

*Computational Methods*  
CMSC/AMSC/MAPL 460

Fourier transform

Ramani Duraiswami,  
Dept. of Computer Science

Several slides from Prof. Healy's  
course at UMD

# Fourier Methods

- Fourier analysis (“harmonic analysis”) a key field of math
- Has many applications and has enabled many technologies.
- Basic idea: Use Fourier representation to represent functions
- Has fast algorithms to manipulate them (the fast Fourier Transform)
- Requires a complete course (Signal Processing in EE or Math 464 “Introduction to Fourier Analysis”)

# Basic idea

- Function spaces can have many different types of bases
- We have already met monomials and other polynomial basis functions
- Fourier introduced another set of basis functions : the Fourier series
- These basis functions are particularly good for describing things that repeat with time

# Fourier's Representation

$$F(t) = A_0/2 + A_1 \cos(t) + A_2 \cos(2t) + A_3 \cos(3t) + \dots \\ + B_1 \sin(t) + B_2 \sin(2t) + B_3 \sin(3t) + \dots$$

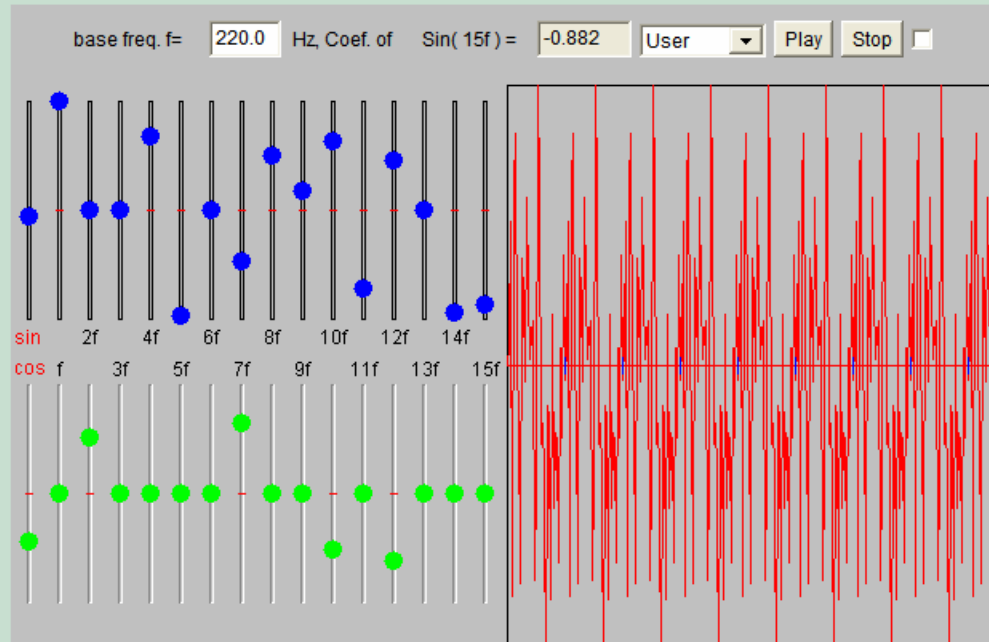
For coefficients that go to 0 fast enough these sums will converge at each value of  $t$ .

This defines a new function, which must be a *periodic function*.  
(Period  $2\pi$ )

Fourier's claim: *ANY* periodic function  $f(t)$  can be written this way

# Music

<http://www.phy.ntnu.edu.tw/ntnujava/viewtopic.php?t=33>



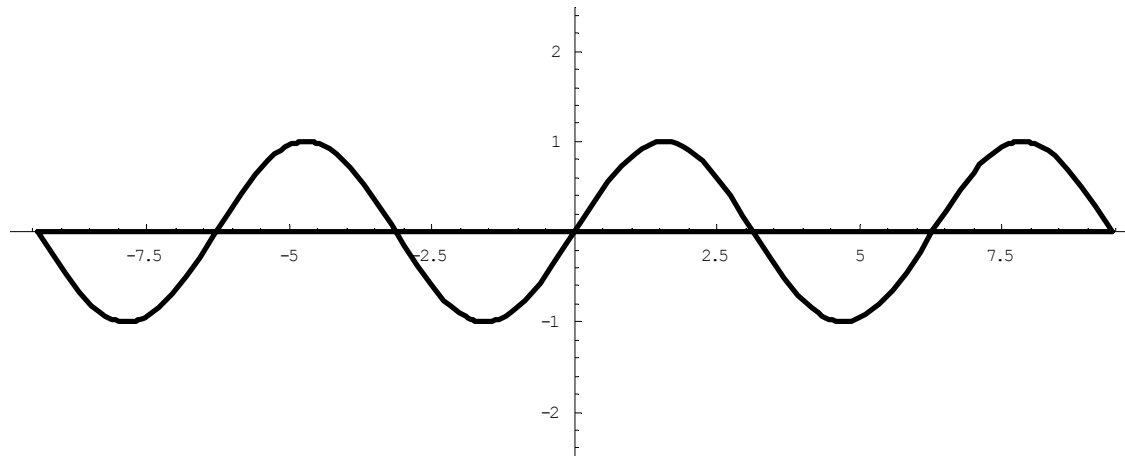
Click **get** button to fetch coefficients. Click **set** button to modify coefficients.

<input type="button" value="set"/>	0f	f	2f	3f	4f	5f	6f	7f
sin	<input type="button" value="get"/>	<input type="text" value="1.0"/>	<input type="text" value="0.0"/>	<input type="text" value="0.0"/>	<input type="text" value="0.69"/>	<input type="text" value="-0.98"/>	<input type="text" value="0.0"/>	<input type="text" value="-0.44"/>
cos	<input type="text" value="-0.44"/>	<input type="text" value="0.0"/>	<input type="text" value="0.52"/>	<input type="text" value="0.0"/>	<input type="text" value="0.0"/>	<input type="text" value="0.0"/>	<input type="text" value="0.0"/>	<input type="text" value="0.66"/>
	8f	9f	10f	11f	12f	13f	14f	15f
sin	<input type="text" value="0.51"/>	<input type="text" value="0.19"/>	<input type="text" value="0.64"/>	<input type="text" value="-0.73"/>	<input type="text" value="0.47"/>	<input type="text" value="0.0"/>	<input type="text" value="-0.93"/>	<input type="text" value="-0.88"/>

# Mr. Fourier's Representation

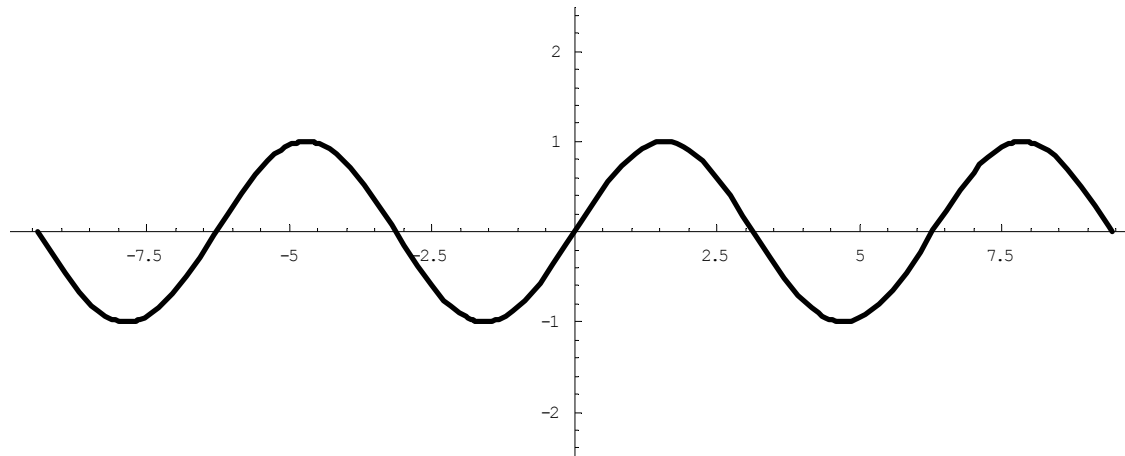
Represent  $f(t) = t$  for  $|t| < \pi$  and  $2\pi$  periodic

$\sin(t) + \dots$



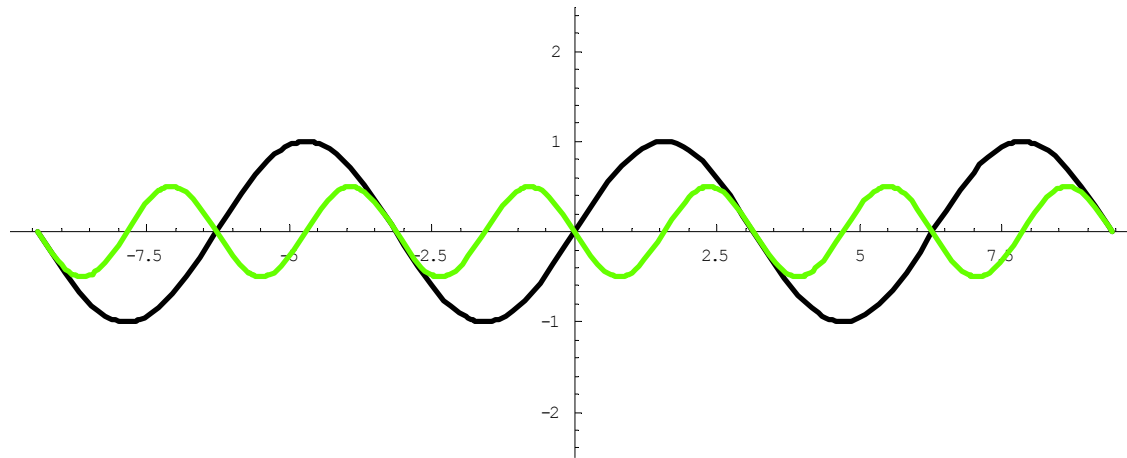
# Mr. Fourier's Representation

$1 \sin(t) + \dots$



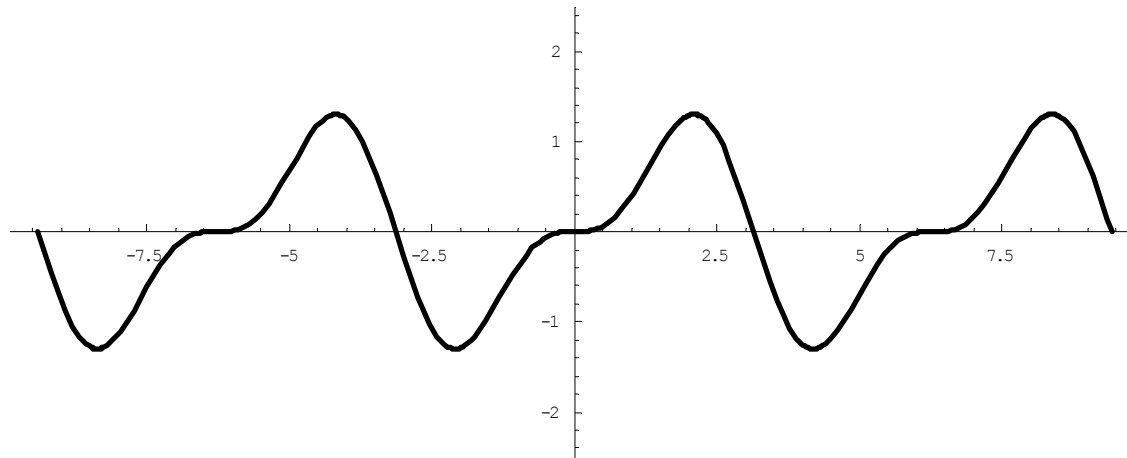
# Mr. Fourier's Representation

$$\sin(t) - \frac{1}{2} \sin(2t) + \dots$$



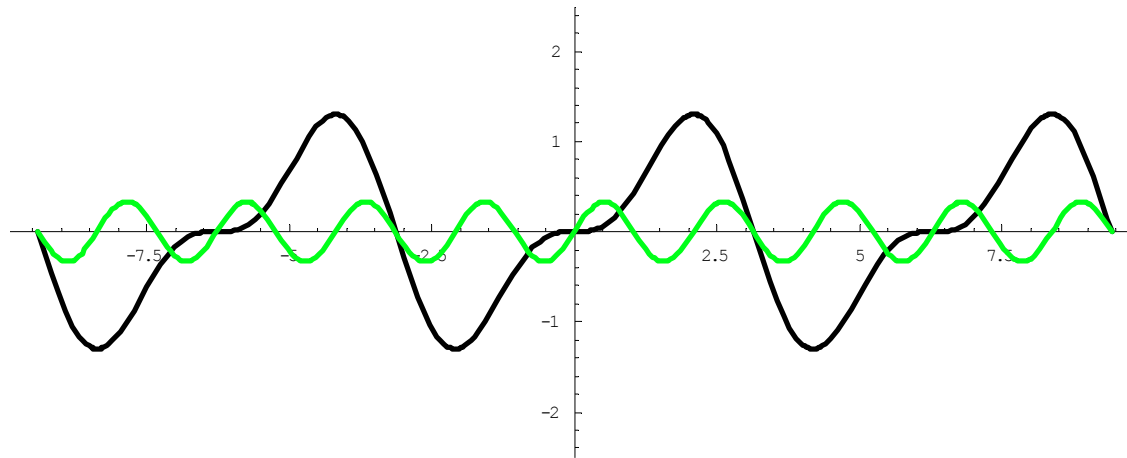
# Mr. Fourier's Representation

$$1 \sin(t) - \frac{1}{2} \sin(2t) + \dots$$



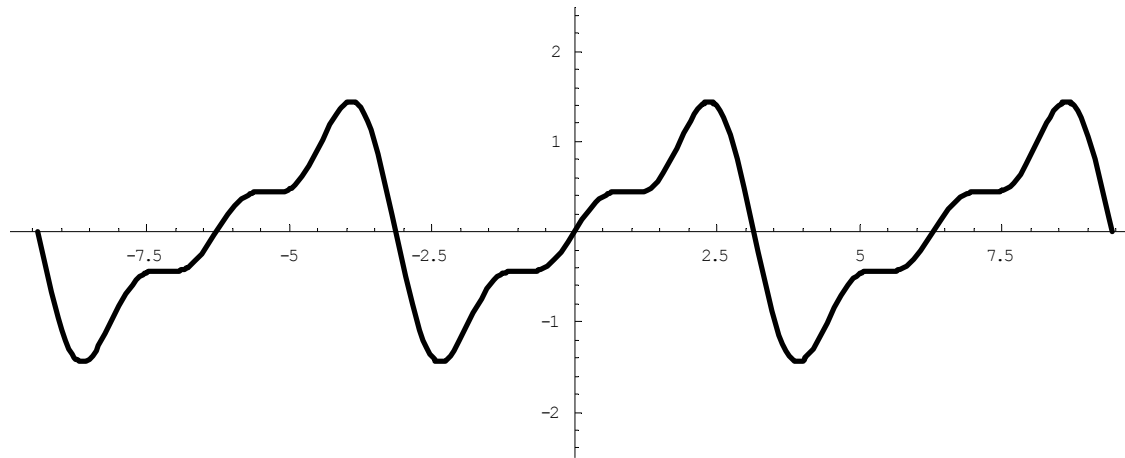
# Mr. Fourier's Representation

$$1 \sin(t) - \frac{1}{2} \sin(2t) + \frac{1}{3} \sin(3t) + \dots$$



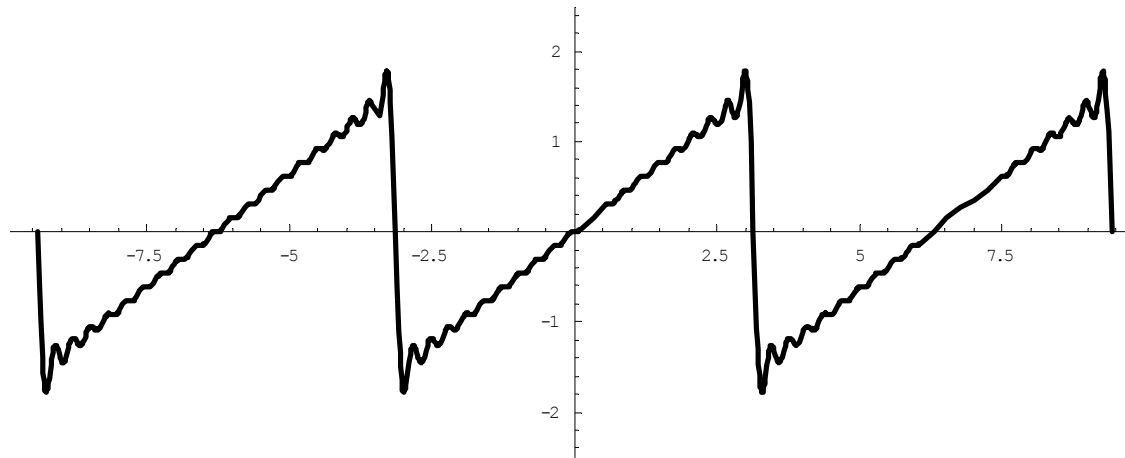
# Mr. Fourier's Representation

$$1 \sin(t) - \frac{1}{2} \sin(2t) + \frac{1}{3} \sin(3t) + \dots$$



# Mr. Fourier's Representation

20'th degree Fourier expansion



How do you get the Coefficients for a given  $f$  ?

$$A_0/2 + A_1 \cos(t) + A_2 \cos(2t) + A_3 \cos(3t) + \dots \\ + B_1 \sin(t) + B_2 \sin(2t) + B_3 \sin(3t) + \dots$$

Fourier's claim:

- *ANY* periodic function  $f(t)$  can be written this way (SYNTHESIS)
- The coefficients are uniquely determined by  $f$ : (ANALYSIS)

$$A_k = \frac{1}{2\pi} \int_0^{2\pi} f(t) \cos(kt) dt$$

$$B_k = \frac{1}{2\pi} \int_0^{2\pi} f(t) \sin(kt) dt$$

*Computational Methods*  
CMSC/AMSC/MAPL 460

Fourier transform

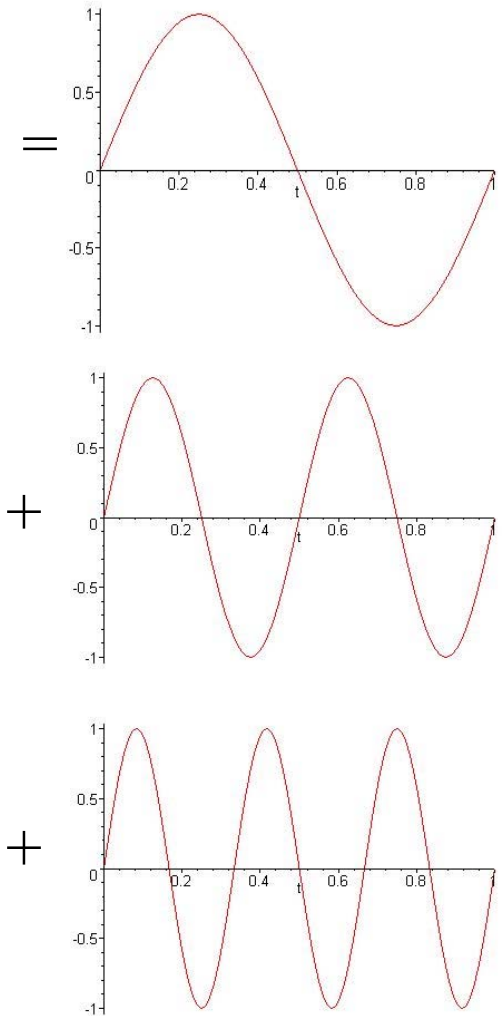
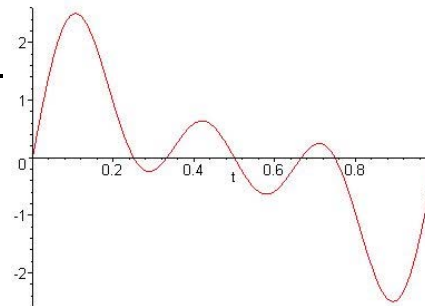
Ramani Duraiswami,  
Dept. of Computer Science

# Fourier Methods

- Last class
  - Introduced the Fourier basis
  - Showed why it might be useful
- This class
  - Introduced the notion of a Fourier Matrix
  - Introduced the Danielson Lanczos Lemma
  - Introduced the FFT algorithm
  - Detailed consideration of the FFT algorithm
  - Inverse FFT
  - Application to polynomial multiplication

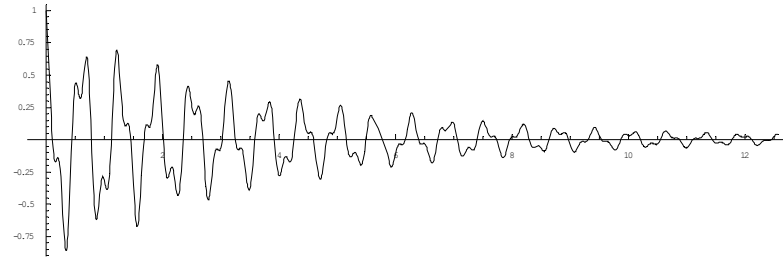
# Fourier Analysis

- Def.: mathematical techniques for breaking up a signal into its components (sinusoids)
- Jean Baptiste Joseph Fourier (1768-1830)
- can represent any continuous periodic signal as a sum of sinusoidal waves

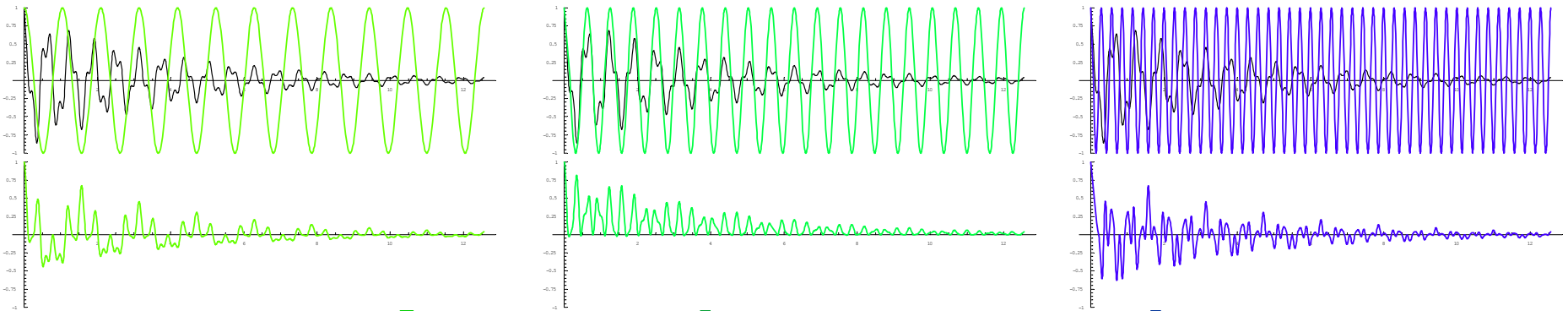


# Fourier Analysis: match data with sinusoids

$s(t)$

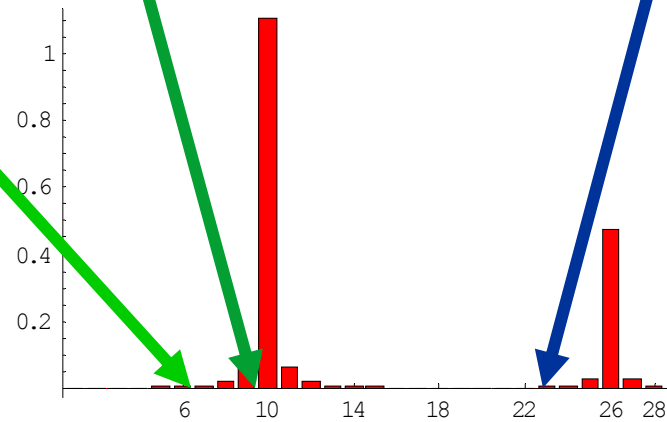


Time t



$S[k]$

$$= \int s(t) \cos(k t) dt$$



Frequency k

# Complex Notation

For  $f$ , periodic with period  $p$

Fourier transform  $f(t) \rightarrow F[k]$

$$\begin{aligned} F[k] &= \frac{1}{p} \int_0^p f(t) e^{-2\pi i k t/p} dt \\ &= \frac{1}{p} \int_0^p f(t) \cos(2\pi k t/p) dt \\ &\quad - \frac{i}{p} \int_0^p f(t) \sin(2\pi k t/p) dt \end{aligned}$$

Inverse Fourier transform  $F[k] \rightarrow f(t)$

$$f(t) = \sum_{k \in \mathbb{Z}} F[k] e^{2\pi i k t/p}$$

# Sampling

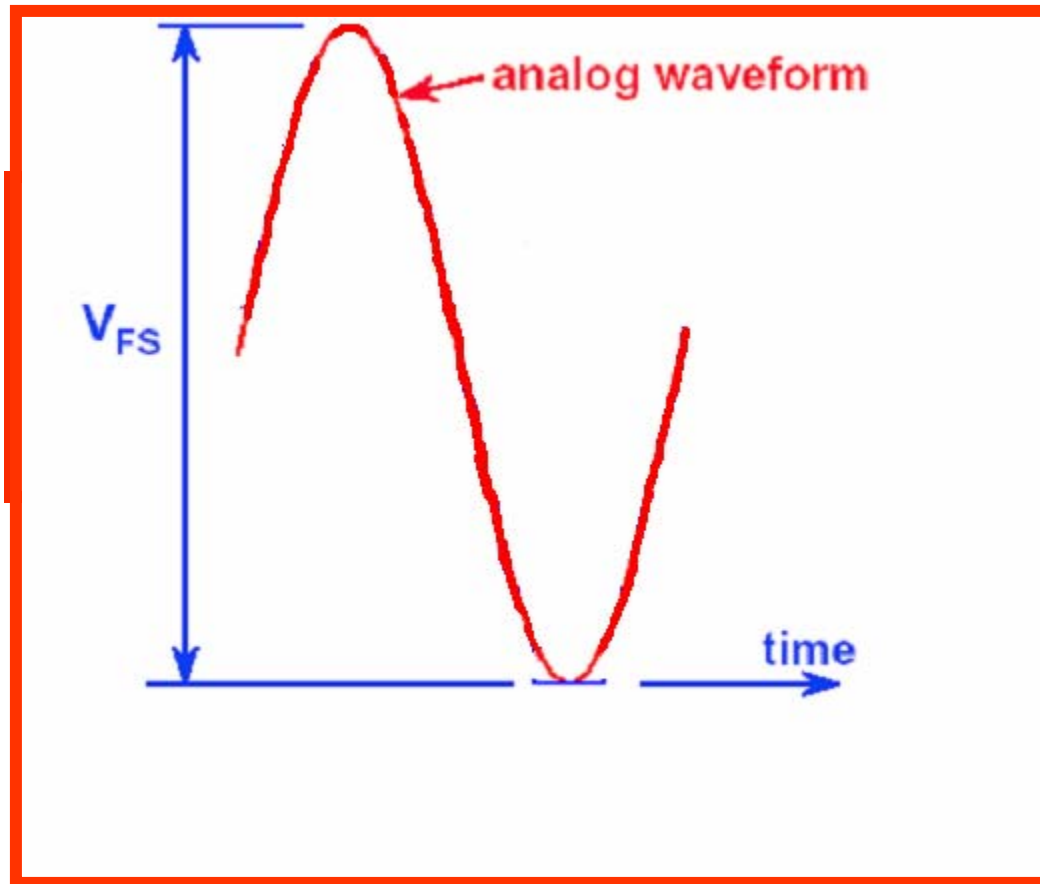
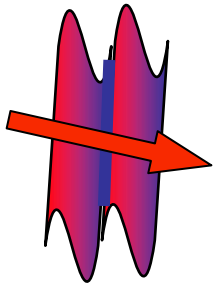
Fourier representations work just fine with sampled data

Simple connection to Fourier of the continuous function it came from

Familiar example: Digital Audio

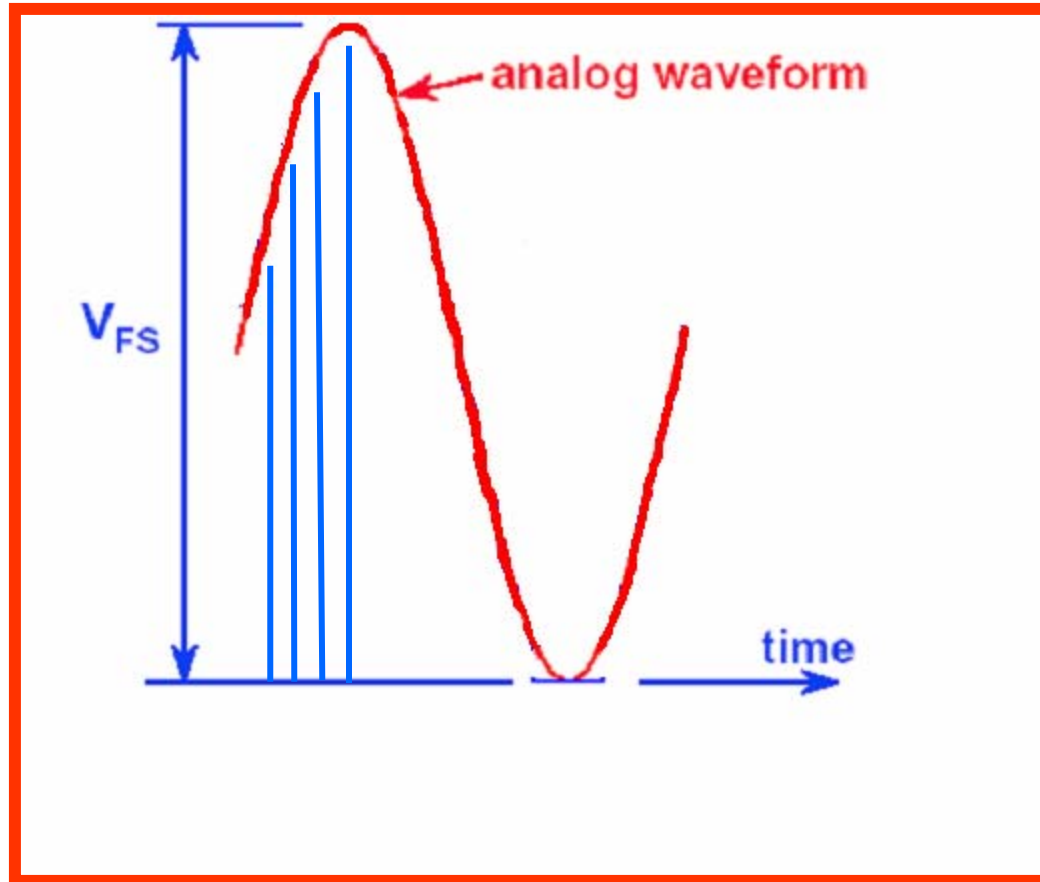
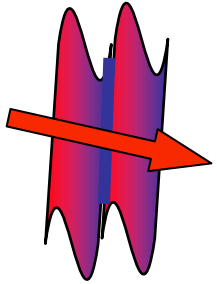
# Measuring and Discretizing Input field

Physical Field  
(continuum)



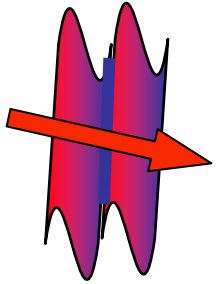
# Sample

Physical Field  
(continuum)

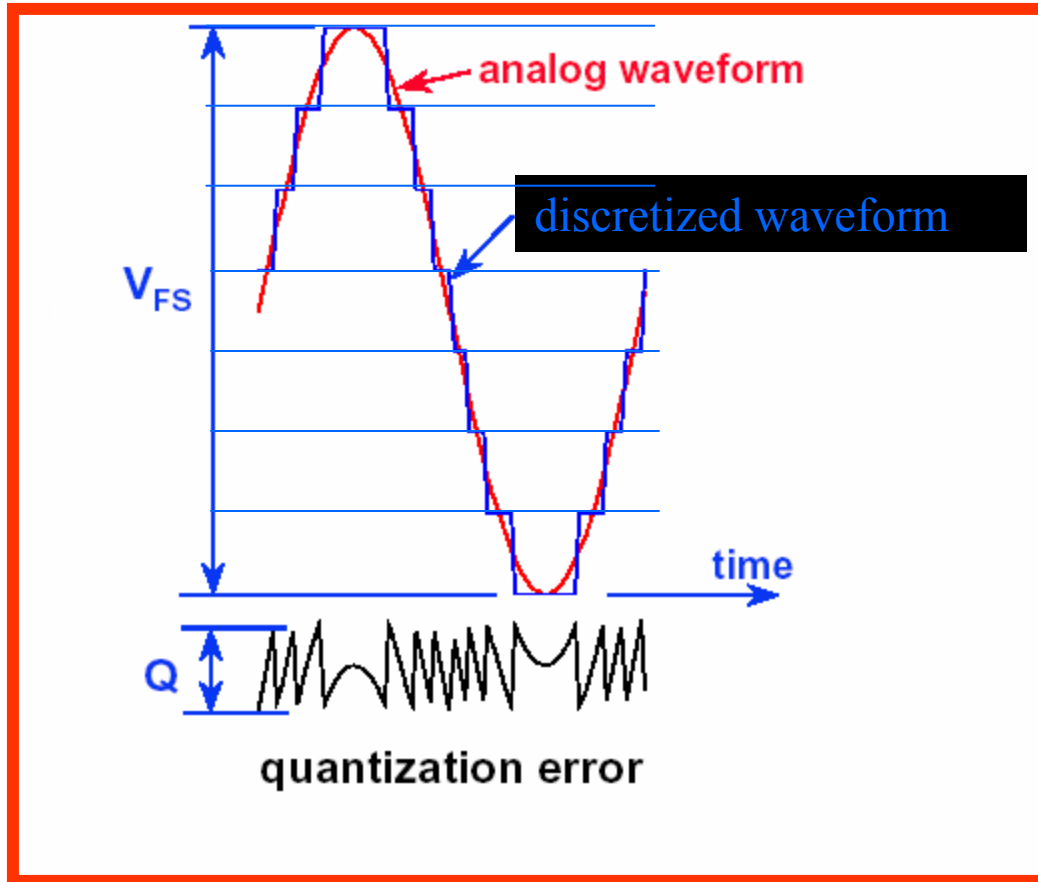


# Quantize

Physical Field  
(continuum)

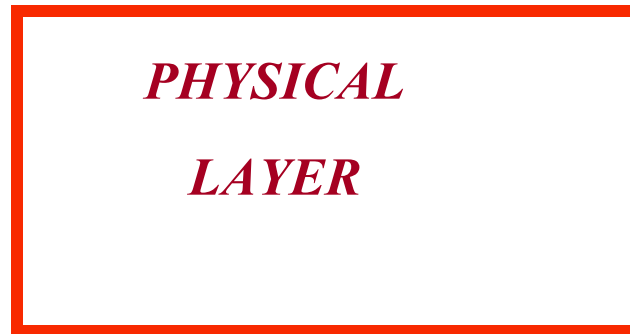
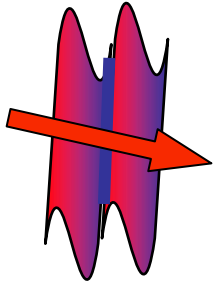


$$NP_0(rms) = \frac{Q}{\sqrt{12}}$$

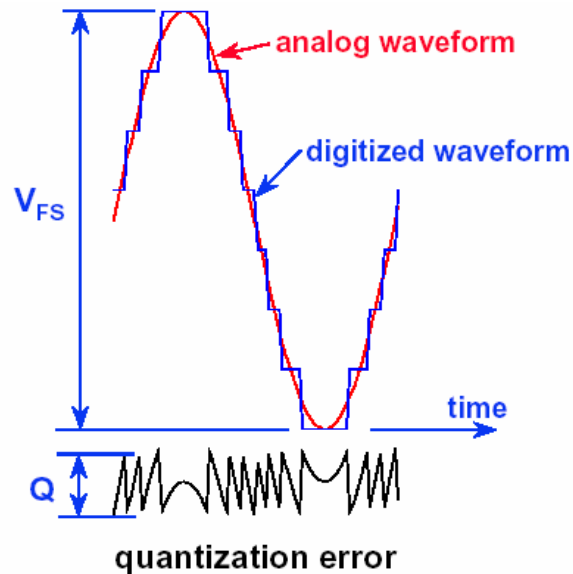
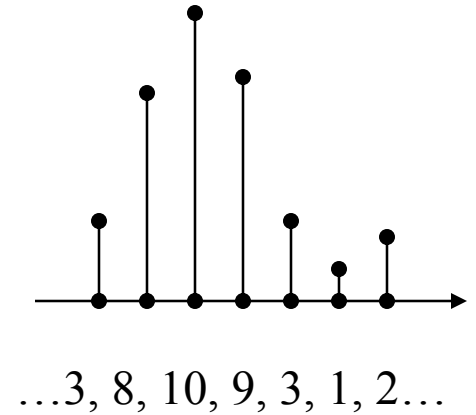


# Code and output

Physical Field  
(continuum)

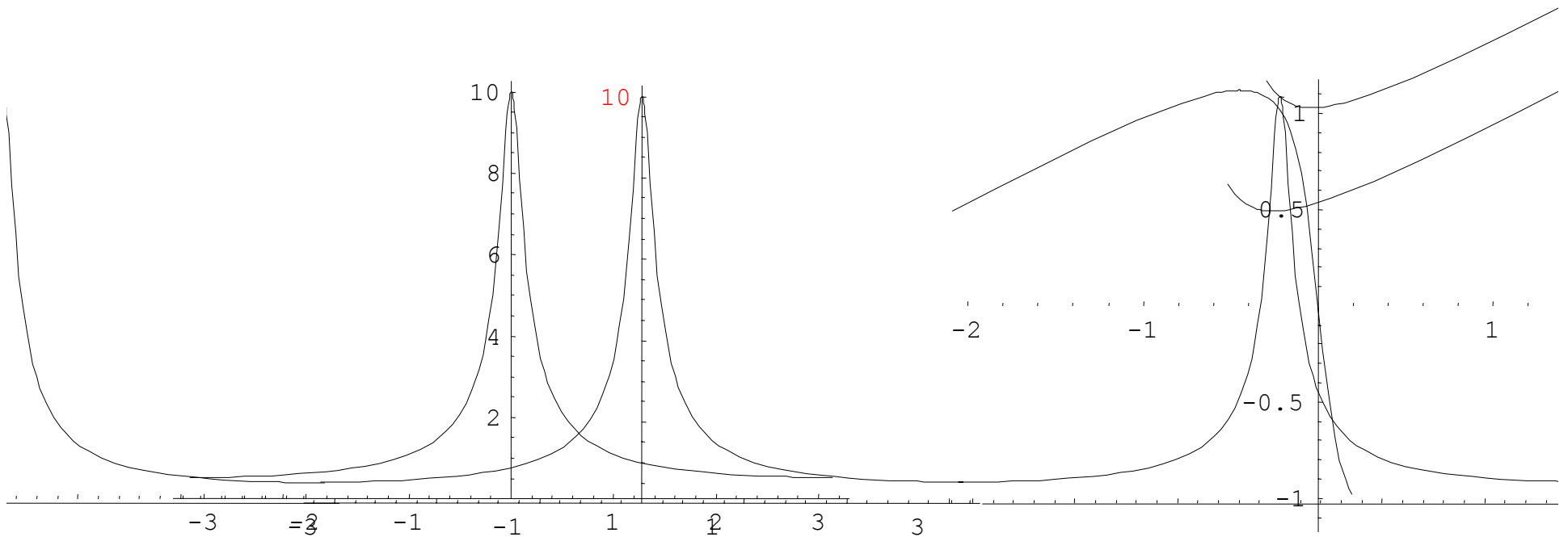


Digital Representation



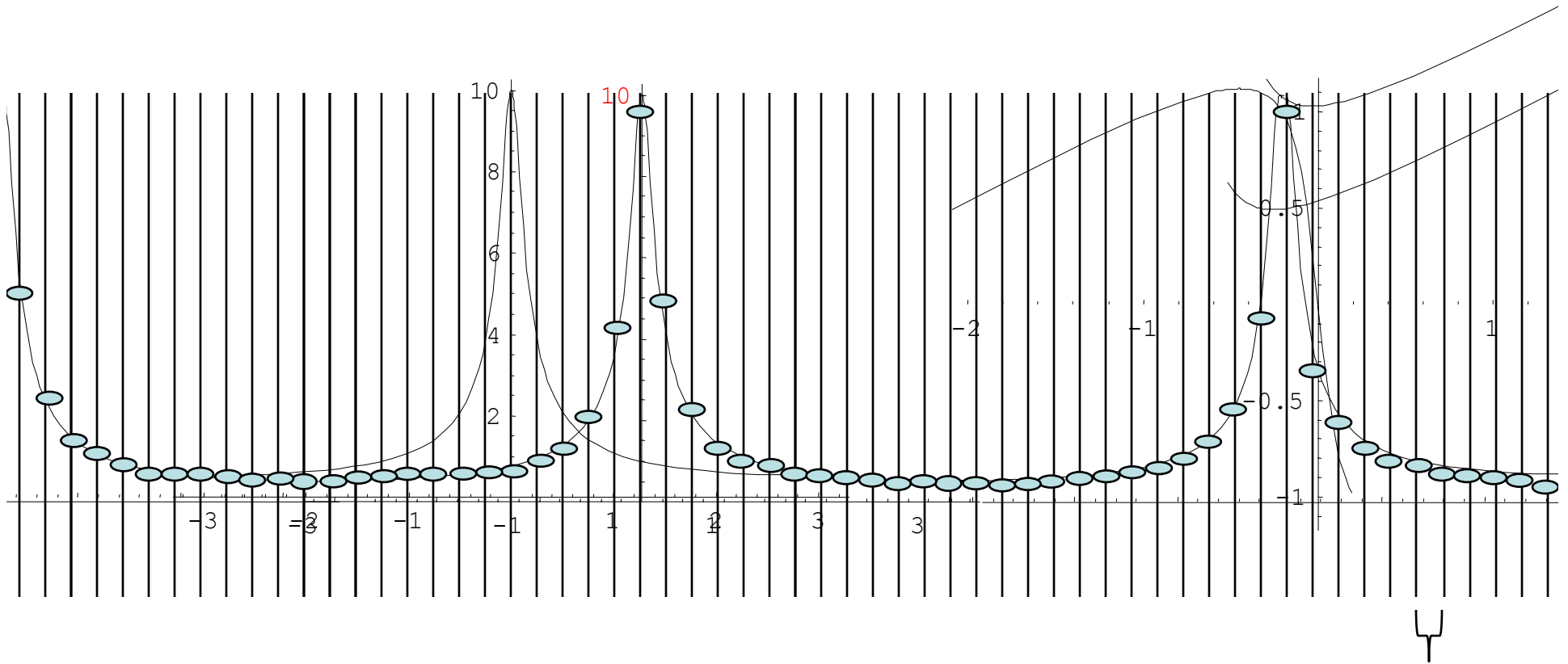
# Sampling

Often must work with a discrete set of measurements of a continuous function



# Sampling

Takes a function defined on  $\mathcal{R}$  and creates a function defined on  $\mathcal{Z}$



$S_h$

$f(t)$



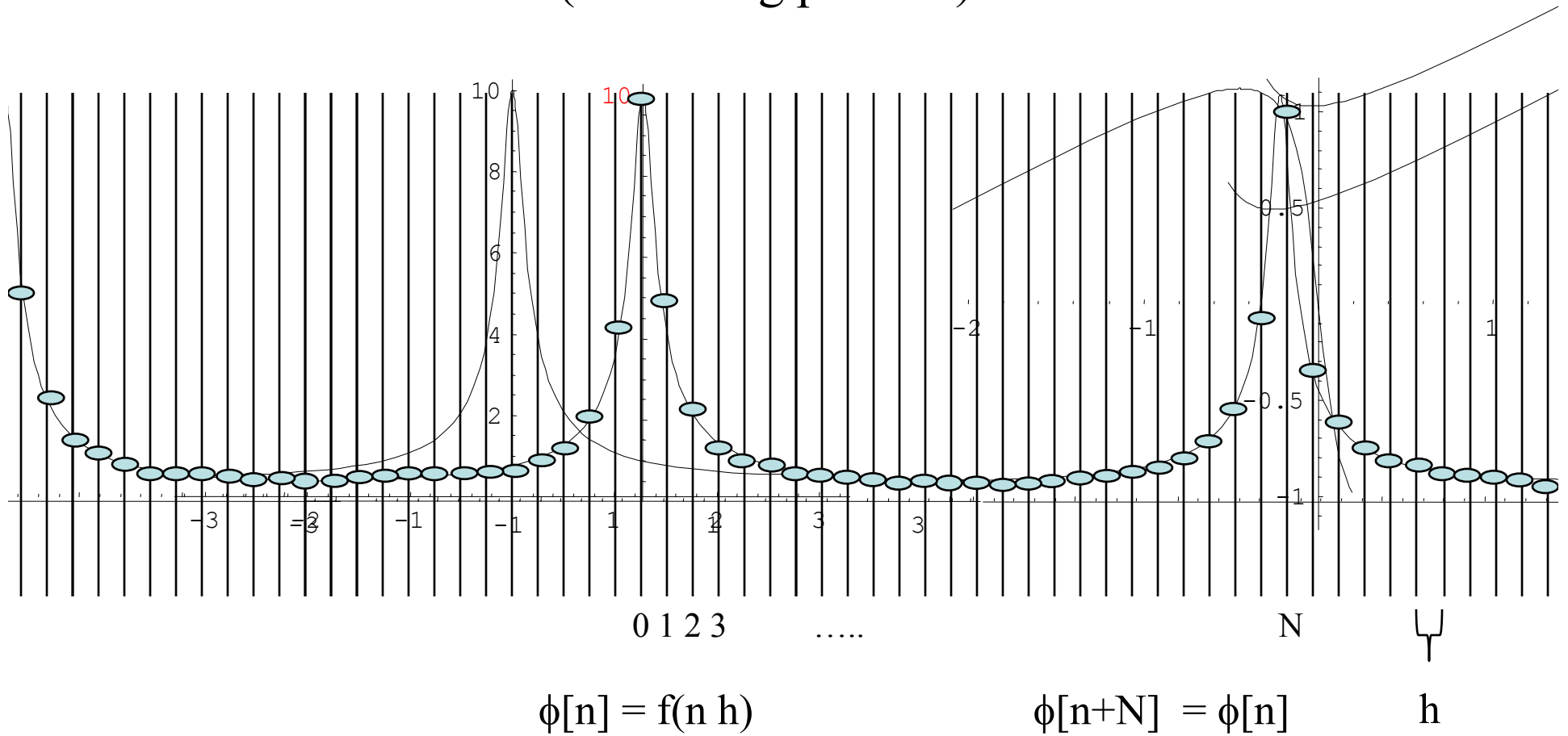
$\phi[n] = f(n h)$

$h$

# Sampling

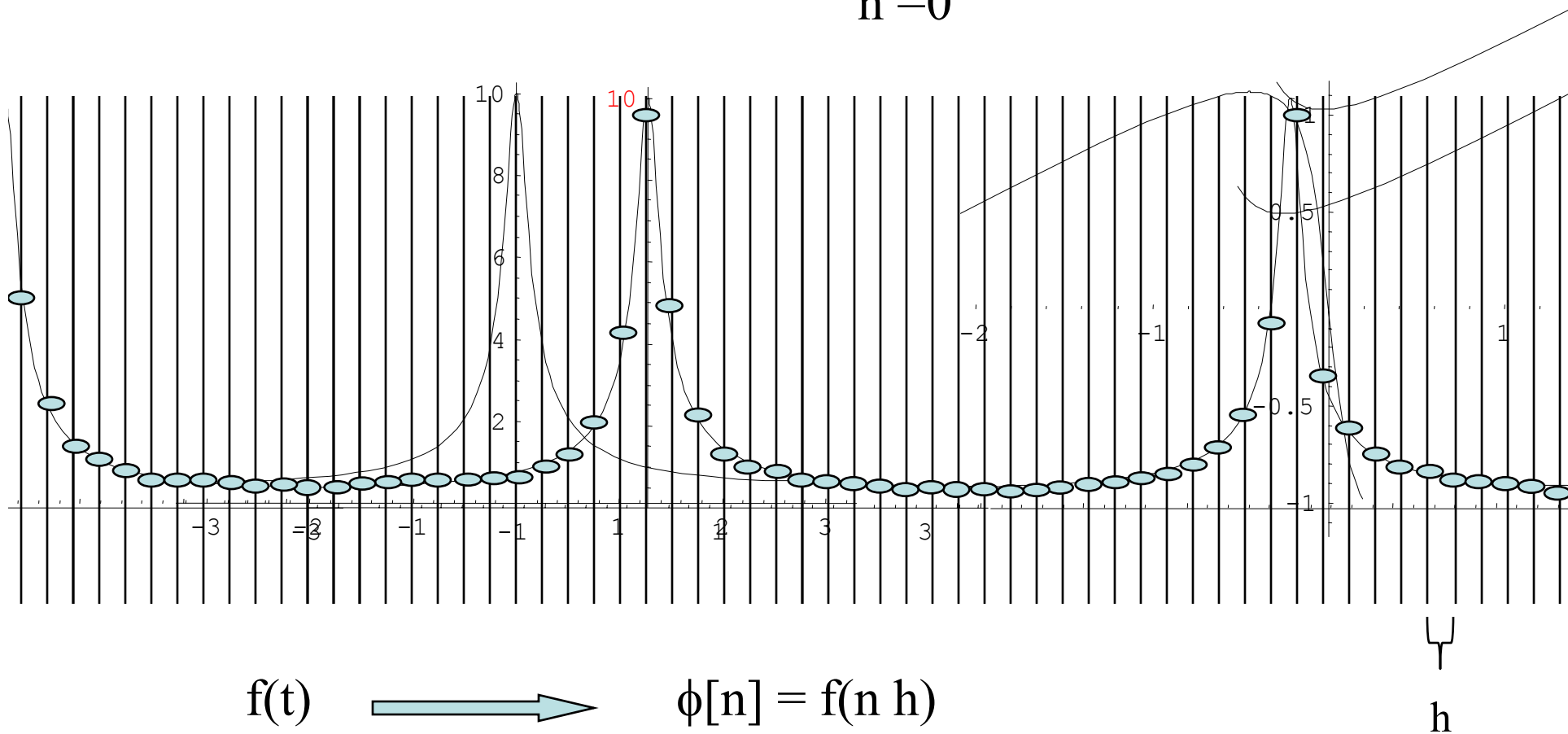
In this case, it is a periodic function on  $\mathbb{Z}$ ,

(Assuming  $p/h = N$ )



# DFT

$$\int_0^p f(t) e^{-2\pi i k t/p} dt \longrightarrow \sum_{n=0}^{N-1} \phi[n] e^{-2\pi i k n h/p}$$



# DFT and its inverse for periodic discrete data

$$\begin{aligned}\Phi[k] &= \sum_{n=0}^{N-1} \phi[n] e^{-2 \pi i k n h / p} & p &= N h \\ &= \sum_{n=0}^{N-1} \phi[n] e^{-2 \pi i k n / N}\end{aligned}$$

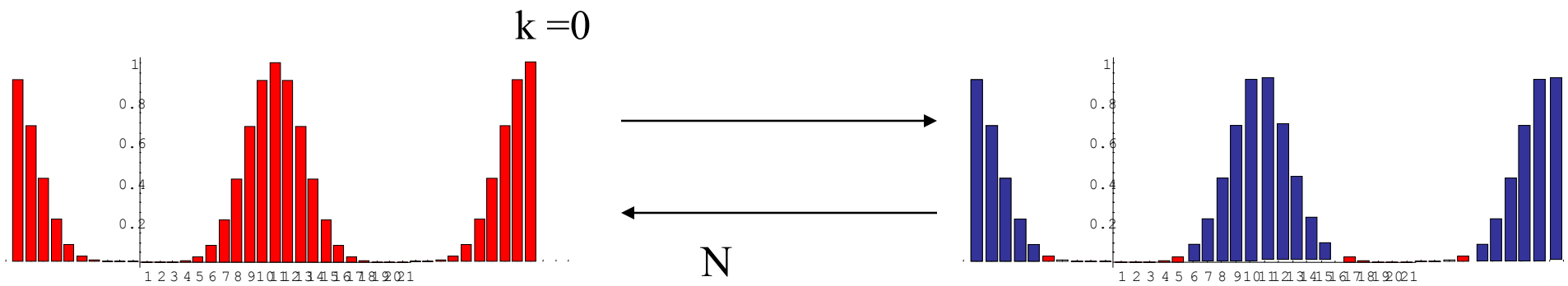
This is automatically periodic in  $k$  with period  $N$   
Inverse is like Fourier series, but with only  $p$  terms

# DFT: Discrete time periodic version of Fourier

*“time” domain*

*“frequency” domain*

$$\frac{1}{N} \sum_{k=0}^{N-1} \gamma[k] e^{-2 \pi i k m/N} = \Gamma [m]$$



$$\gamma[k] = \sum_{m=0}^{N-1} \Gamma [m] e^{2 \pi i m k/N}$$

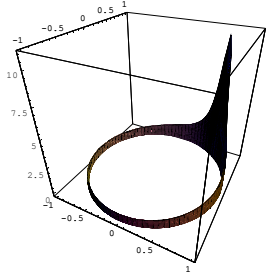
$\gamma[k]$ , on  $P_N$

i.e. on  $\mathbb{Z}$ , Period N

$\Gamma [m]$  on  $P_N$

i.e. on  $\mathbb{Z}$ , Period N

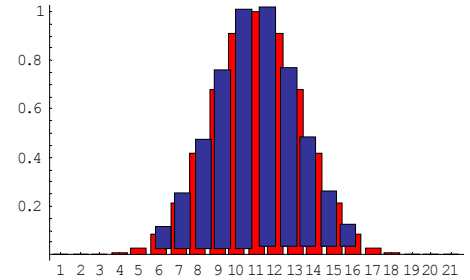
# Two PERIODIC time versions of Fourier



$f(t)$ , period  $p$

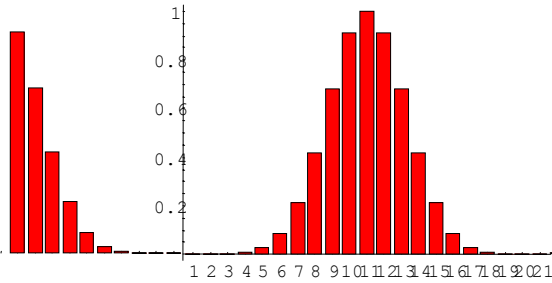
$$\int_0^p f(t) e^{-2\pi i k t/p} dt/p$$

$$\sum_{k \in \mathbb{Z}} F[k] e^{2\pi i k t/p}$$



$F[k]$ , on  $\mathbb{Z}$

$S_{p/N}$



$\chi[k]$ , on  $P_N$

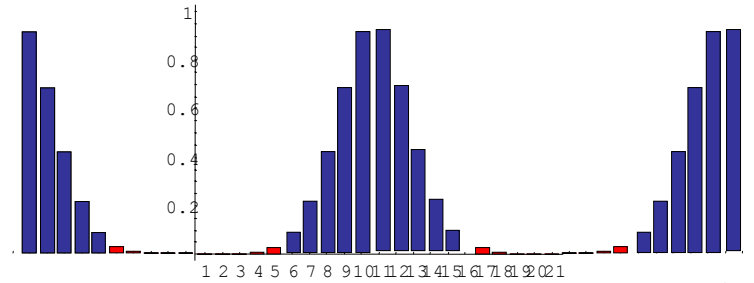
i.e. on  $\mathbb{Z}$ , Period  $N$

$$1/N \sum_{k=0}^{N-1} \gamma[k] e^{-2\pi i k m/N}$$

$k=0$

$$\sum_{k=0}^{N-1} \Gamma[k] e^{2\pi i m k/N}$$

$k=0$



$\Gamma[m]$  on  $P_N$

i.e. on  $\mathbb{Z}$ , Period  $N$

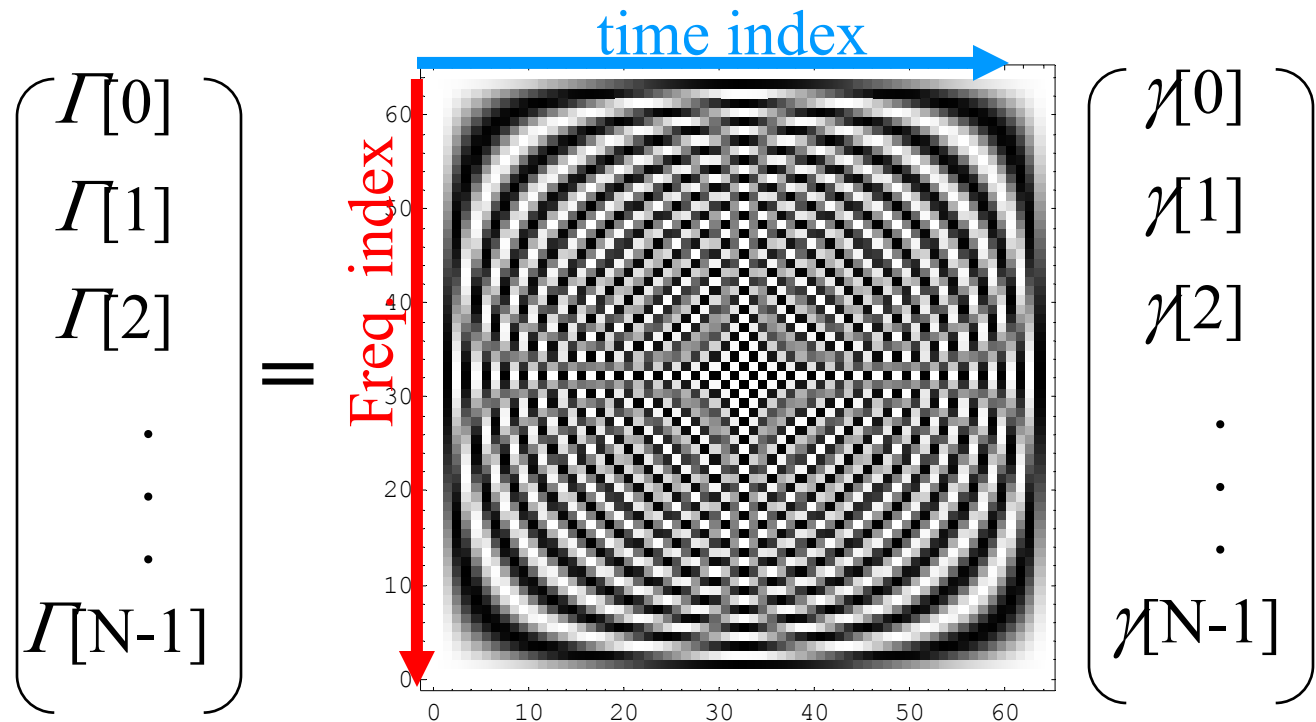
*“time” domain*

*“frequency” domain*

# Discrete time Numerical Fourier Analysis

DFT is really just a matrix multiplication!

$$\Gamma[m] = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i k m/N} \mathcal{X}[k]$$

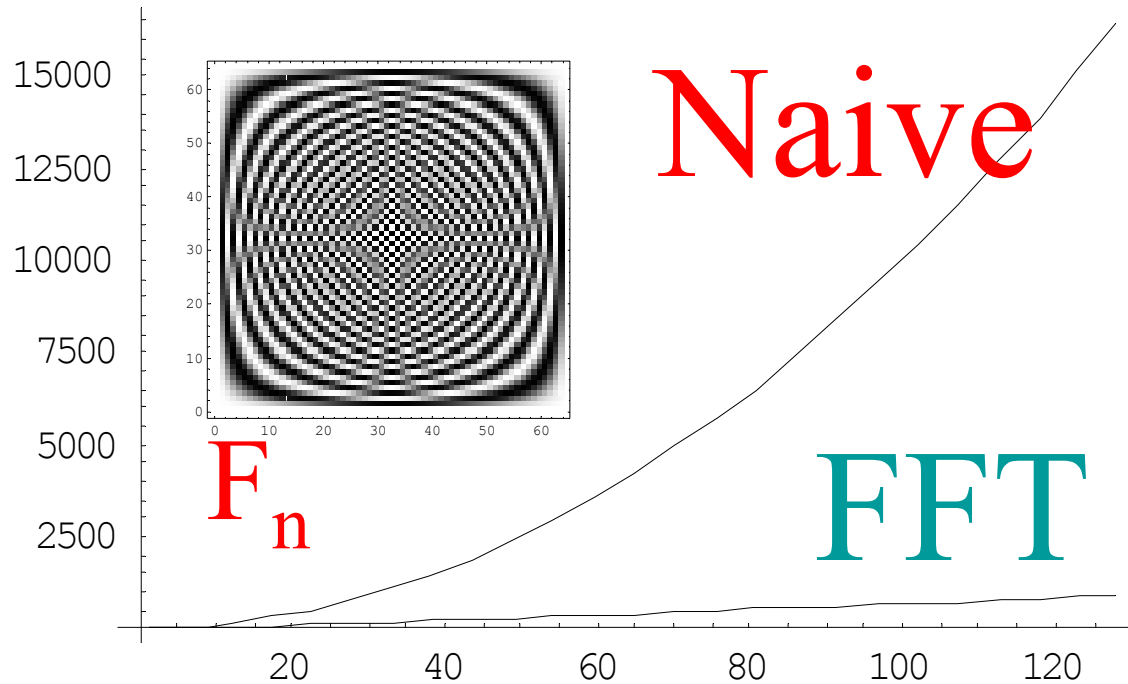


$$\Gamma = F_N \mathcal{X}$$

# Numerical Harmonic Analysis

FFT: Symmetry Properties permits “Divide and Conquer”  
Sparse Factorization

$$F_{mn} = (F_m \otimes I_n) \cdot T_n^{mn} \cdot (I_m \otimes F_n) \cdot L_m^{mn}$$



# Structured matrices

- Fast algorithms have been found for many dense matrices
- Typically the matrices have some “*structure*”
- Definition:
  - A dense matrix of order  $N \times N$  is called structured if its entries depend on only  $O(N)$  parameters.
- Most famous example – the fast Fourier transform

# Fourier Matrices

A Fourier matrix of order  $n$  is defined as the following

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix},$$

where

$$\omega_n = e^{-\frac{2\pi i}{n}},$$

is an  $n$ th root of unity.

A Fourier matrix of order  $n$  is defined as the following

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W & W^2 & \dots & W^{n-1} \\ 1 & W^2 & W^4 & \dots & W^{2(n-1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & W^{n-1} & W^{2(n-1)} & \dots & W^{(n-1)(n-1)} \end{bmatrix},$$

where

$$W = e^{\frac{2\pi i}{n}},$$

is an  $n$ th root of unity.

$$i = \sqrt{-1}$$

# Primitive Roots of Unity

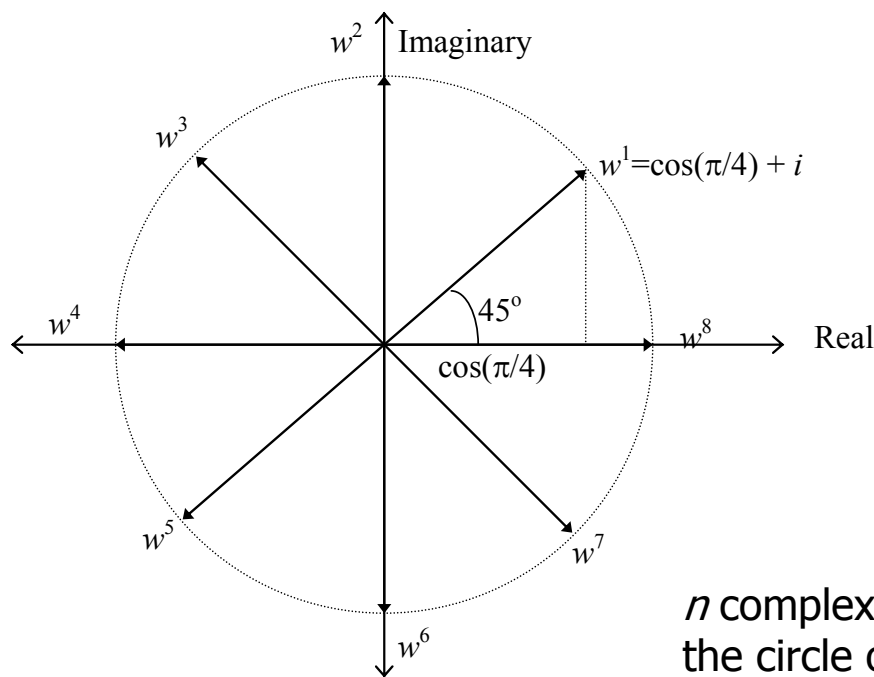
A number  $\omega$  is a *primitive n-th root of unity*, for  $n > 1$ , if

$$\omega^n = 1$$

The numbers  $1, \omega, \omega^2, \dots, \omega^{n-1}$  are all distinct

- Example: The complex number  $e^{2\pi i/n}$  is a primitive n-th root of unity, where

Check: if properties are satisfied



$$1. \omega^1 = e^{\frac{2\pi i}{n}} \neq 1$$

$$2. \omega^n = \left( e^{\frac{2\pi i}{n}} \right)^n = e^{2\pi i} = \cos 2\pi + i \sin 2\pi = 1$$

$$3. S = \sum_{p=0}^{n-1} \omega^{jp} = \omega^0 + \omega^j + \omega^{2j} + \omega^{3j} + \dots + \omega^{j(n-1)} = 0$$

$n$  complex roots of unity equally spaced around the circle of unit radius centered at the origin of the complex plane.

# Roots of Unity: Properties

- Property 1: Let  $\omega$  be the principal  $n^{\text{th}}$  root of unity. If  $n > 0$ , then  $\omega^{n/2} = -1$ .
  - Proof:  $\omega = e^{2\pi i / n} \Rightarrow \omega^{n/2} = e^{\pi i} = -1$ . (Euler's formula)
  - **Reflective Property:**
  - Corollary:  $\omega^{k+n/2} = -\omega^k$ .
- Property 2: Let  $n > 0$  be even, and let  $\omega$  and  $\nu$  be the principal  $n^{\text{th}}$  and  $(n/2)^{\text{th}}$  roots of unity. Then  $(\omega^k)^2 = \nu^k$ .
  - Proof:  $(\omega^k)^2 = e^{(2k)2\pi i / n} = e^{(k)2\pi i / (n/2)} = \nu^k$ .
  - **Reduction Property:** If  $\omega$  is a primitive  $(2n)$ -th root of unity, then  $\omega^2$  is a primitive  $n$ -th root of unity.

- L3: Let  $n > 0$  be even. Then, the squares of the  $n$  complex  $n^{\text{th}}$  roots of unity are the  $n/2$  complex  $(n/2)^{\text{th}}$  roots of unity.
  - Proof: If we square all of the  $n^{\text{th}}$  roots of unity, then each  $(n/2)^{\text{th}}$  root is obtained exactly twice since:
    - L1  $\Rightarrow \omega^{k+n/2} = -\omega^k$
    - thus,  $(\omega^{k+n/2})^2 = (\omega^k)^2$
    - L2  $\Rightarrow$  both of these  $= \omega^k$
    - $\omega^{k+n/2}$  and  $\omega^k$  have the same square
- **Inverse Property:** If  $\omega$  is a primitive root of unity, then  $\omega^{-1} = \omega^{n-1}$ 
  - Proof:  $\omega\omega^{n-1} = \omega^n = 1$

# Fast Fourier Transform

- Presented by Cooley and Tukey in 1965, but invented several times, including by Gauss (1809) and Danielson & Lanczos (1948)
- Danielson Lanczos lemma

$$\begin{aligned} F_k &= \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j \\ &= \sum_{j=0}^{N/2-1} e^{2\pi ik(2j)/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi ik(2j+1)/N} f_{2j+1} \\ &= \sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j+1} \\ &= F_k^e + W^k F_k^o \end{aligned}$$

- So far we have seen what happens on the right hand side
- How about the left hand side?
- When we split the sums in two we have two sets of sums with  $N/2$  quantities for  $N$  points.
- So the complexity is  $N^2/2 + N^2/2 = N^2$
- So there is no improvement
- Need to reduce the number of sums on the left hand side
  - We need to reduce the number of sums computed from  $2N$  to a lower number
  - Notice that the values corresponding to  $k$  and  $k+N/2$  will be the same.
  - The transforms  $F_e^k$  and  $F_o^k$  are periodic in  $k$  with length  $N/2$ .
  - So we need only compute half of them!

# FFT

- So DFT of order  $N$  can be expressed as sum of two DFTs of order  $N/2$  *evaluated at*  $N/2$  points
- Does this improve the complexity?
- Yes  $(N/2)^2 + (N/2)^2 = N^2/2 < N^2$
- But we are not done ....
- Can apply the lemma recursively

$$F_k^e = F_k^{ee} + W^k F_k^{eo}, \quad F_k^o = F_k^{oe} + W^k F_k^{oo},$$

- Finally we have a set of one point transforms
- One point transform is identity

$$F_k^{eooeooeoo\cdots oee} = f_n$$

# FFT Algorithm

```
FFT (n, a0, a1, a2, . . . , an-1)  
if (n == 1) // n is a power of 2  
    return a0  
  
ω ← e2π i / n  
(e0, e1, e2, . . . , en/2-1) ← FFT (n/2, a0, a2, a4, . . . , an-2)  
(d0, d1, d2, . . . , dn/2-1) ← FFT (n/2, a1, a3, a5, . . . , an-1)  
  
for k = 0 to n/2 - 1  
    yk ← ek + ωk dk  
    yk+n/2 ← ek - ωk dk  
  
return (y0, y1, y2, . . . , yn-1)
```

← O(n) complex multiplies  
if we pre-compute ω<sup>k</sup>.

$$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

# Complexity

- Each  $F_k$  is a sum of  $\log_2 N$  transforms and (factors)
- There are  $N$   $F_k$  s
- So the algorithm is  $O(N \log_2 N)$
- *This is a recursive algorithm*

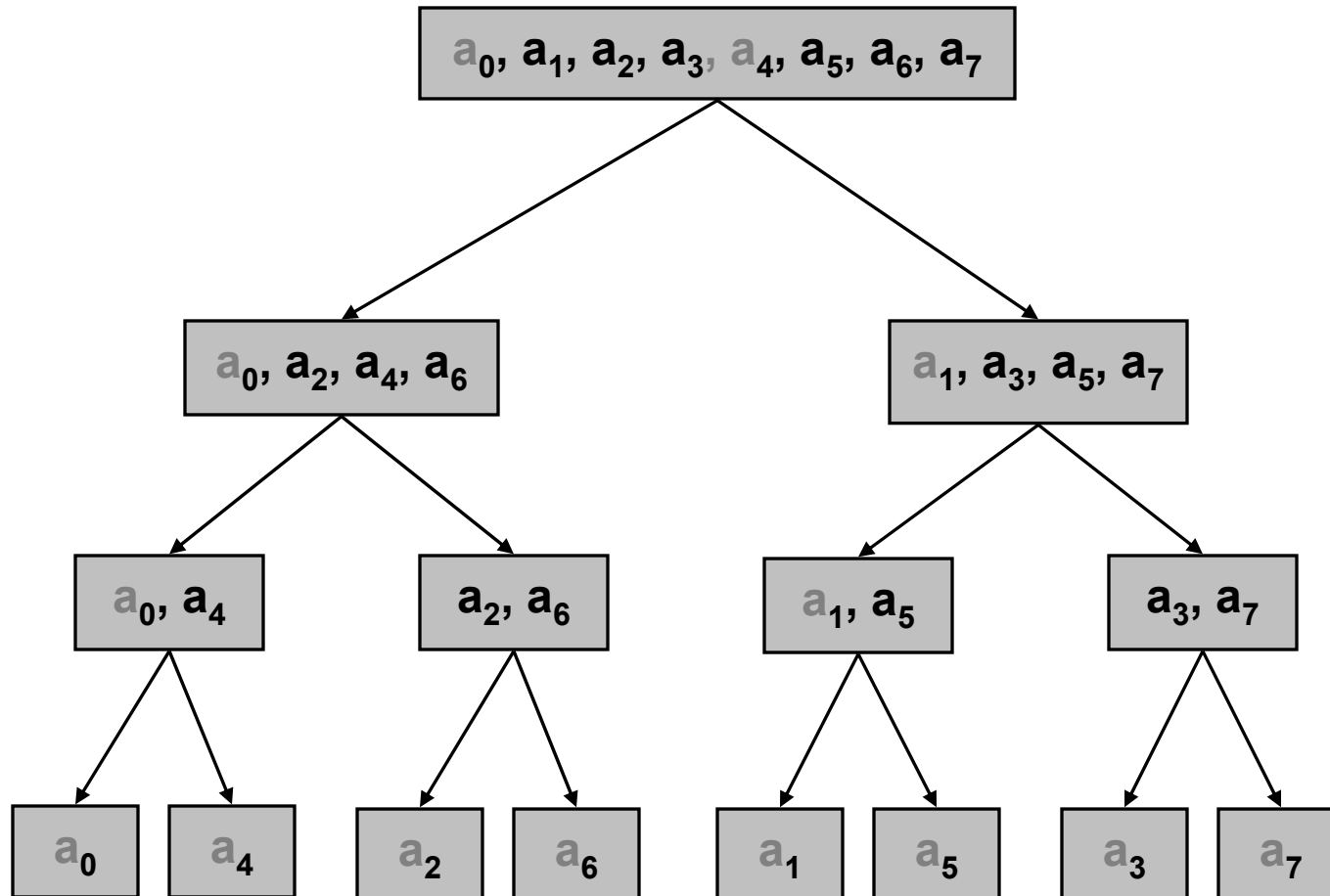
# FFTtx

```
function y = ffttx(x)
%FFTTX Textbook Fast Finite
  Fourier Transform.
%  FFTTX(X) computes the same
  finite Fourier transform as FFT(X).
%  The code uses a recursive divide
  and conquer algorithm for
%  even order and matrix-vector
  multiplication for odd order.
%  If length(X) is m*p where m is
  odd and p is a power of 2, the
%  computational complexity of this
  approach is  $O(m^2)*O(p*\log_2(p))$ .

x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0
  % Recursive divide and conquer
  k = (0:n/2-1)';
  w = omega .^ k;
  u = ffttx(x(1:2:n-1));
  v = w.*ffttx(x(2:2:n));
  y = [u+v; u-v];
else
  % The Fourier matrix.
  j = 0:n-1;
  k = j';
  F = omega .^ (k*j);
  y = F*x;
end
```

# Scrambled Output of the FFT



**"bit-reversed" order**

# FFT and IFFT

The *discrete Fourier transform* of a vector  $x$  is the product  $F_n x$ .

The *inverse discrete Fourier transform* of a vector  $x$  is the product  $F_n^* x$ .

Both products can be done efficiently using the fast Fourier transform (FFT) and the inverse fast Fourier transform (IFFT) in  $O(n \log n)$  time.

The FFT has revolutionized many applications by reducing the complexity by a factor of almost  $n$

Can relate many other matrices to the Fourier Matrix

## Circulant Matrices

$$C_n = C(x_1, \dots, x_n) = \begin{bmatrix} x_1 & x_n & x_{n-1} & \cdots & x_2 \\ x_2 & x_1 & x_n & \cdots & x_3 \\ x_3 & x_2 & x_1 & \cdots & x_4 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_n & x_{n-1} & x_{n-2} & \cdots & x_1 \end{bmatrix}$$

## Toeplitz Matrices

$$T_n = T(x_{-n+1}, \dots, x_0, \dots, x_{n-1}) = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_{-1} & x_0 & x_1 & \cdots & x_{n-2} \\ x_{-2} & x_{-1} & x_0 & \cdots & x_{n-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \end{bmatrix}$$

## Hankel Matrices

$$H_n = H(x_{-n+1}, \dots, x_0, \dots, x_{n-1}) = \begin{bmatrix} x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \\ x_{-n+2} & x_{-n+3} & x_{-n+4} & \cdots & x_1 \\ x_{-n+3} & x_{-n+4} & x_{-n+5} & \cdots & x_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_0 & x_1 & x_2 & \cdots & x_{n-1} \end{bmatrix}$$

## Vandermonde Matrices

$$V = V(x_0, x_1, \dots, x_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ x_0^{n-1} & x_1^{n-1} & \cdots & x_{n-1}^{n-1} \end{bmatrix}$$

- Modern signal processing very strongly based on the FFT
- One of the defining inventions of the 20<sup>th</sup> century