}^{N-1} f_l \sum_{m=0}^{N-1} K_m \sum_{k=0}^{N-1} \exp \left[ \frac{2\pi i (j - l - m) k}{N} \right] = N^{-1} \sum_{l=0}^{N-1} f_l \sum_{m=0}^{N-1} K_m \delta_{j-l-m} \]

\[ = N^{-1} \sum_{l=0}^{N-1} f_l K_{j-l} = (K \ast f)_j \]

Computing convolutions requires 2 forward FFTs, one element-wise product, and one inverse FFT, for a total cost $N \log N$ instead of $N^2$. 
Spectral Derivative

- Consider approximating the derivative of a periodic function \( f(x) \), computed at a set of \( N \) equally-spaced nodes, \( f \).
- One way to do it is to use the **finite difference approximations**:

\[
f'(x_j) \approx \frac{f(x_j + h) - f(x_j - h)}{2h} = \frac{f_{j+1} - f_{j-1}}{2h}.
\]

- In order to achieve spectral accuracy of the derivative, we can differentiate the spectral approximation: **Spectral derivative**

\[
f'(x) \approx \phi'(x) = \frac{d}{dx} \phi(x) = \frac{d}{dx} \left( \sum_{k=0}^{N-1} \hat{f}_k e^{ikx} \right) = \sum_{k=0}^{N-1} \hat{f}_k \frac{d}{dx} e^{ikx}
\]

\[
\phi' = \sum_{k=0}^{N-1} \left( ik \hat{f}_k \right) e^{ikx} = \mathcal{F}^{-1} \left( i\hat{f} \otimes k \right)
\]

- Differentiation, like convolution, becomes multiplication in Fourier space.
DFTs and FFTs generalize straightforwardly to higher dimensions due to separability: Transform each dimension independently

\[
\hat{f} = \frac{1}{N_x N_y} \sum_{j_y=0}^{N_y-1} \sum_{j_x=0}^{N_x-1} f_{j_x,j_y} \exp \left[ -\frac{2\pi i (j_x k_x + j_y k_y)}{N} \right]
\]

\[
\hat{f}_{k_x,k_y} = \frac{1}{N_x} \sum_{j_y=0}^{N_y-1} \exp \left( -\frac{2\pi i j_y k_x}{N} \right) \left[ \frac{1}{N_y} \sum_{j_y=0}^{N_y-1} f_{j_x,j_y} \exp \left( -\frac{2\pi i j_y k_y}{N} \right) \right]
\]

For example, in two dimensions, do FFTs of each column, then FFTs of each row of the result:

\[
\hat{f} = \mathcal{F}_{\text{row}} \left( \mathcal{F}_{\text{col}} (f) \right)
\]

The cost is \(N_y\) one-dimensional FFTs of length \(N_x\) and then \(N_x\) one-dimensional FFTs of length \(N_y\):

\[
N_x N_y \log N_x + N_x N_y \log N_y = N_x N_y \log (N_x N_y) = N \log N
\]
The need for wavelets

- Fourier basis is great for analyzing periodic signals, but is not good for functions that are **localized in space**, e.g., brief bursts of speech.
- Fourier transforms are not good with handling **discontinuities** in functions because of the Gibbs phenomenon.
- Fourier polynomials **assume periodicity** and are not as useful for non-periodic functions.
- Because Fourier basis is not localized, the highest frequency present in the signal must be used everywhere: One cannot use **different resolutions in different regions of space**.
An example wavelet
Wavelet basis

- A **mother wavelet function** $W(x)$ is a localized function in space. For simplicity assume that $W(x)$ has compact support on $[0, 1]$.

- A **wavelet basis** is a collection of wavelets $W_{s,\tau}(x)$ obtained from $W(x)$ by **dilation** with a **scaling factor** $s$ and **shifting** by a **translation factor** $\tau$:

  $$W_{s,\tau}(x) = W(sx - \tau).$$

- Here the scale plays the role of frequency in the FT, but the shift is novel and localized the basis functions in space.

- We focus on **discrete wavelet basis**, where the scaling factors are chosen to be powers of 2 and the shifts are integers:

  $$W_{j,k} = W(2^jx - k), \quad k \in \mathbb{Z}, \ j \in \mathbb{Z}, \ j \geq 0.$$
Haar Wavelet Basis

\[ \psi_{0,0} = \psi(x) \]

\[ \psi_{1,0} = \psi(2x) \]

\[ \psi_{1,1} = \psi(2x - 1) \]

\[ \psi_{2,0} = \psi(4x) \]

\[ \psi_{2,1} = \psi(4x - 1) \]

\[ \psi_{2,2} = \psi(4x - 2) \]

\[ \psi_{2,3} = \psi(4x - 3) \]
Wavelet Transform

- Any function can now be represented in the wavelet basis:

\[ f(x) = c_0 + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j - 1} c_{jk} W_{j,k}(x) \]

This representation picks out frequency components in different spatial regions.

- As usual, we truncate the basis at \( j < J \), which leads to a total number of coefficients \( c_{jk} \):

\[ \sum_{j=0}^{J-1} 2^j = 2^J \]
Similarly, we discretize the function on a set of $N = 2^J$ equally-spaced nodes $x_{j,k}$ or intervals, to get the vector $f$:

$$f = c_0 + \sum_{j=0}^{J-1} \sum_{k=0}^{2^j-1} c_{jk} W_{j,k}(x_{j,k}) = W_j c$$

In order to be able to quickly and stably compute the coefficients $c$ we need an **orthogonal wavelet basis**:

$$\int W_{j,k}(x) W_{l,m}(x) dx = \delta_{j,l} \delta_{k,m}$$

The Haar basis is discretely orthogonal and computing the transform and its inverse can be done using a **fast wavelet transform**, in **linear time** $O(N)$ time.
Scaleogram
Wavelets

Another scaleogram

![Wavelet Scaleogram](image-url)
For the Haar basis, the wavelet approximation

\[ \phi(x) = c_0 + \sum_{j=0}^{J-1} \sum_{k=0}^{2^j-1} c_{jk} W_{j,k}(x) \]

is piecewise constant on each of the \( N \) sub-intervals of \([0, 1] \).

It is desirable to construct wavelet basis for which:

- The basis is orthogonal.
- One can exactly represent linear functions (differentiable).
- One can compute the forward and reverse wavelet transforms efficiently.

Constructions of such basis start from a father wavelet function \( \phi(x) \):

\[ \phi(x) = \sum_{k=0}^{N} c_k \phi(2x - k), \text{ and } W(x) = \sum_{k=1-N}^{1} (-1)^k c_{1-k} \phi(2x - k) \]
Wavelets
Mother and Father Wavelets

![Graph showing wavelet functions](image)

Daubechies 4 tap wavelet

- Scaling function
- Wavelet function
Periodic functions can be approximated using basis of orthogonal trigonometric polynomials.

The Fourier basis is discretely orthogonal and gives spectral accuracy for smooth functions.

Functions with discontinuities are not approximated well: Gibbs phenomenon.

The Discrete Fourier Transform can be computed very efficiently using the Fast Fourier Transform algorithm: $O(N \log N)$.

FFTs can be used to filter signals, to do convolutions, and to provide spectrally-accurate derivatives, all in $O(N \log N)$ time.

For signals that have different properties in different parts of the domain a wavelet basis may be more appropriate.

Using specially-constructed orthogonal discrete wavelet basis one can compute fast discrete wavelet transforms in time $O(N)$. 