

Scientific Computing: The Fast Fourier Transform

Aleksandar Donev
Courant Institute, NYU¹
donev@courant.nyu.edu

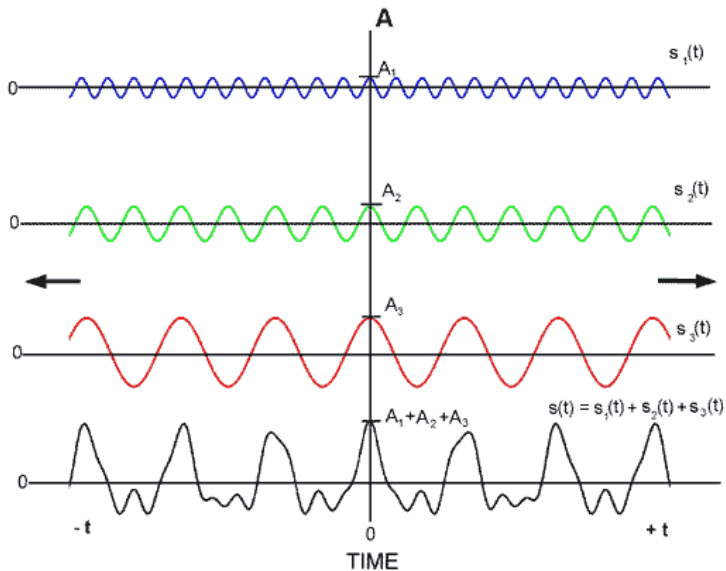
¹Course MATH-GA.2043 or CSCI-GA.2112, Fall 2015

October 29th, 2015

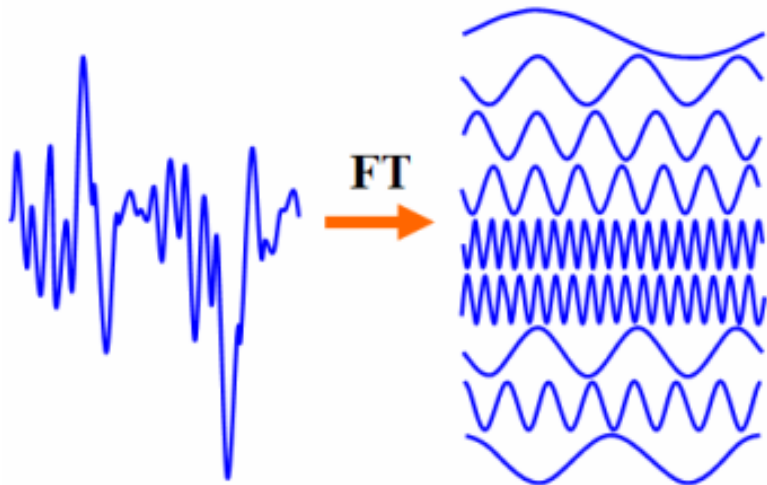
Outline

- 1 Fourier Series
- 2 Discrete Fourier Transform
- 3 Fast Fourier Transform
- 4 Applications of FFT
- 5 Wavelets
- 6 Conclusions

Fourier Composition



Fourier Decomposition



Periodic Functions

- 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).

- We will consider functions that are **square-integrable functions** (function space $L^2_{2\pi}$), $\int_0^{2\pi} |f(x)|^2 dx < \infty$.
- 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**

$$\phi_k(x) = e^{ikx} = \cos(kx) + i \sin(kx), \quad k = 0, \pm 1, \pm 2, \dots$$

Here k is the **frequency** or wavenumber.

Fourier Basis

$$\phi_k(x) = e^{ikx}, \quad k = 0, \pm 1, \pm 2, \dots$$

- The complex exponentials can be shown to form a complete **trigonometric polynomial basis** for the space $L^2_{2\pi}$, i.e., any square-integrable function can be represented in an infinite **Fourier series**:

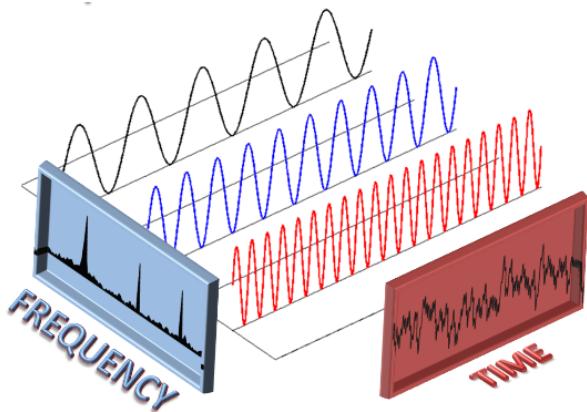
$$\forall f \in L^2_{2\pi} : \quad f(x) = \sum_{k=-\infty}^{\infty} \hat{f}_k e^{ikx},$$

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

$$\hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx.$$

This is the **continuous Fourier Transform**: mapping a function from **real** (time) to **Fourier** (frequency) space.

Fourier Decomposition



Discrete Fourier Basis

- Let us discretize a given function on a set of N **equi-spaced nodes** as a vector

$$\mathbf{f}_j = f(x_j) \quad \text{where} \quad x_j = jh \quad \text{and} \quad h = \frac{2\pi}{N}.$$

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

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

$$\begin{cases} k = -(N-1)/2, \dots, 0, \dots, (N-1)/2 & \text{if } N \text{ is odd} \\ k = -N/2, \dots, 0, \dots, N/2 - 1 & \text{if } N \text{ is even,} \end{cases}$$

and for simplicity we focus on N odd and denote the largest frequency with

$$K = (N-1)/2.$$

Fourier Interpolant

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

- We will try to approximate periodic functions with a **truncated Fourier series**:

$$f(x) \approx \phi(x) = \sum_{k=-K}^K \phi_k(x) = \sum_{k=-K}^K \hat{f}_k e^{ikx}.$$

- The discrete Fourier basis is $\{\phi_{-K}, \dots, \phi_K\}$,

$$(\phi_k)_j = \frac{1}{\sqrt{N}} \exp(ikx_j),$$

and it is an **orthonormal basis** in which we can represent periodic functions,

$$\phi_k \cdot \phi_{k'} = \delta_{k,k'}$$

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
 $z^N = \exp(ihN(\Delta k)) = \exp(2\pi i(\Delta k)) = 1$.

Fourier Matrix

- Let us collect the discrete Fourier basis functions as columns in a **unitary** $N \times N$ **matrix** (`fft(eye(N))` in MATLAB)

$$\Phi_N = [\phi_{-K} | \dots | \phi_0 | \dots | \phi_K] \Rightarrow \Phi_{jk}^{(N)} = \exp(2\pi ijk/N)$$

- The truncated Fourier series is

$$\mathbf{f} = \Phi_N \hat{\mathbf{f}}.$$

- Since the matrix Φ_N is unitary, we know that $\Phi_N^{-1} = \Phi_N^*$ and therefore

$$\hat{\mathbf{f}} = \Phi_N^* \mathbf{f},$$

which is nothing more than a change of basis!

Discrete Fourier Transform

- We can make the expressions more symmetric if we shift the frequencies to $k = 0, \dots, N$, but one should still think of half of the frequencies as “negative” and half as “positive”.
See MATLAB’s functions *fftshift* and *ifftshift*
- The **Discrete Fourier Transform** (DFT) is a change of basis taking us from real/time to Fourier/frequency domain:

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

$$\text{Inverse } \hat{\mathbf{f}} \rightarrow \mathbf{f} : \quad f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}_k \exp\left(\frac{2\pi ijk}{N}\right), \quad j = 0, \dots, N-1$$

- A **direct** matrix-vector multiplication algorithm therefore takes $O(N^2)$ multiplications and additions. **Can we do it faster?**

Discrete spectrum

- The set of discrete Fourier coefficients $\hat{\mathbf{f}}$ 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.

Spectral Accuracy (or not)

- The Fourier interpolating polynomial $\phi(x)$ has **spectral accuracy**, i.e., exponential in the number of nodes N

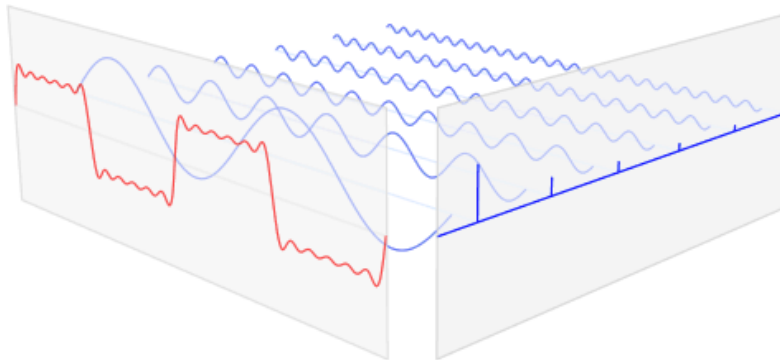
$$\|f(x) - \phi(x)\| \sim e^{-N}$$

for **sufficiently smooth functions**.

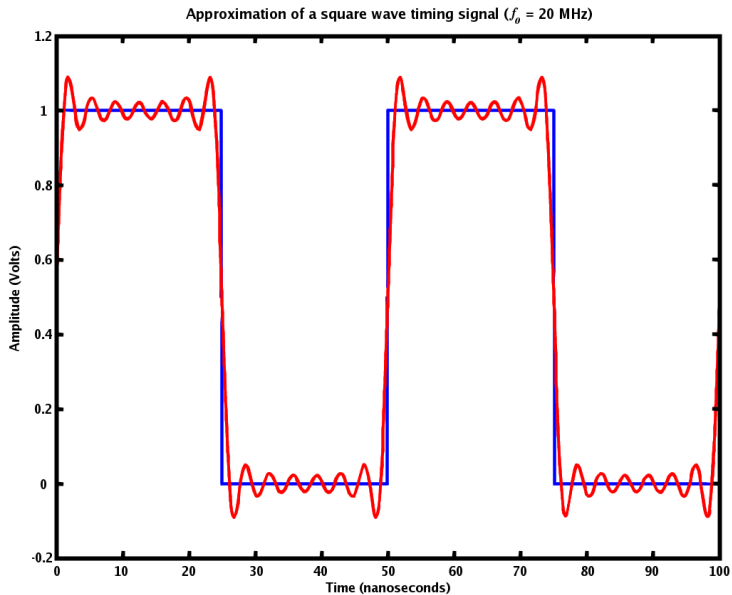
- Specifically, what is needed is sufficiently **rapid decay of the Fourier coefficients** with k , e.g., exponential decay $|\hat{f}_k| \sim e^{-|k|}$.
- Discontinuities cause slowly-decaying Fourier coefficients, e.g., power law decay $|\hat{f}_k| \sim k^{-1}$ for **jump discontinuities**.
- Jump discontinuities 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

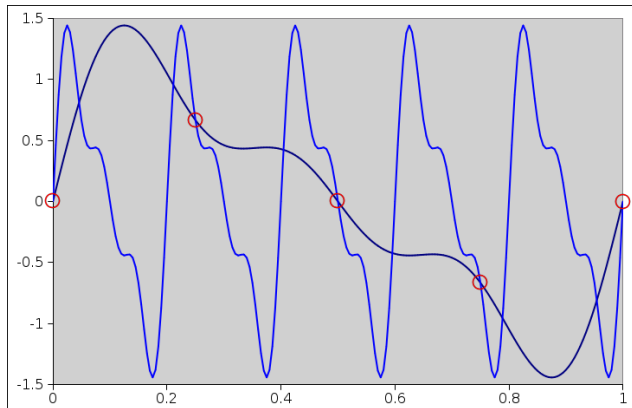


Gibbs Phenomenon



Aliasing

If we sample a signal at too few points the Fourier interpolant may be wildly wrong: **aliasing** of frequencies k and $2k, 3k, \dots$



Standard anti-aliasing rule is the **Nyquist–Shannon** criterion (theorem):
 Need **at least 2 samples per period**.

Fast Fourier Transform

- For now assume that N is even and in fact a power of two, $N = 2^n$. Denote the **root of unity**

$$\omega_N = e^{-2\pi i/N}.$$

- 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**:

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

$$\hat{f}_k = \Phi_N^* \mathbf{f} = \left(\Phi_{N/2}^* \mathbf{f}_{\text{even}} + \omega_N^k \Phi_{N/2}^* \mathbf{f}_{\text{odd}} \right)$$

which is called the **Fast Fourier Transform**.

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

$$FFT_N(\mathbf{f}) = FFT_{\frac{N}{2}}(\mathbf{f}_{\text{even}}) + \mathbf{w} \square FFT_{\frac{N}{2}}(\mathbf{f}_{\text{odd}}),$$

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

FFT Complexity

- 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 (we used $1/\sqrt{N}$ in both forward and inverse FFT to make them symmetric).

In MATLAB

- The forward transform is performed by the function $\hat{f} = \text{fft}(f)$ and the inverse by $f = \text{fft}(\hat{f})$. Note that $\text{ifft}(\text{fft}(f)) = f$ and f and \hat{f} may be complex.
- In MATLAB, and other software, the frequencies are not ordered in the “normal” way $-(N-1)/2$ to $+(N-1)/2$, but rather, the nonnegative frequencies come first, then the positive ones, so the “funny” ordering is

$$0, 1, \dots, (N-1)/2, \quad -\frac{N-1}{2}, -\frac{N-1}{2} + 1, \dots, -1.$$

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

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

$$\hat{f} = \text{fftshift}(\text{fft}(f)) \text{ (normal ordering).}$$

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')

```

FFT-based noise filtering (2)

```

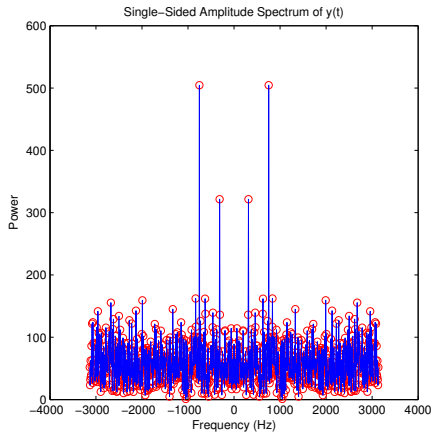
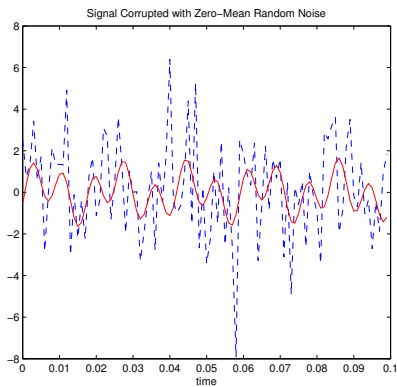
if(0)
    N=(L/2)*2; % Even N
    y_hat = fft(y(1:N));
    % Frequencies ordered in a funny way:
    f_funny = 2*pi/T* [0:N/2-1, -N/2:-1];
    % Normal ordering:
    f_normal = 2*pi/T* [-N/2 : N/2-1];
else
    N=(L/2)*2-1; % Odd N
    y_hat = fft(y(1:N));
    % Frequencies ordered in a funny way:
    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

```

FFT-based noise filtering (3)

```
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-')
```


FFT results



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

$$\hat{\mathbf{f}} = \mathcal{F}(\mathbf{f}) \quad \text{and} \quad \mathbf{f} = \mathcal{F}^{-1}(\hat{\mathbf{f}}).$$

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

$$\mathbf{f}_{filt} = \mathcal{F}^{-1}(\hat{\mathbf{s}} \square \hat{\mathbf{f}}).$$

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

Spectral Derivative

- Consider approximating the derivative of a periodic function $f(x)$, computed at a set of N equally-spaced nodes, \mathbf{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:

Spectrally-accurate finite-difference derivative

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

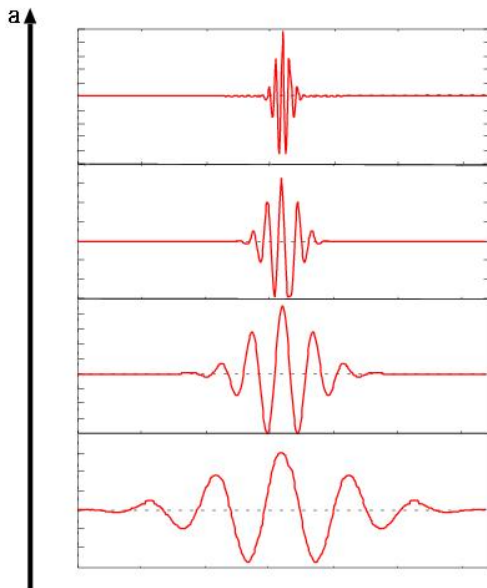
$$\phi' = \sum_{k=0}^{N-1} (ik \hat{f}_k) e^{ikx} = \mathcal{F}^{-1} \left(i \hat{\mathbf{f}} \square \mathbf{k} \right)$$

- Differentiation becomes multiplication in Fourier space.**

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

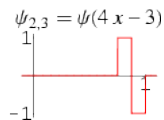
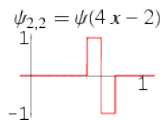
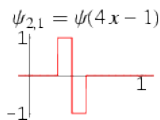
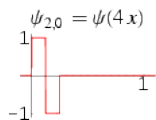
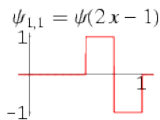
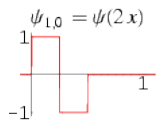
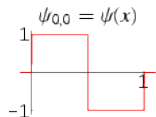
- A **mother wavelet function** $W(x)$ is a localized function in space. For simplicity assume that $W(x)$ has compact support on $[0, 1]$.
- A **wavelet basis** is a collection of **wavelets** $W_{s,\tau}(x)$ obtained from $W(x)$ by **dilation** with a **scaling factor** s and **shifting** by a **translation factor** τ :

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

- 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:

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

Haar Wavelet Basis



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} :

$$\sum_{j=0}^{J-1} 2^j = 2^J$$

Discrete Wavelet Basis

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

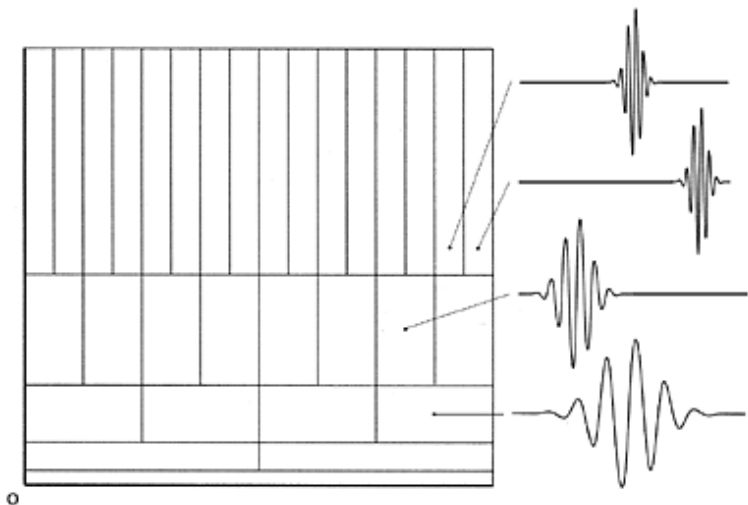
$$\mathbf{f} = c_0 + \sum_{j=0}^{J-1} \sum_{k=0}^{2^j-1} c_{jk} W_{j,k}(x_{j,k}) = \mathbf{W}_j \mathbf{c}$$

- In order to be able to quickly and stably compute the coefficients \mathbf{c} we need an **orthogonal wavelet basis**:

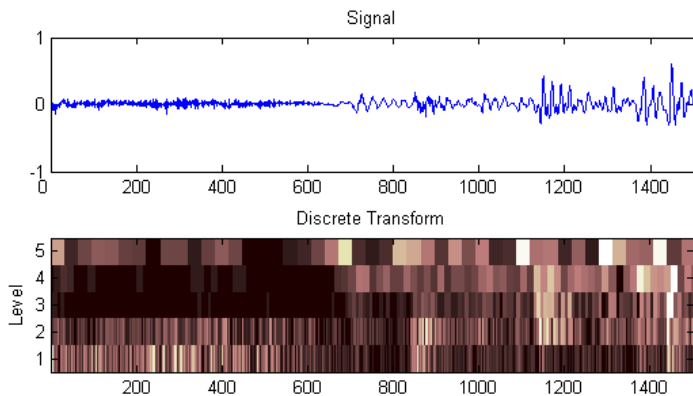
$$\int W_{j,k}(x) W_{l,m}(x) dx = \delta_{j,l} \delta_{k,m}$$

- 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.

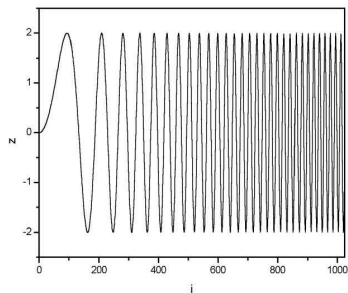
Discrete Wavelet Transform



Scaleogram



Another scaleogram



Conclusions/Summary

- **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 $O(N)$.