Scientific Computing: The Fast Fourier Transform

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

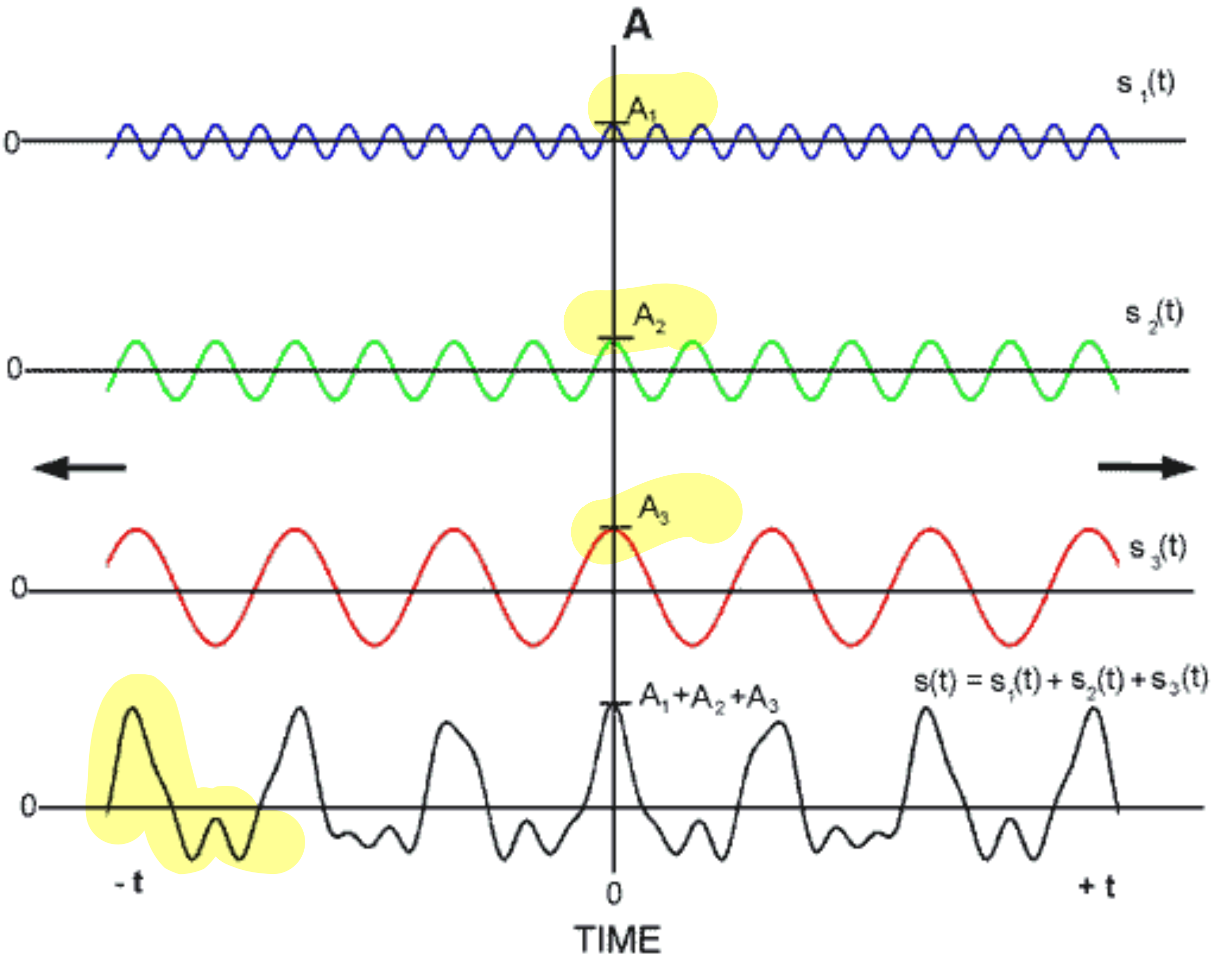
[1]Course MATH-GA.2043 or CSCI-GA.2112, Fall 2020

October 29, 2020

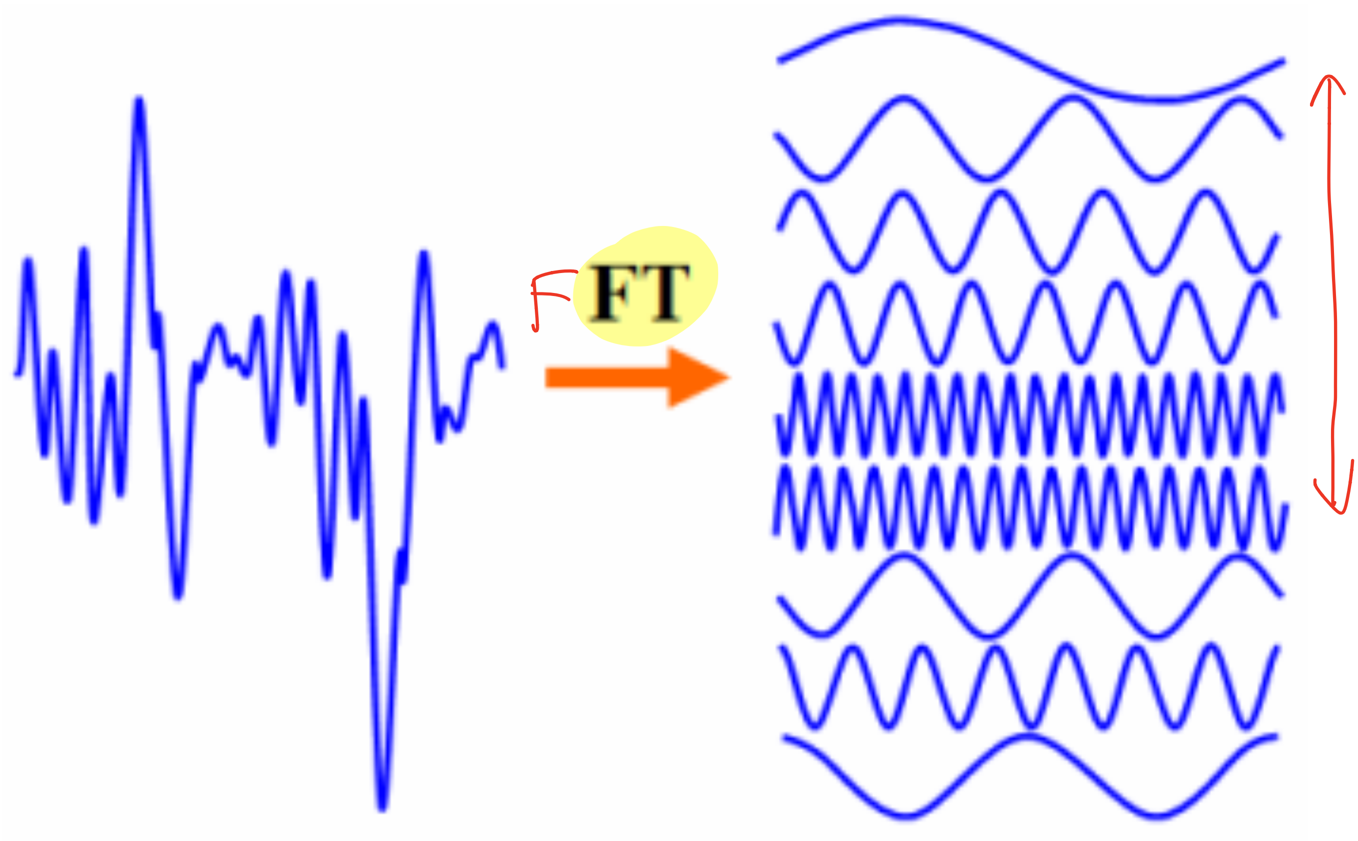
Outline

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

Fourier Basis Functions

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

- It is easy to see that these are **orhogonal** with respect to the continuous dot product

complex conjugate

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

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

L_2 sense

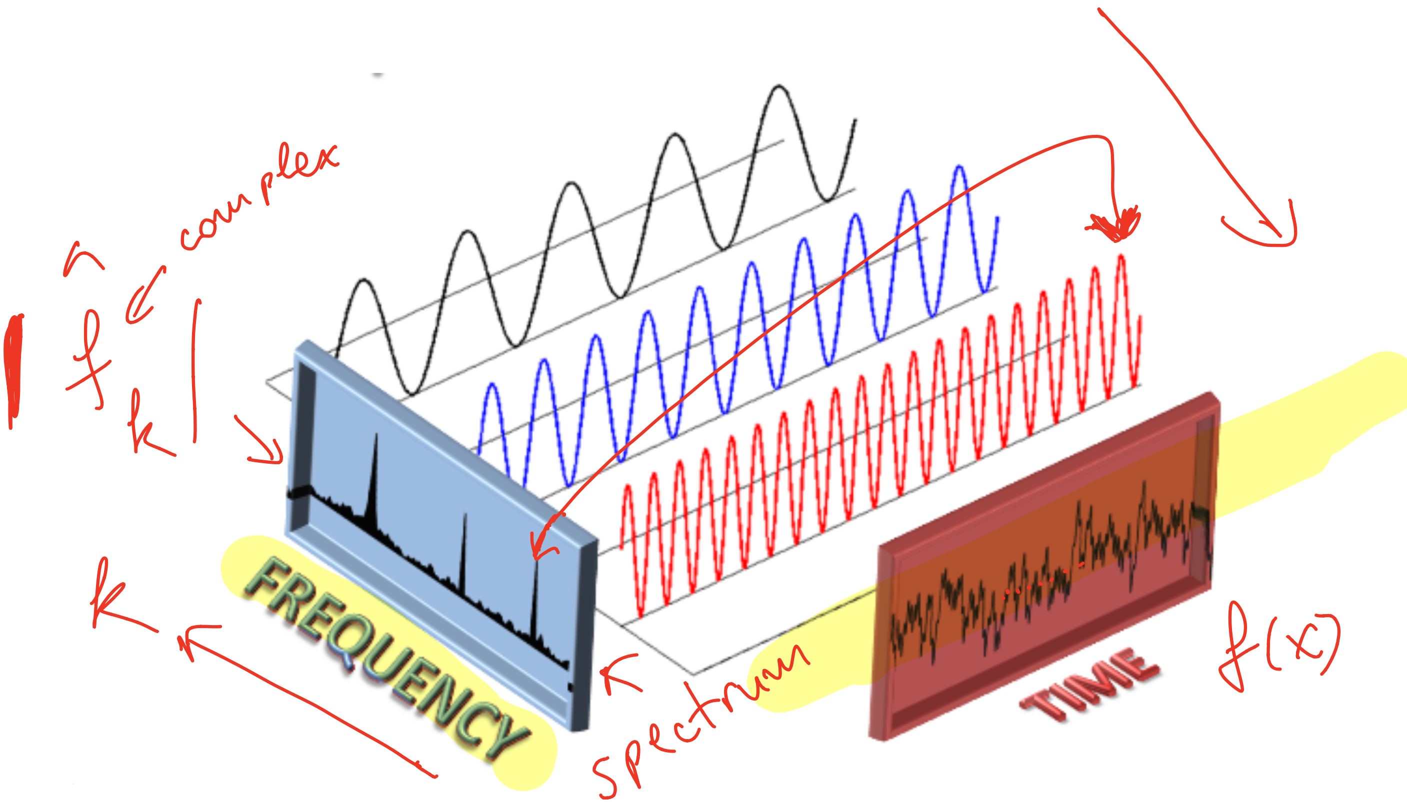
$$\forall f \in L^2_{2\pi}: \quad f(x) = \sum_{k=-\infty}^{\infty} \hat{f}_k e^{ikx}, \Rightarrow \left\| f - \sum \hat{f}_k e^{ikx} \right\|_2 = 0$$

Fourier series

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

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

Fourier Decomposition



Truncated Fourier Basis

- 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.,

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

and for simplicity we focus on N odd.

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

$$f(x) \approx \phi(x) = \sum_{k=-(N-1)/2}^{(N-1)/2} \hat{f}_k e^{ikx}.$$

Discrete Fourier Basis

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

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

  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.,

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

- Focus on N odd and denote $K = (N-1)/2$.
- **Discrete dot product** between discretized "functions":

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

Fourier Interpolant

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

- 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 $\{\phi_{-K}, \ldots, \phi_K\}$,

$$(\phi_k)_j = \exp(ikx_j),$$

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

$$\phi_k \cdot \phi_{k'} = 2\pi\delta_{k,k'}$$

$e^{ikx_j} \; e^{-ikx_j}$

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)

$$\boldsymbol{\Phi}_N = \left[\phi_{-K} \middle| \dots \phi_0 \dots \middle| \phi_K \right] \quad \Rightarrow \quad \Phi_{jk}^{(N)} = \frac{1}{\sqrt{N}} \exp\left(2\pi ijk/N\right)$$

- The truncated Fourier series is

$$\mathbf{f} = \boldsymbol{\Phi}_N \hat{\mathbf{f}}.$$

$\hat{f} \to f$

- Since the matrix $\boldsymbol{\Phi}_N$ is unitary, we know that $\boldsymbol{\Phi}_N^{-1} = \boldsymbol{\Phi}_N^{\star}$ and therefore

$$\hat{\mathbf{f}} = \boldsymbol{\Phi}_N^{\star} \mathbf{f},$$

$f \to \hat{f}$

which is nothing more than a change of basis!

Discrete Fourier Transform

- The **Fourier interpolating polynomial** is thus easy to construct

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

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

$$\hat{f}_k^{(N)} = \frac{\mathbf{f} \cdot \phi_k}{2\pi} = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) \exp(-ikx_j) \approx \hat{f}_k$$

- We can make the expressions more symmetric if we shift the frequencies to $k = 0, \ldots, N$, but one should still think of half of the frequencies as "negative" and half as "positive". See MATLAB's functions *fftshift* and *ifftshift*.

Discrete Fourier Transform

- The **Discrete Fourier Transform** (DFT) is a change of basis taking us from real/time to Fourier/frequency domain:

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

1 in matlab

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

N in matlab

- There is **different conventions** for the DFT depending on the interval on which the function is defined and placement of factors of N and 2π.
  Read the documentation to be consistent!
- A **direct** matrix-vector multiplication algorithm therefore takes $O(N^2)$ multiplications and additions. **Can we do it faster?** FFT yes!

Discrete spectrum

- The set of discrete Fourier coefficients $\hat{\mathbf{f}}$ is called the **discrete spectrum**, and in particular,

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

  is the **power spectrum** which measures the frequency content of a signal.
- If f is real, then $\hat{f}$ satisfies the **conjugacy property**

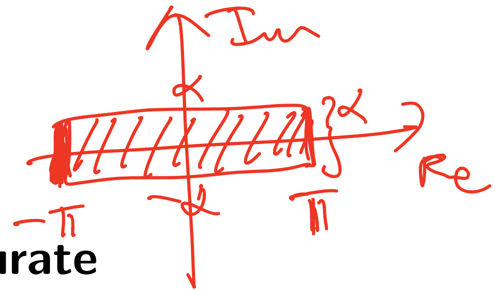
$$\hat{f}_{-k} = \hat{f}_k^\star,$$

  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.

Approximation error: Analytic

- If $f\,(t = x + iy)$ is **analytic** in a half-strip around the real axis of half-width α and bounded by $|f\,(t)| < M$, then

$$\left|\hat{f}_k\right| \leq Me^{-\alpha|k|}.$$



- Then the Fourier interpolant is **spectrally-accurate**

$$\|f - \phi\|_\infty \leq 4 \sum_{k=n+1}^{\infty} Me^{-\alpha k} = \frac{2Me^{-\alpha n}}{e^{\alpha} - 1} \text{ (geometric series sum)}$$

- The Fourier interpolating trigonometric polynomial is spectrally accurate and a really great approximation for (very) smooth functions.

Spectral Accuracy (or not)

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

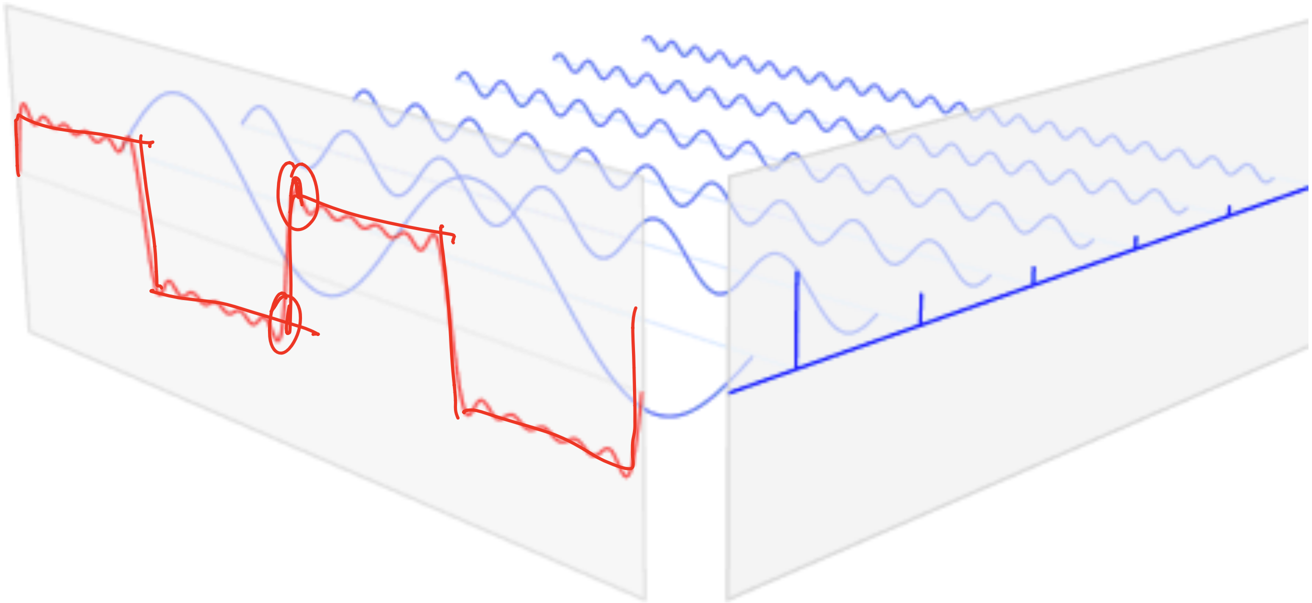
$$\|f(x) - \phi(x)\| \sim e^{-N}$$

for **sufficiently smooth functions**.

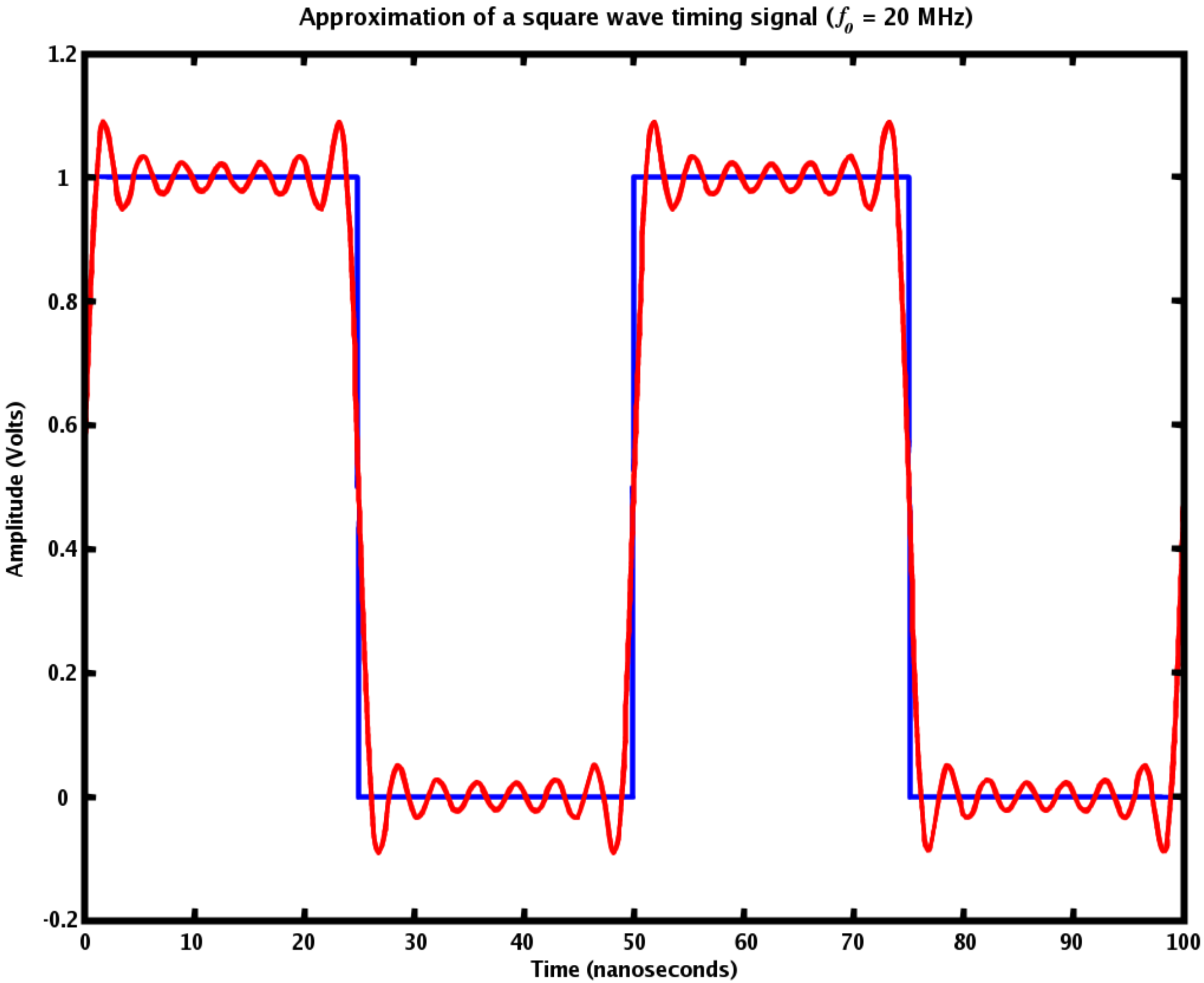
- Specifically, what is needed is sufficiently **rapid decay of the Fourier coefficients** with k, e.g., exponential decay $\left|\hat{f}_k\right| \sim e^{-|k|}$.
- Discontinuities cause slowly-decaying Fourier coefficients, e.g., power law decay $\left|\hat{f}_k\right| \sim k^{-1}$ for **jump discontinuities**.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

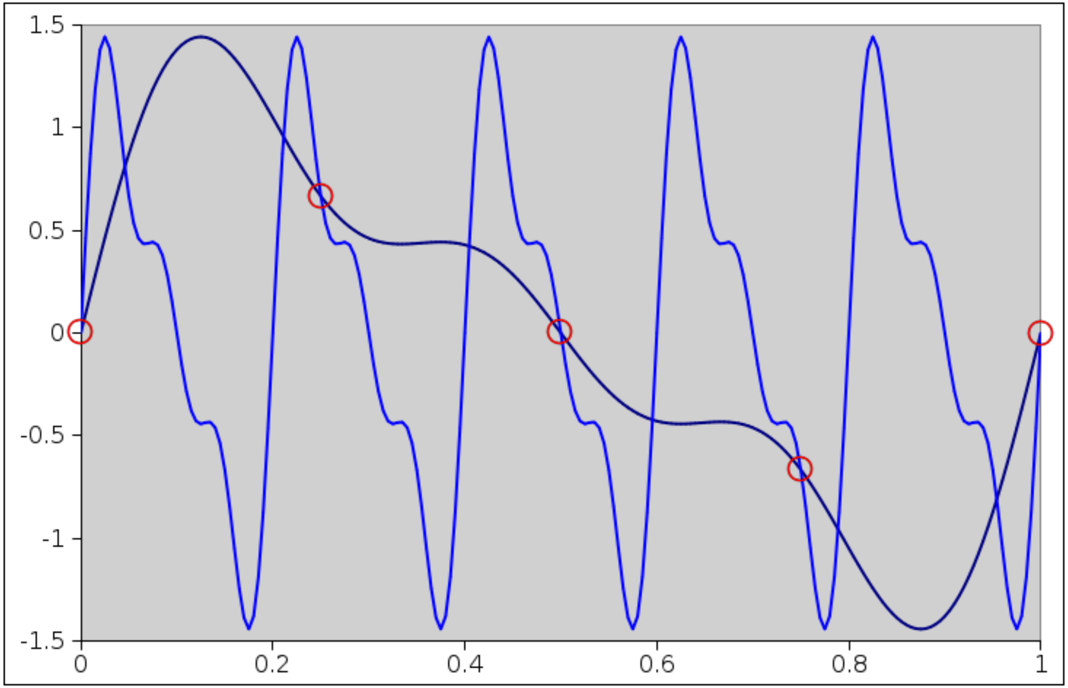


Gibbs Phenomenon



Aliasing

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



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

DFT

- Recall the transformation from real space to frequency space and back:

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

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

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

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

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

FFT

- We can write the transforms in matrix notation:

$$\hat{\mathbf{f}} = \frac{1}{\sqrt{N}} \mathbf{U}_N \mathbf{f}$$

$$\mathbf{f} = \frac{1}{\sqrt{N}} \mathbf{U}_N^\star \hat{\mathbf{f}},$$

$\Phi = U$

where the **unitary Fourier matrix** is an $N \times N$ matrix with entries

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

- A **direct** matrix-vector multiplication algorithm therefore takes $O(N^2)$ multiplications and additions.
- Is there a faster way to compute the **non-normalized**

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j \omega_N^{jk} \;?$$

$k = 0, \ldots, N-1$

Can we do it in $O(N)$ time?

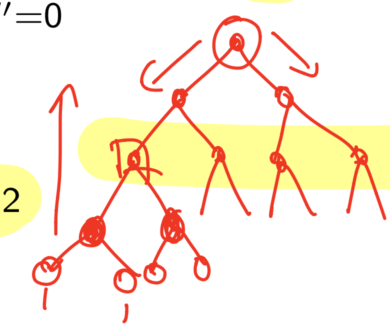
FFT

- 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:

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

- 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**:

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

$$\hat{f}_k = \mathbf{U}_N \mathbf{f} = \left(\mathbf{U}_{N/2} \mathbf{f}_{even} + \omega_N^k \mathbf{U}_{N/2} \mathbf{f}_{odd}\right)$$

FFT Complexity

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

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

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

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

  This is because such ordering (shift) makes the forward and inverse transforms symmetric.
- The function *fftshift* can be used to order the frequencies in the "normal" way, and *ifftshift* does the reverse:

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

Multidimensional FFT

- DFTs and FFTs generalize straightforwardly to higher dimensions due to separability: **Transform each dimension independently**

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

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

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

$$\hat{\mathbf{f}} = \mathcal{F}_{row}\left(\mathcal{F}_{col}\left(\mathbf{f}\right)\right)$$

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

$$N_x N_y \log N_x + N_x N_y \log N_y = N_x N_y \log\left(N_x N_y\right) = N \log N$$

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
$$\hat{\mathbf{f}} = \mathcal{F}(\mathbf{f}) \text{ and } \mathbf{f} = \mathcal{F}^{-1}\left(\hat{\mathbf{f}}\right).$$
- Plain FFT is used in signal processing for **digital filtering**: Multiply the spectrum by a filter $\hat{S}(k)$ discretized as $\hat{\mathbf{s}} = \left\{\hat{S}(k)\right\}_k$:
$$\mathbf{f}_{filt} = \mathcal{F}^{-1}\left(\hat{\mathbf{s}} \boxdot \hat{\mathbf{f}}\right).$$
- Examples include **low-pass**, **high-pass**, or **band-pass filters**. Note that **aliasing** can be a problem for digital filters.

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)

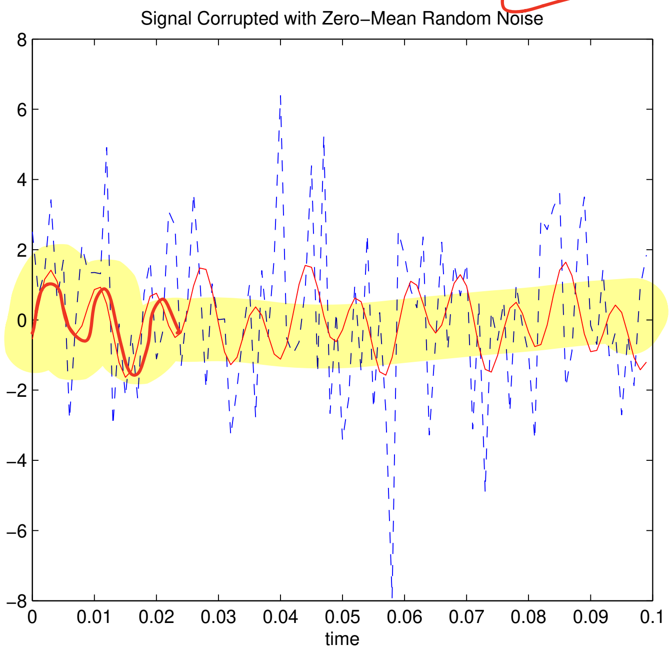
```
if (0)
   N=(L/2)*2; % Even N
   y_hat = fft(y(1:N));
   % Frequencies ordered in a funny way:
   f_funny = 2*pi/T* [0:N/2-1, -N/2:-1];
   % Normal ordering:
   f_normal = 2*pi/T* [-N/2 : N/2-1];
else
   N=(L/2)*2-1; % Odd N
   y_hat = fft(y(1:N));
   % Frequencies ordered in a funny way:
   f_funny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

or fftshift(fft(y(1:N)))

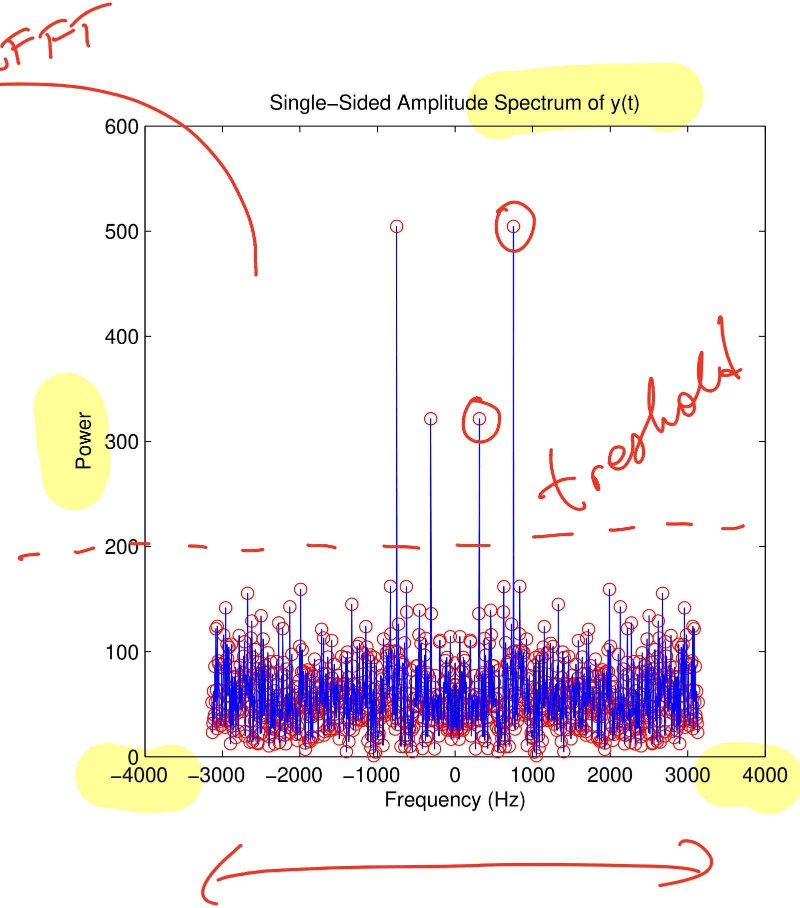
FFT-based noise filtering (3)

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

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



Signal Corrupted with Zero−Mean Random Noise
8
6
4
2
0
−2
−4
−6
−8
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
time



iFFT
Single−Sided Amplitude Spectrum of y(t)
600
500
400
300
200
100
0
Power
treshold
−4000
−3000
−2000
−1000
0
1000
2000
3000
4000
Frequency (Hz)

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

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

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

- **Differentiation becomes multiplication in Fourier space.**

Unmatched mode

- Recall that for even N there is one unmatched mode, the one with the highest frequency and amplitude $\hat{f}_{N/2}$.
- We need to choose what we want to do with that mode; see notes by S. G. Johnson (MIT) linked on webpage for details:

$$\phi(x) = \hat{f}_0 + \sum_{0<k<N/2} \left(\hat{f}_k e^{ikx} + \hat{f}_{N-k} e^{-ikx} \right) + \hat{f}_{N/2} \cos\left(\frac{Nx}{2}\right).$$

This is the unique "**minimal oscillation**" trigonometric interpolant.

- Differentiating this we get

$$\widehat{(\phi')}_k = \hat{f}_k \begin{cases} 0 & \text{if } k = N/2 \\ ik & \text{if } k < N/2 \\ i\,(k-N) & \text{if } k > N/2 \end{cases}.$$

- Real valued interpolation samples result in **real-valued** $\phi(x)$ for all x.

FFT-based differentiation

```
% From Nick Trefethen's Spectral Methods book
% Differentiation of exp(sin(x)) on (0,2*pi]:
  N = 8; % Even number!
  h = 2*pi/N; x = h*(1:N)';
  v = exp(sin(x)); vprime = cos(x).*v;
  v_hat = fft(v);
  ik = 1i*[0:N/2-1 0 -N/2+1:-1]'; % Zero special mode
  w_hat = ik .* v_hat;
  w = real(ifft(w_hat));
  error = norm(w-vprime,inf)
```

(no fftshift)

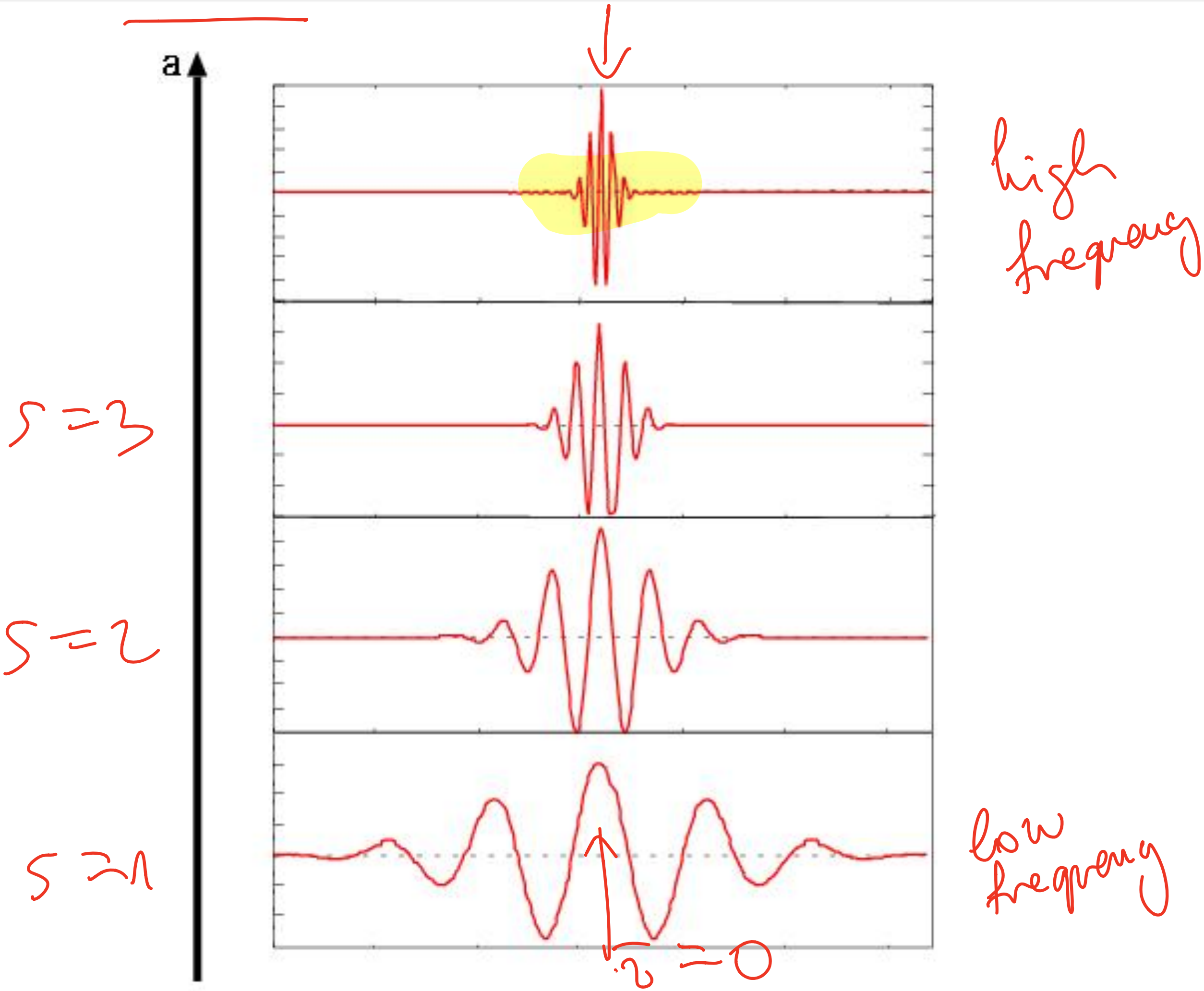
integral = ifft $\left(\frac{1}{ik} .* \text{fft}(f) \right)$

↑ except $k=0$

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 speach.
- Fourier transforms are not good with handling **discontinuities** in functions because of the Gibbs phenomenon.
- Fourier polynomails **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:

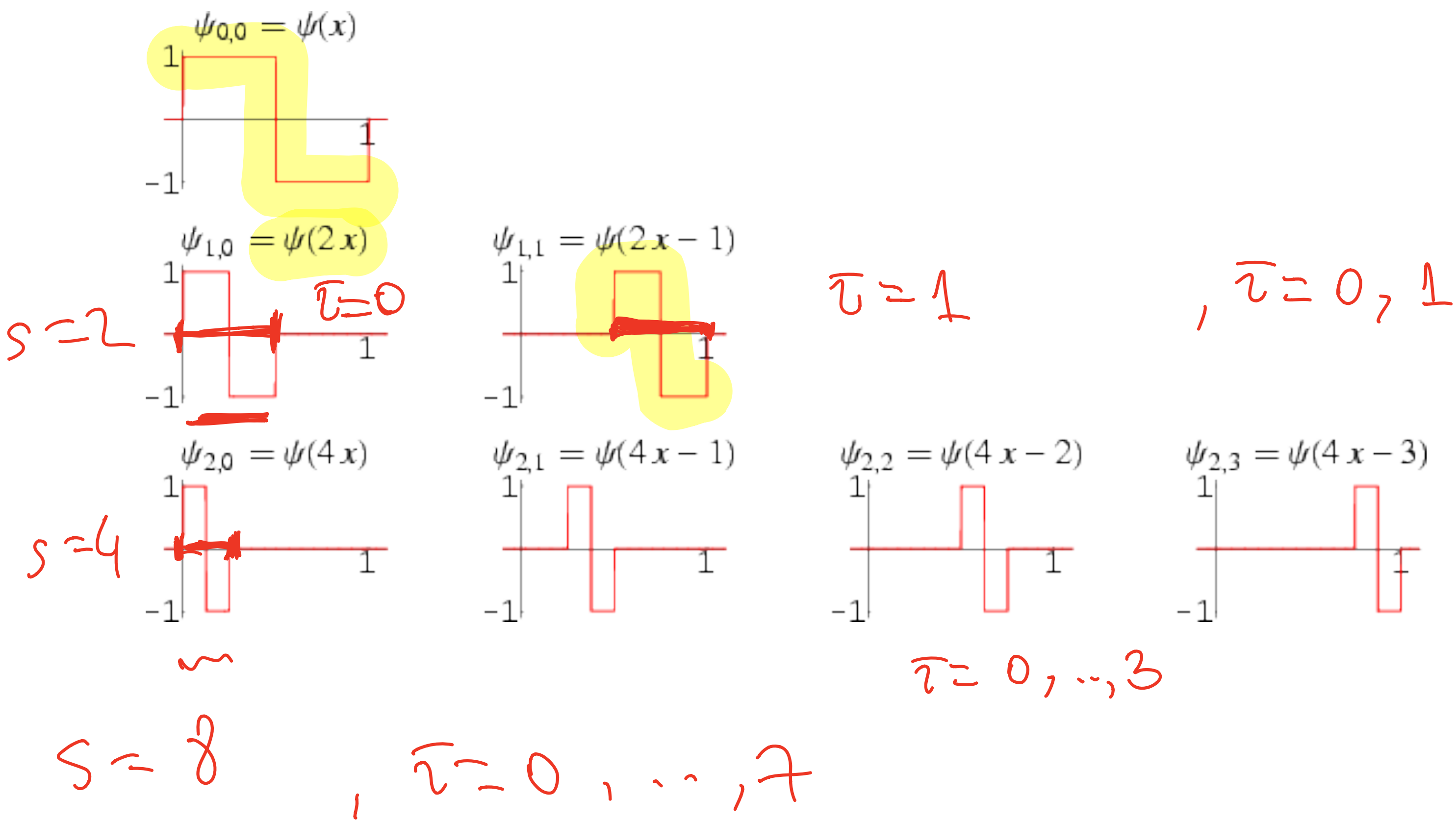
$$W_{s,\tau}(x) = W\left(sx - \tau\right).$$

frequency, location; e.g. Debauchies wavelet

- 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 **f**:

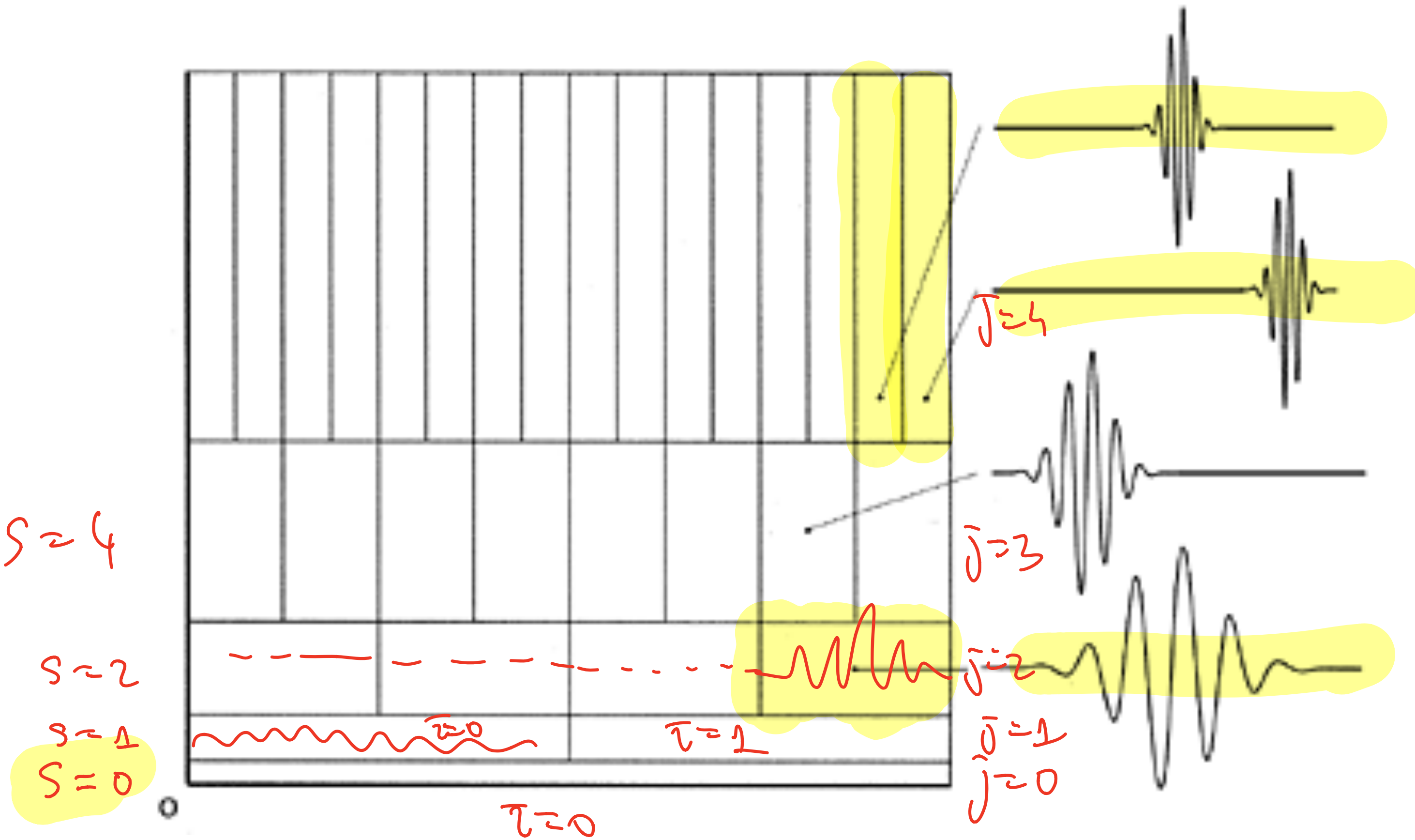
$$\mathbf{f} = c_0 + \sum_{j=0}^{J-1} \sum_{k=0}^{2^j-1} c_{jk} W_{j,k}(x_{j,k}) = \mathbf{W}_j \mathbf{c}$$

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

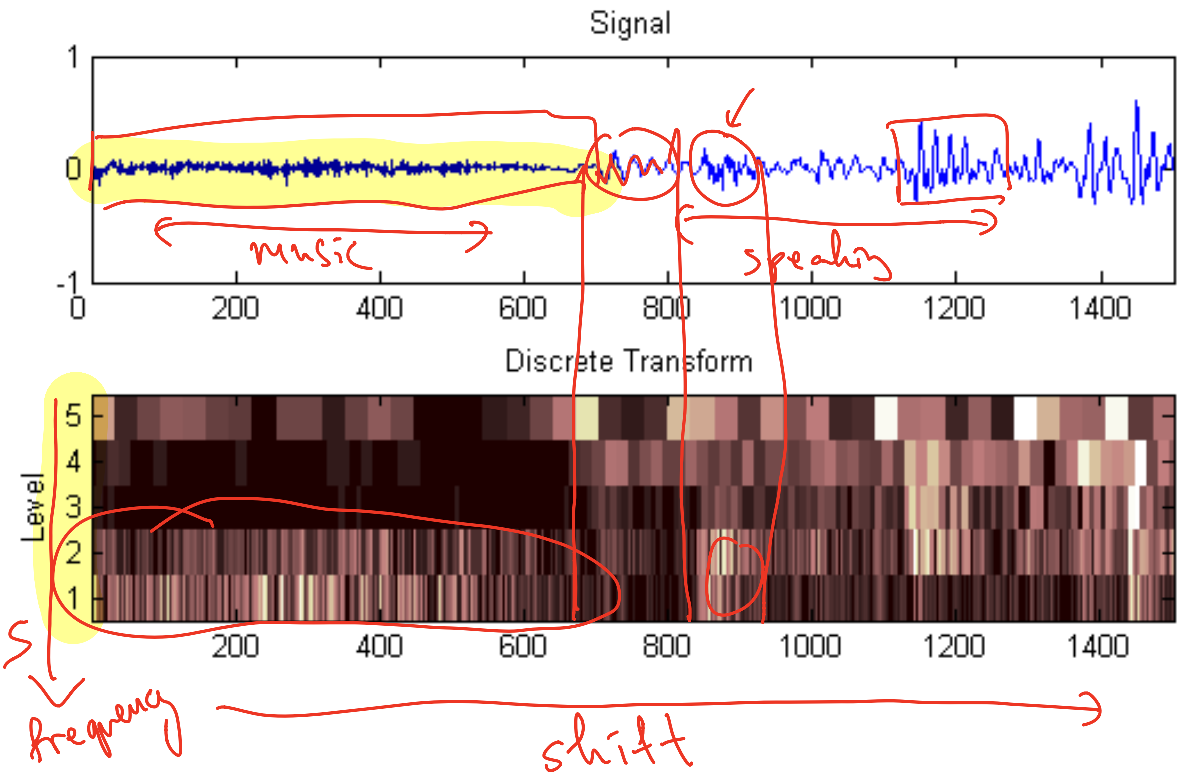
$$\int W_{j,k}(x) W_{l,m}(x) dx = \delta_{j,l} \delta_{l,m}$$

- 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.

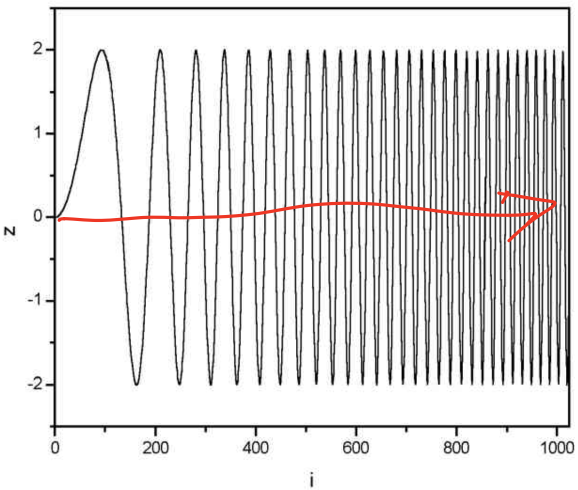
Discrete Wavelet Transform

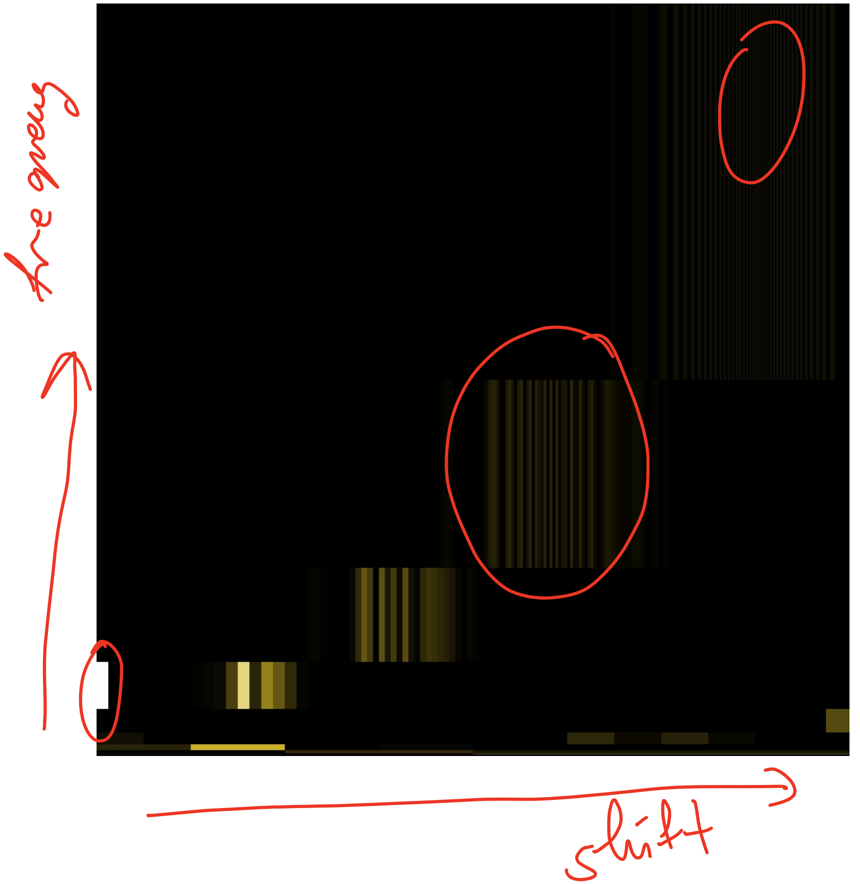


Scaleogram


Signal
1
0
-1
0
200
400
600
800
1000
1200
1400
music
speaking
Discrete Transform
Level
5
4
3
2
1
200
400
600
800
1000
1200
1400
frequency
shift

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 $O(N)$.