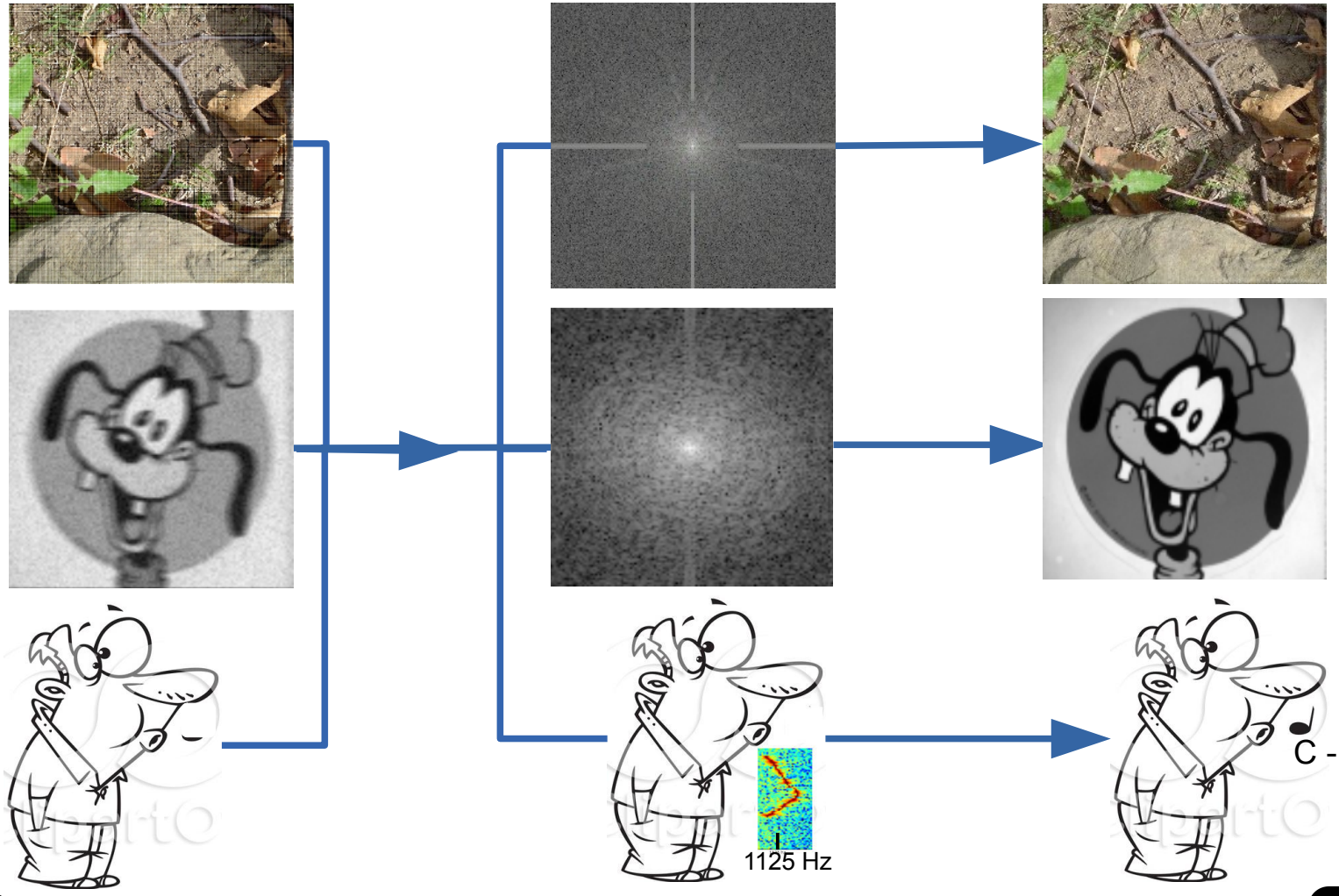
Fast Fourier Transforms



Analabha Roy

Department of Physics, The University of Burdwan

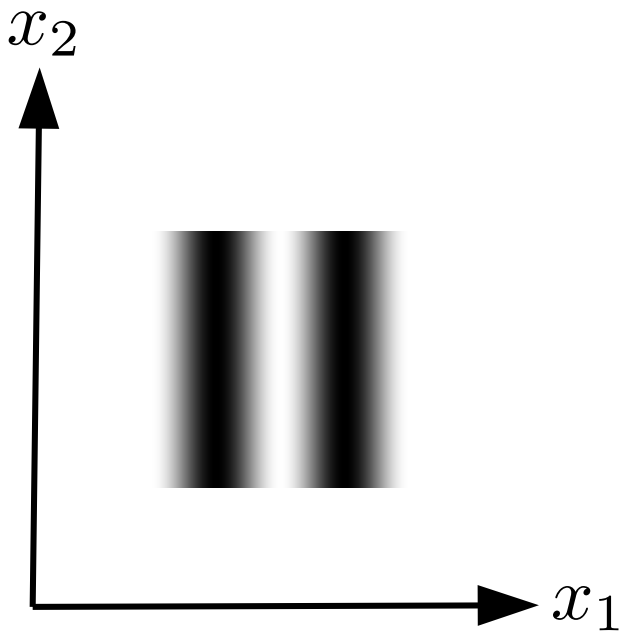
https://bit.ly/fft_lab

https://bit.ly/fft_lab

1. Introduction

Intuitive explanation of Fourier Theory

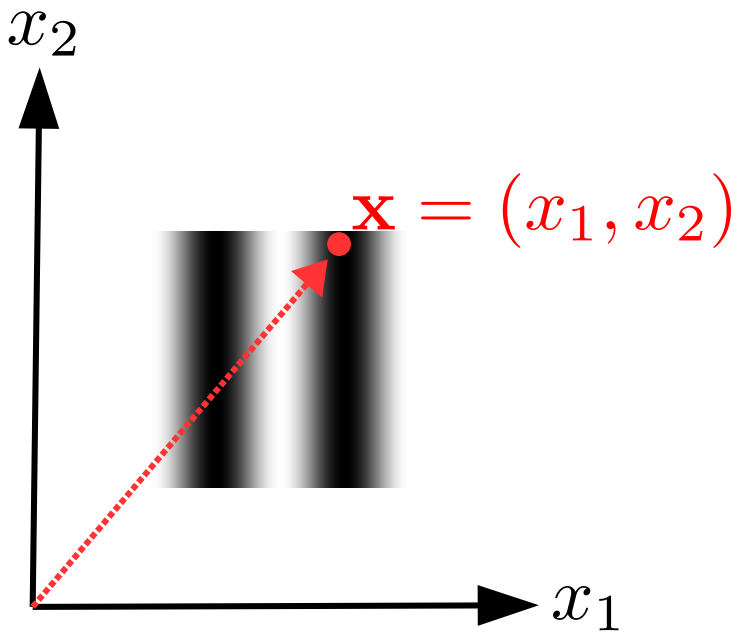
Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a formal sum of sinusoids



1. Introduction

Intuitive explanation of Fourier Theory

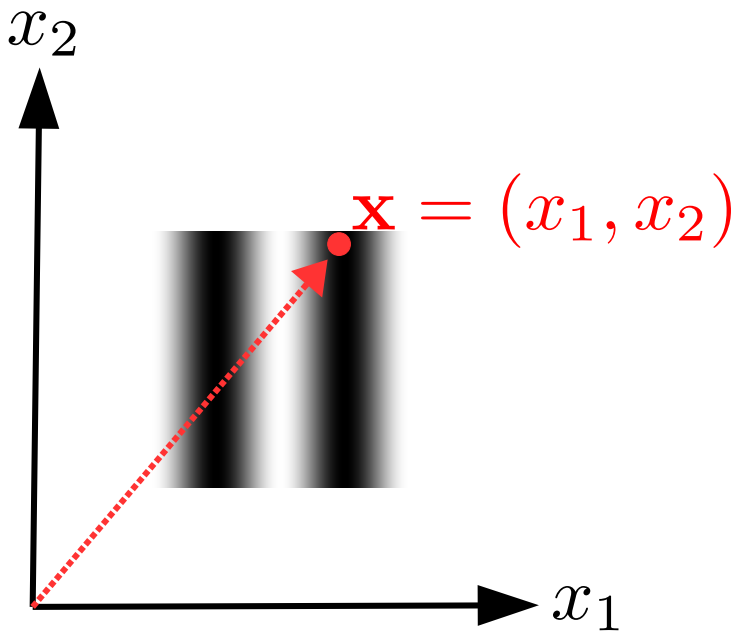
Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a formal sum of sinusoids



1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a formal sum of sinusoids



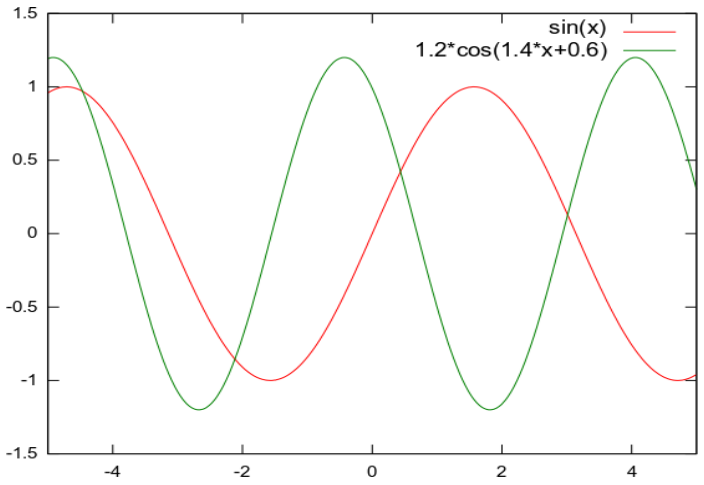
$$I(\mathbf{x}) = I_0 \sin(\mathbf{k} \cdot \mathbf{x} + \phi) = I_0 \sin(k_1 x_1 + k_2 x_2 + \phi)\ k_2 \simeq 0$$

1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)

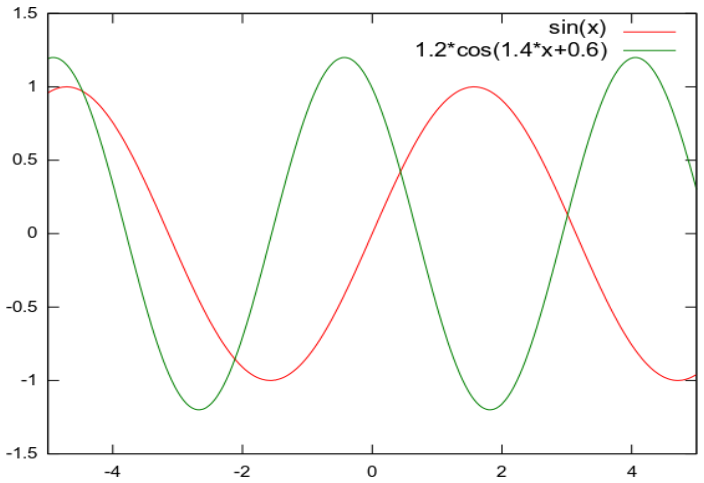


1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)



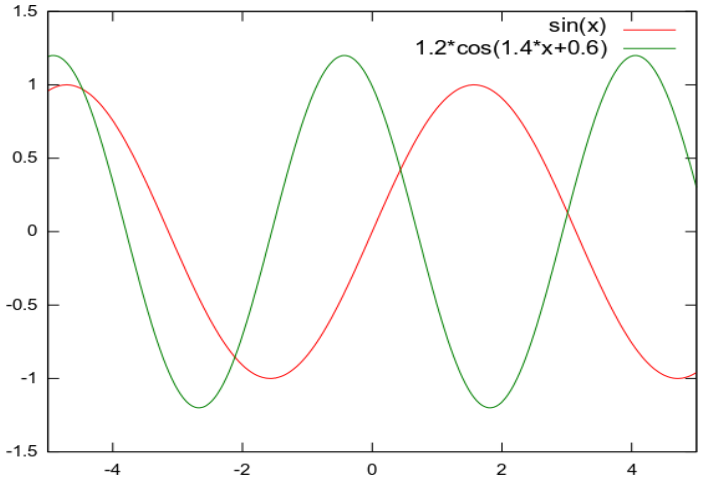
Add the components

1. Introduction

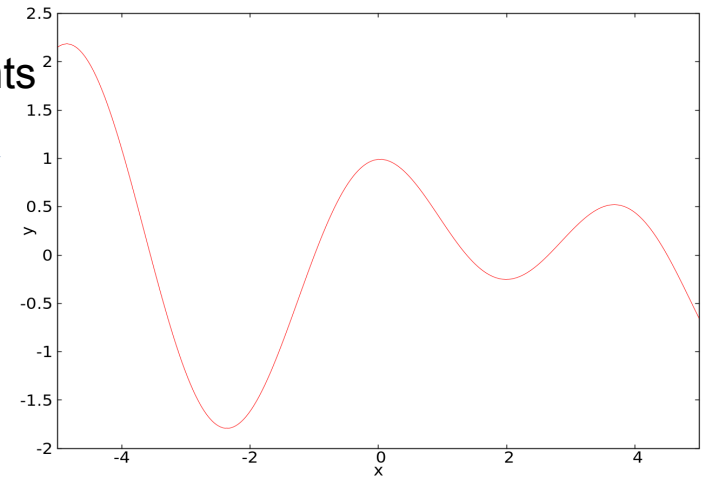
Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)



Add the components

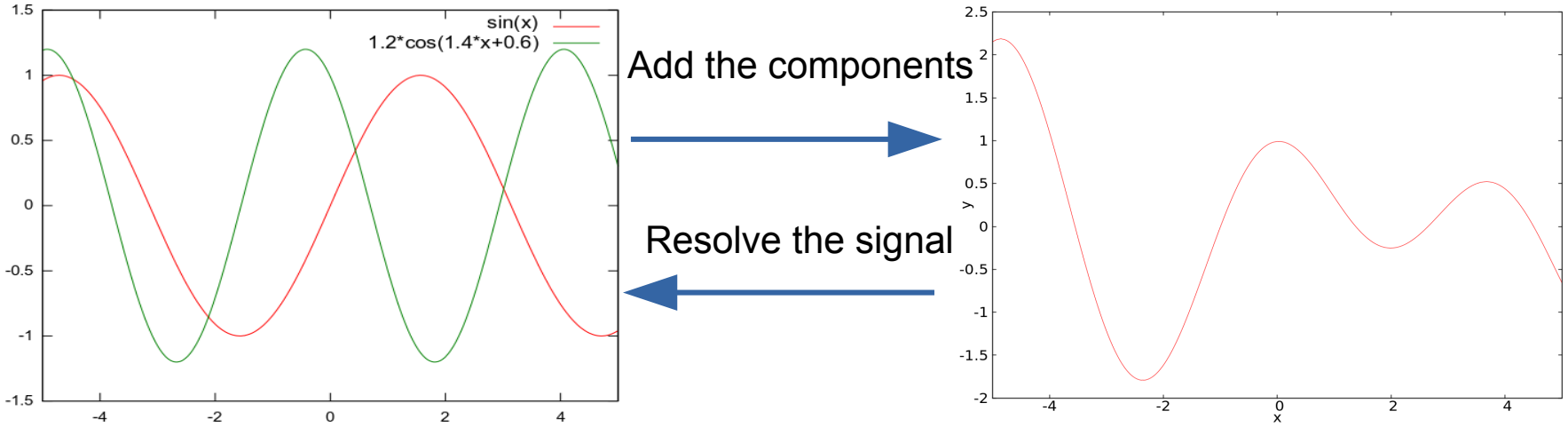


1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)

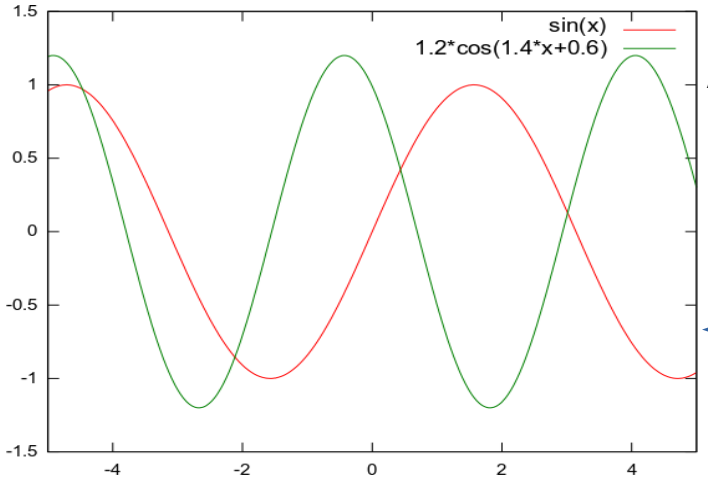


1. Introduction

Intuitive explanation of Fourier Theory

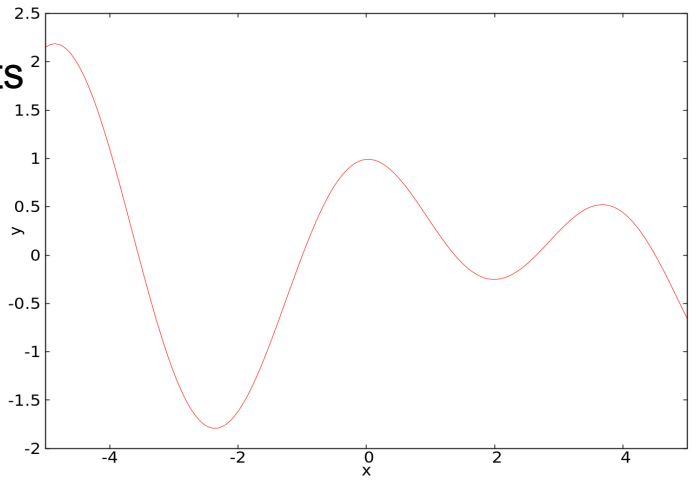
Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)



Add the components

Resolve the signal



$$\sin x = \frac{1}{2i}e^{ix} - \frac{1}{2i}e^{-ix} = \frac{1}{2}e^{-i\pi/2}e^{ix} + \frac{1}{2}e^{i\pi/2}e^{-ix}$$

$$1.2\ \cos(1.4\,x + 0.6) = \left\{\frac{1.2}{2}\ e^{i(0.6)}\ e^{i(1.4\,x)}\right\} + \left\{\frac{1.2}{2}\ e^{-i(0.6)}\ e^{-i(1.4\,x)}\right\}$$

Fourier Transform:

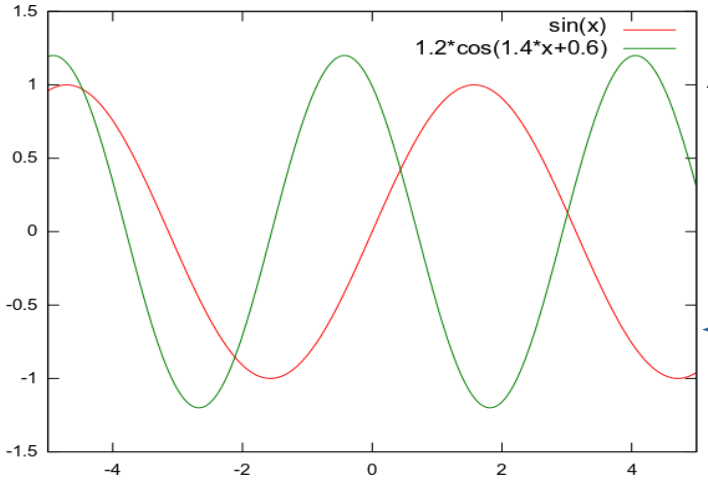
Plot of Amplitudes (Phases) as functions of resolved frequencies

1. Introduction

Intuitive explanation of Fourier Theory

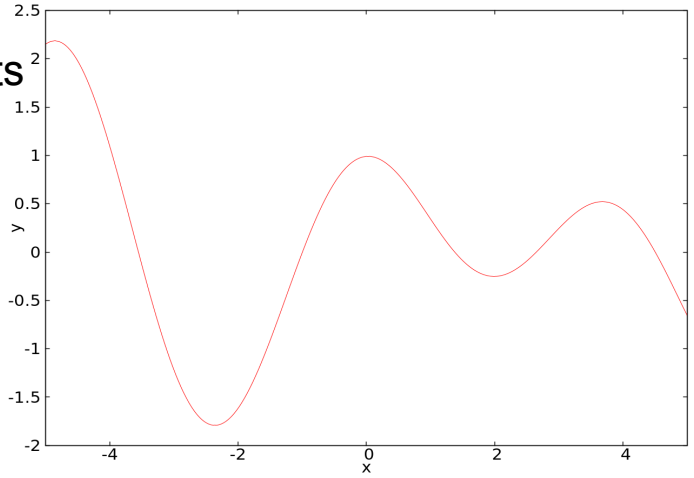
Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

1 – Dimensional Example (we'll stick to 1D for the rest of this session)



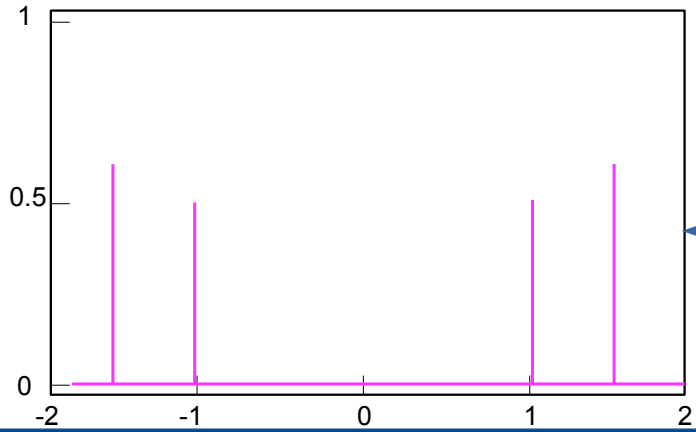
Add the components

Resolve the signal



$$\sin x = \frac{1}{2i}e^{ix} - \frac{1}{2i}e^{-ix} = \frac{1}{2}e^{-i\pi/2}e^{ix} + \frac{1}{2}e^{i\pi/2}e^{-ix}$$

$$1.2\ \cos(1.4\,x + 0.6) = \left\{\frac{1.2}{2}\ e^{i(0.6)}\ e^{i(1.4\,x)}\right\} + \left\{\frac{1.2}{2}\ e^{-i(0.6)}\ e^{-i(1.4\,x)}\right\}$$



Fourier Transform:

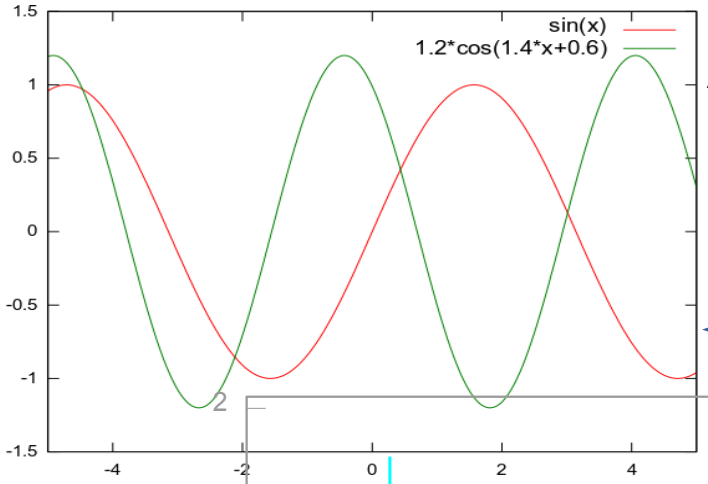
Plot of Amplitudes (Phases) as functions of resolved frequencies

1. Introduction

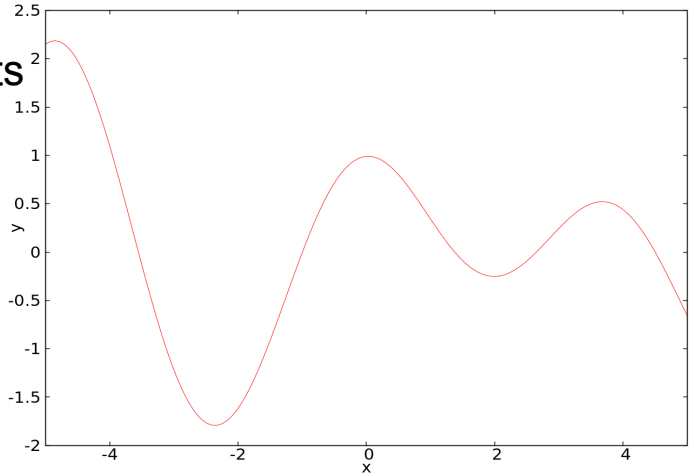
Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

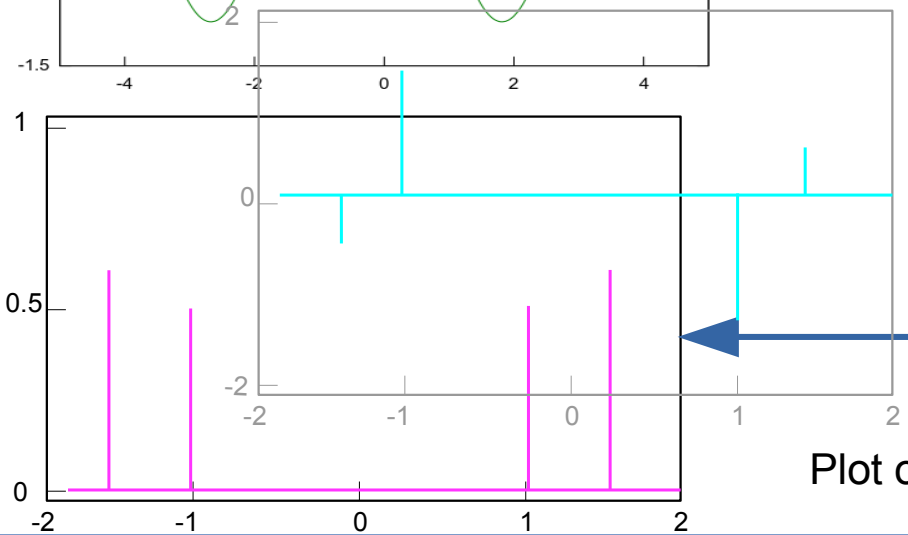
1 – Dimensional Example (we'll stick to 1D for the rest of this session)



Add the components



Resolve the signal



$$\sin x = \frac{1}{2i}e^{ix} - \frac{1}{2i}e^{-ix} = \frac{1}{2}e^{-i\pi/2}e^{ix} + \frac{1}{2}e^{i\pi/2}e^{-ix}$$

$$1.2\ \cos(1.4\,x + 0.6) = \left\{\frac{1.2}{2}\ e^{i(0.6)}\ e^{i(1.4\,x)}\right\} + \left\{\frac{1.2}{2}\ e^{-i(0.6)}\ e^{-i(1.4\,x)}\right\}$$

Fourier Transform:

Plot of Amplitudes (Phases) as functions of resolved frequencies

1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

2 – Dimensional Examples (we'll stick to 1D for the rest of this session)

1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

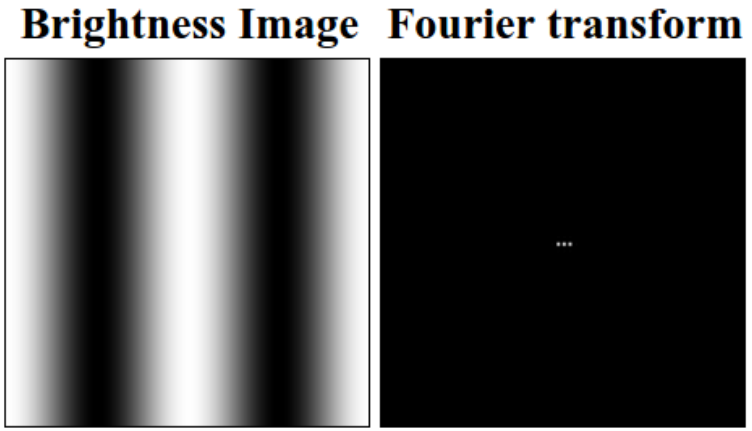
2 – Dimensional Examples (we'll stick to 1D for the rest of this session)

1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

2 – Dimensional Examples (we'll stick to 1D for the rest of this session)

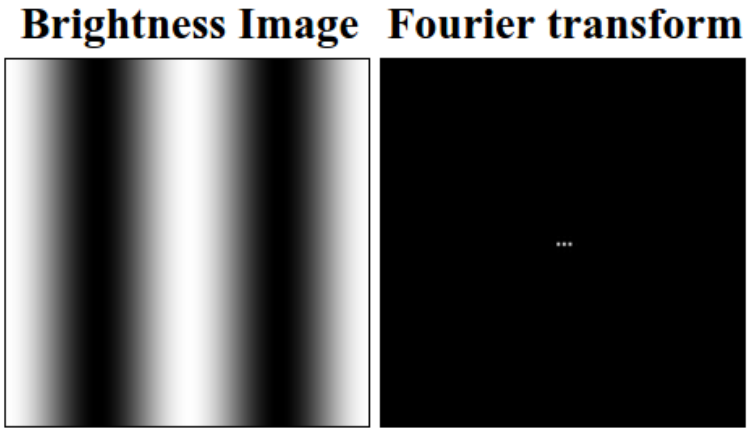


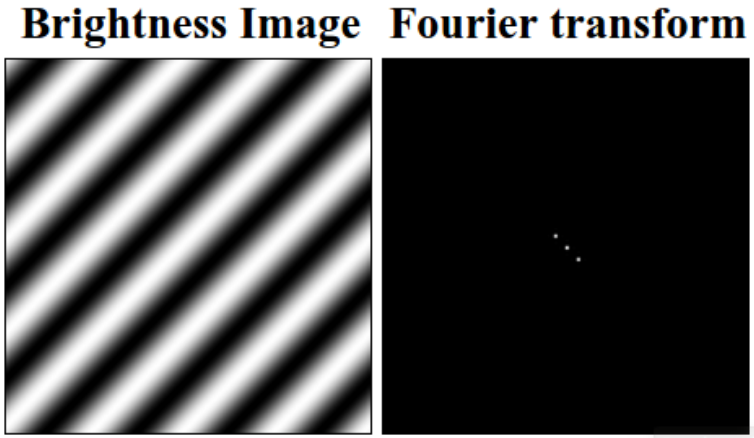
1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

2 – Dimensional Examples (we'll stick to 1D for the rest of this session)



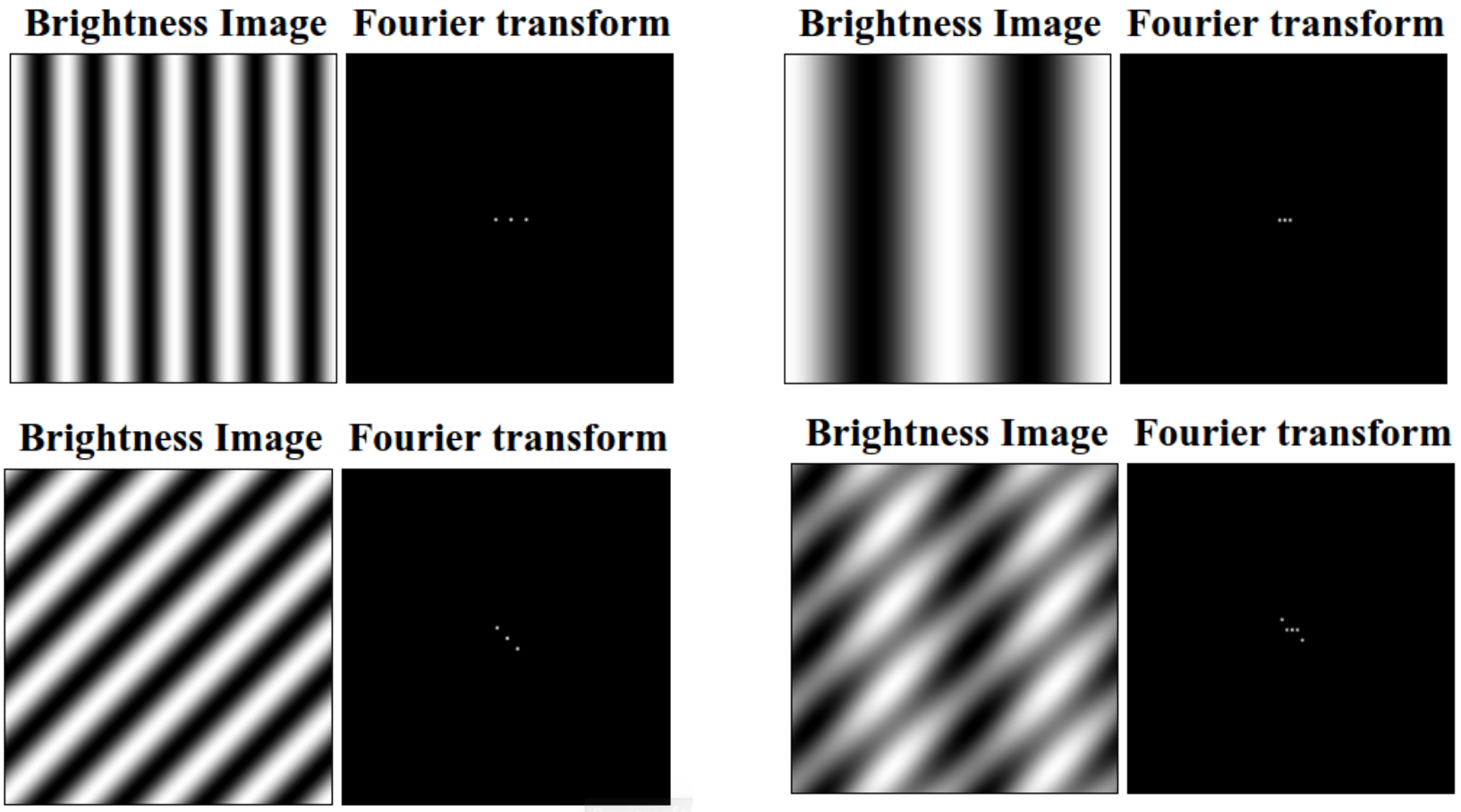


1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

2 – Dimensional Examples (we'll stick to 1D for the rest of this session)



1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

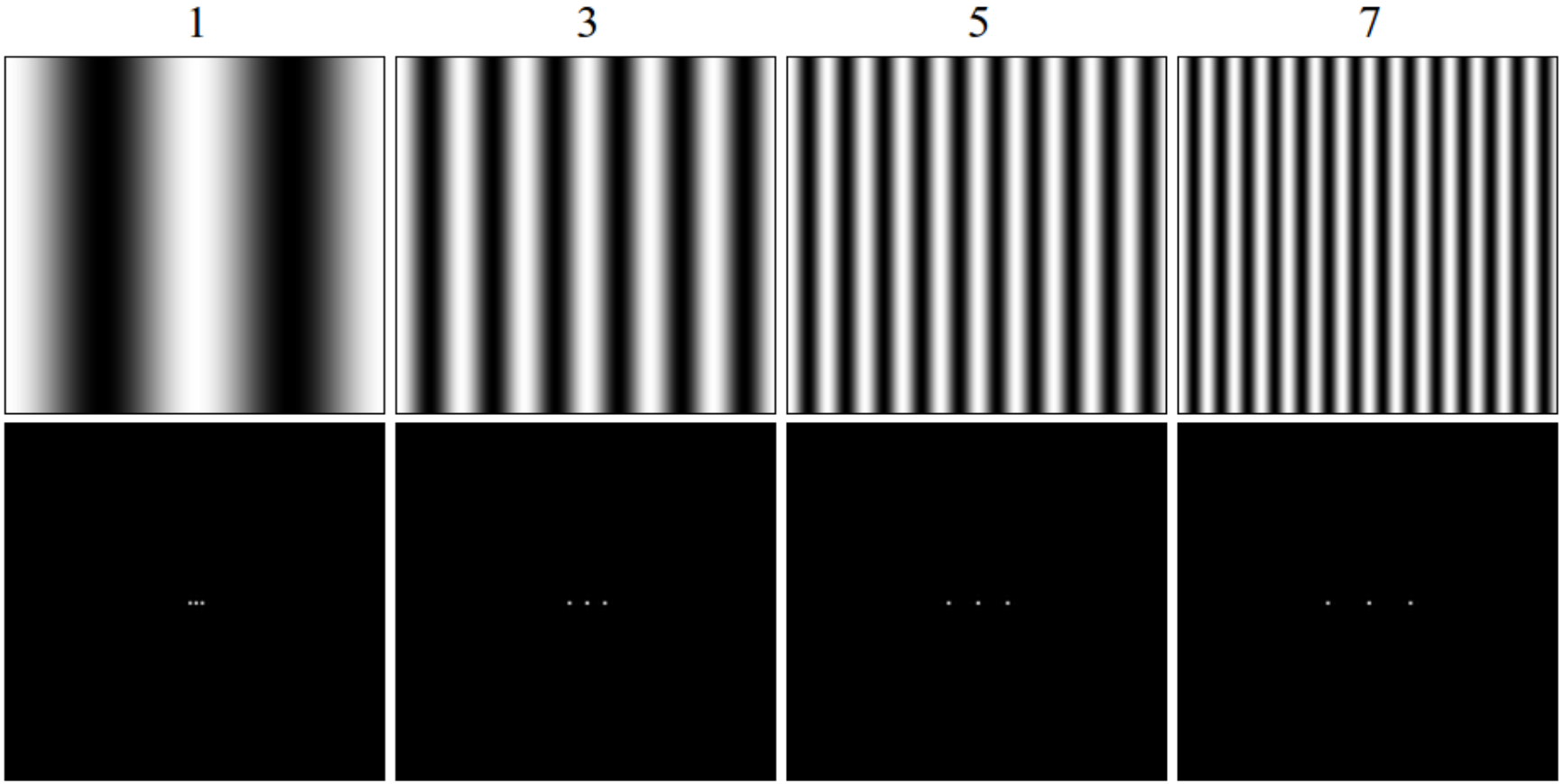
Higher Harmonics – If signal is “finite” (in extent) or has sharp discontinuities

1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

Higher Harmonics – If signal is “finite” (in extent) or has sharp discontinuities

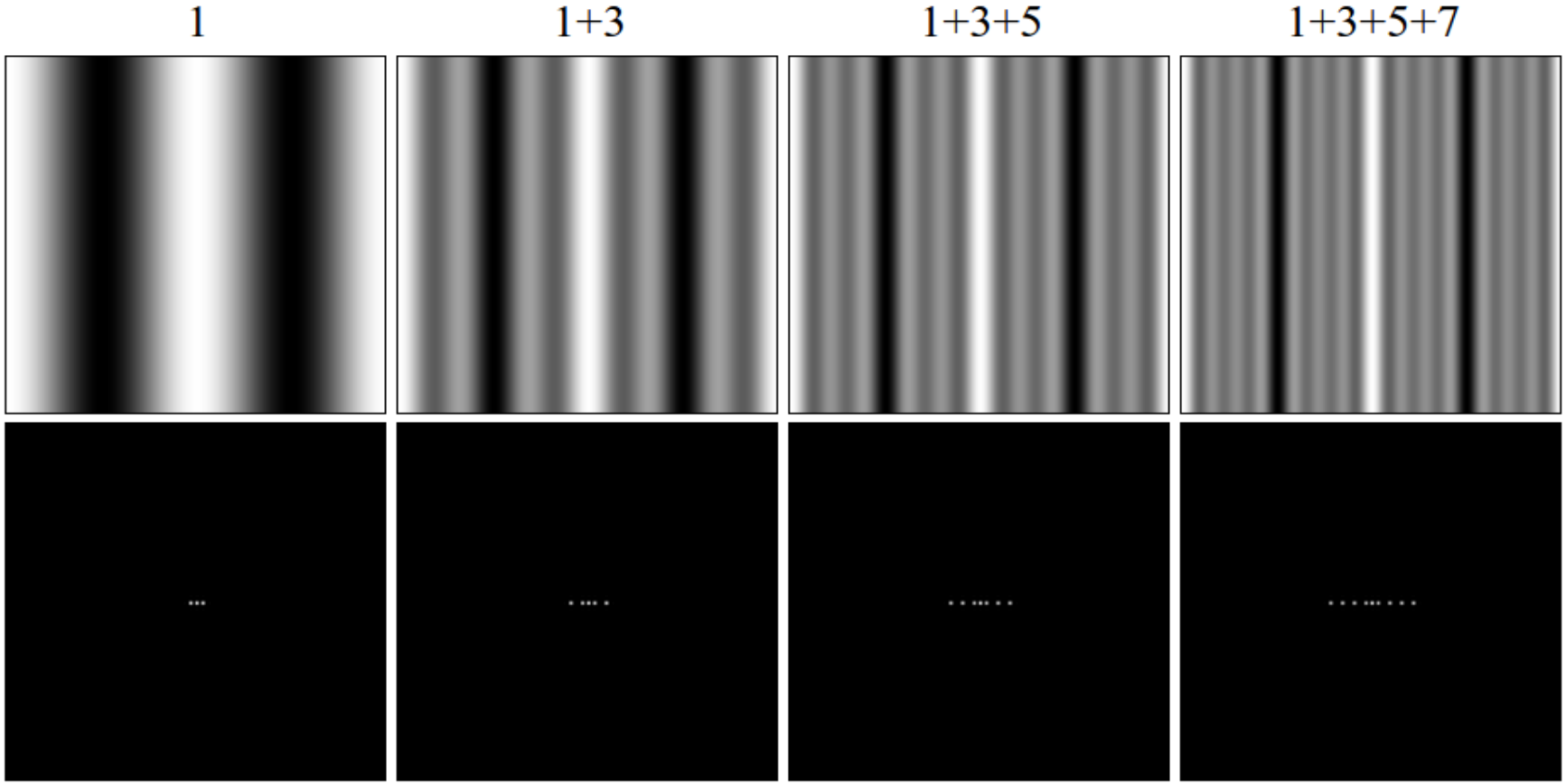


1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

Higher Harmonics – If signal is “finite” (in extent) or has sharp discontinuities



1. Introduction

Intuitive explanation of Fourier Theory

Any signal (Spatially and/or temporally varying data like images, sounds etc.) can be expressed as a sum of series of sinusoids

Higher Harmonics – If signal is “finite” (in extent) or has sharp discontinuities

2. Theoretical Section:

- Formal Definition of Fourier Transform (FT):

  Suppose $f(x)$ is absolutely integrable in $(-\infty, \infty)$, then the FT is

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

- Moreover, if the function is square integrable, then the inverse FT (IFT) is

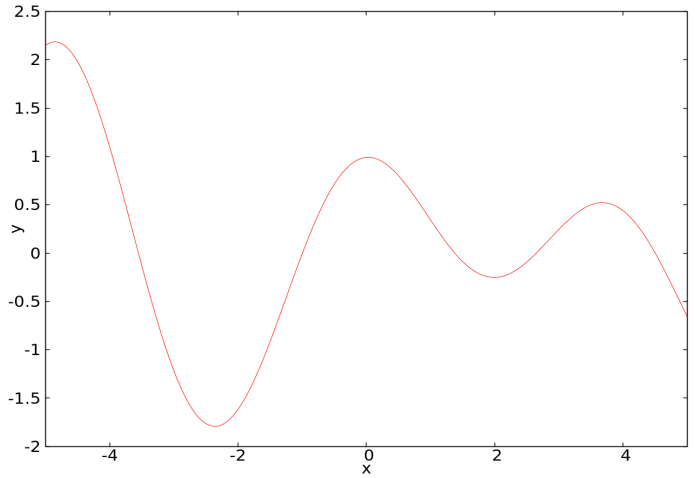
$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{d}\omega \; \bar{f}(\omega) \; e^{i\omega t}$$

2. Theoretical Section:

- Formal Definition of Fourier Transform (FT):

  Suppose $f(x)$ is absolutely integrable in $(-\infty, \infty)$, then the FT is

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$



So if $f(t) = \sin t + a\cos(\Omega t + \phi)$

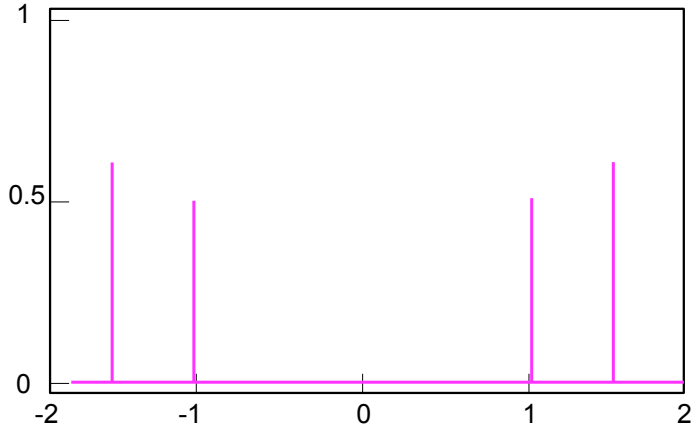
$$\bar{f}(\omega) = \frac{1}{2i}\int_{-\infty}^{\infty} \mathrm{d}t \; e^{i(1-\omega)t} - \frac{1}{2i}\int_{-\infty}^{\infty} \mathrm{d}t \; e^{-i(1+\omega)t} + \frac{ae^{i\phi}}{2}\int_{-\infty}^{\infty} \mathrm{d}t \; e^{i(\Omega-\omega)t} + \frac{ae^{-i\phi}}{2}\int_{-\infty}^{\infty} \mathrm{d}t \; e^{-i(\Omega+\omega)t}$$

2. Theoretical Section:

- Formal Definition of Fourier Transform (FT):

  Suppose $f(x)$ is absolutely integrable in $(-\infty, \infty)$, then the FT is

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$



So if $f(t) = \sin t + a \cos(\Omega t + \phi)$

$$\begin{aligned} \bar{f}(\omega) \quad = \quad & i\pi\delta(\omega + 1) + \pi a e^{i\phi}\delta(\omega - \Omega) - \\ & i\pi\delta(\omega - 1) - \pi a e^{-i\phi}\delta(\omega + \Omega) \end{aligned}$$

$$\delta(z) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{d}t \; e^{izt}$$

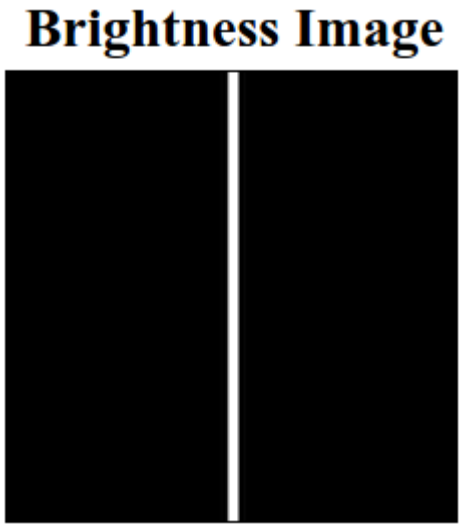
2. Theoretical Section:

- Formal Definition of Fourier Transform (FT):

  Suppose $f(x)$ is absolutely integrable in $(-\infty, \infty)$, then the FT is

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

So if $f(x,y) = \begin{cases} 1 & \text{if } -1 \leq x \leq 1 \\ 0 & \text{if } |x| > 1 \end{cases}$

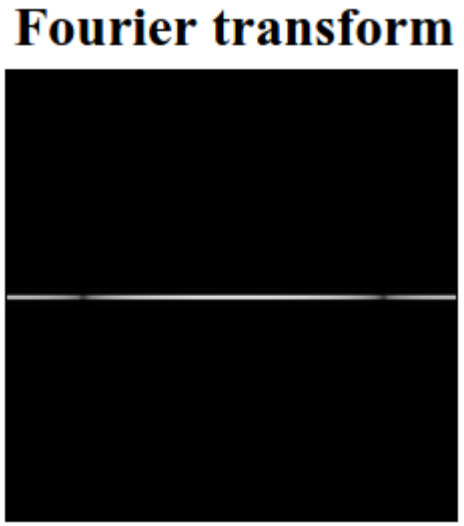


2. Theoretical Section:

- Formal Definition of Fourier Transform (FT):

  Suppose $f(x)$ is absolutely integrable in $(-\infty, \infty)$, then the FT is

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$



So if $f(x,y) = \begin{cases} 1 & \text{if } -1 \le x \le 1 \\ 0 & \text{if } |x| > 1 \end{cases}$

$$\begin{aligned} \bar{f}(\omega_x, \omega_y) &= \int_{-1}^{1} \mathrm{d}x \; e^{i\omega_x x} \int_{-\infty}^{\infty} \mathrm{d}y \; e^{i\omega_y y} \\ &= 4\pi \; \frac{\sin \omega_x}{\omega_x} \; \delta(\omega_y) \end{aligned}$$

2. Theoretical Section:

- Important Properties of Fourier Transform (FT):
  - 1. Derivative to Coefficient:

$$\overline{\frac{\mathrm{d}f}{\mathrm{d}t}} = i\omega \bar{f}$$

  - 2. Translation Property:

$$\overline{f(t-T)} = e^{-i\omega T}\,\bar{f}$$

  - 3. Convolution Property:

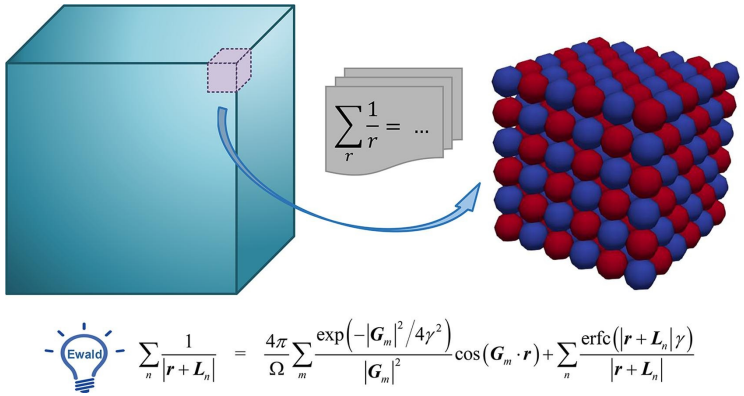
$$\overline{f_1 \otimes f_2} = \bar{f}_1\ \bar{f}_2$$

where $\{f_1 \otimes f_2\}(x) \equiv \int_{-\infty}^{\infty} \mathrm{d}t\ f_1(x-t) f_2(t)$

  - 4. Parseval's theorem:

$$\int_{-\infty}^{\infty} \mathrm{d}t\ |f(t)|^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathrm{d}\omega\ |\bar{f}(\omega)|^2$$

2. Theoretical Section:



- Important Properties of Fourier Transform (FT):
  - 3. Convolution Property:

$$\overline{f_1 \otimes f_2} = \bar{f}_1 \; \bar{f}_2$$

where $\{f_1 \otimes f_2\}(x) \equiv \int_{-\infty}^{\infty} \mathrm{d}t \; f_1(x-t) f_2(t)$

This property is especially powerful numerically, especially if $f_1(x-t) \sim 1/|x-t|^{\alpha}$

- Physicists will immediately recognize Coulomb-like electrostatic potential energy
- Need this to get lattice energies in Solid State Physics, as well as interactions in Molecular Dynamics (MD)
- However, the Coulomb Integral converges slowly, leading to numerical inefficiency
- However, the FFTs converge much faster, and even better if $\bar{f}_1$ is known analytically
- In physics, this is called the Ewald summation
- In computational (bio)chemistry, this is called the “Particle-Mesh Ewald” method
- Implemented in MD software such as GROMACS, NAMD, GAMESS CP2K etc. to compute long-range non-bonded interactions (Coulomb, Van-der-waals etc.)

2. Theoretical Section:

- Important Properties of Fourier Transform (FT):

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

If $f(t)$ ...	Then ...
is real	$\bar{f}(-\omega) = \bar{f}^*(\omega)$
is imaginary	$\bar{f}(-\omega) = -\bar{f}^*(\omega)$
is even	$\bar{f}(-\omega) = \bar{f}(\omega)$ *i.e.* so is $\bar{f}(\omega)$
is odd	$\bar{f}(-\omega) = -\bar{f}(\omega)$ *i.e.* so is $\bar{f}(\omega)$
is real and even	$\bar{f}(\omega)$ is real and even
is real and odd	$\bar{f}(\omega)$ is imaginary and odd
is imaginary and even	$\bar{f}(\omega)$ is imaginary and even
is imaginary and odd	$\bar{f}(\omega)$ is real and odd

3. The Discreet Fourier Transform (DFT)

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

- Given the function, sample it at discrete points and estimate the FT

  The value of the function is recorded at regular time intervals and assembled into a vector

$$|f\rangle = (f_1, f_2, f_3 \; \cdots \; f_N) \;\; , \;\; f_n = f(n\Delta)$$

$\Delta^{-1} =$ Sampling Rate

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \; \cdots \; \bar{f}_N) \;\; , \;\; \bar{f}_n = ?$$

- Given a discreet vector of data, estimate the FT

  The data given might not correspond to a function sampled at regular intervals

  So regularize it (estimate the function locally by interpolation and sample it as above)

3. The Discreet Fourier Transform (DFT)

$$\bar{f}(k) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t)\, e^{-2\pi i k t} \;,\; k \in (-\infty, \infty)$$

$$|f\rangle = (f_1, f_2, f_3 \; \cdots \; f_N) \;,\; f_n = f(n\Delta)$$

DFT

$$\bar{f}_m = \sum_{n=-N/2+1}^{N/2-1} \Delta \; f_n \, e^{-2\pi i n \Delta k_m} \;,\; k_m = \frac{m}{N\Delta}$$

$$|\bar{f}\rangle = \left(\bar{f}_{-N/2} \cdots \;, \bar{f}_{-1}, \bar{f}_0, \bar{f}_1 \; \cdots \; \bar{f}_{N/2}\right) \quad \left[-\frac{N}{2}, +\frac{N}{2}\right]$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}(k) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t)\, e^{-2\pi i k t} \;,\; k \in (-\infty, \infty)$$

$$|f\rangle = (f_1, f_2, f_3 \; \cdots \; f_N) \;,\; f_n = f(n\Delta)$$

DFT

$$\bar{f}_m = \sum_{n=-N/2+1}^{N/2-1} \Delta \; f_n \; e^{-2\pi i m n / N}$$

$$|\bar{f}\rangle = \left(\bar{f}_{-N/2} \cdots \;, \bar{f}_{-1}, \bar{f}_0, \bar{f}_1 \; \cdots \; \bar{f}_{N/2}\right) \quad \left[-\frac{N}{2}, +\frac{N}{2}\right]$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}(k) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t)\, e^{-2\pi i k t} \;,\; k \in (-\infty, \infty)$$

$$|f\rangle = (f_1, f_2, f_3 \cdots f_N) \;,\; f_n = f(n\Delta)$$

DFT

$$\bar{f}_m = \sum_{n=-N/2+1}^{N/2-1} \Delta f_n \, e^{-2\pi i m n/N}$$

$$|\bar{f}\rangle = \left(\bar{f}_{-N/2} \cdots \;, \bar{f}_{-1}, \bar{f}_0, \bar{f}_1 \; \cdots \; \bar{f}_{N/2}\right) \quad \left[-\frac{N}{2}, +\frac{N}{2}\right]$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Periodic in m with period N *i.e* $\bar{f}_{-j} = \bar{f}_{N-j}\ ,\ j = 1, 2, \ldots$

$$|\bar{f}\rangle = \left(\bar{f}_{-N/2} \cdots\ , \bar{f}_{-1}, \bar{f}_0, \bar{f}_1\ \cdots\ \bar{f}_{N/2}\right) \quad \left[-\frac{N}{2}, +\frac{N}{2}\right]$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Periodic in m with period N *i.e* $\bar{f}_{-j} = \bar{f}_{N-j}\ ,\ j = 1, 2, \ldots$

$$|\bar{f}\rangle = \left(\bar{f}_0, f_2\ \ldots\ ,\ \ldots\ ,\ \ldots\ ;\ldots\ \bar{f}_{N-1}\right) \quad [1, N)$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Periodic in m with period N *i.e* $\bar{f}_{-j} = \bar{f}_{N-j}\ ,\ j = 1, 2, \ldots$

$$|\bar{f}\rangle = \left(\bar{f}_0, f_2 \ \ldots\ , \ \ldots\ , \ \ldots\ ; \ldots\ \bar{f}_{N-1}\right) \quad [1, N)$$

- Important Properties of DFT:
  - 1. Convolution Property

$$\left\{\overline{f^1 \otimes f^2}\right\}_l = \bar{f}_l^1 \ \bar{f}_l^2$$

where $\left\{f^1 \otimes f^2\right\}_l \equiv \sum_{j=1}^{N} f_{l-j}^1 f_j^2$ extended by periodic summation

  - 2. Parseval's theorem:

$$N \sum_i |f_i|^2 = \sum_j |\bar{f}_j|^2$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Inverse DFT

$$f_m = \frac{1}{N} \sum_{n=0}^{N-1} \bar{f}_n \, e^{2\pi i m n/N}$$

$$|f\rangle = (f_1, f_2, f_3 \cdots f_N) \;,\; f_n = f(n\Delta)$$

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \cdots \bar{f}_N) \;,$$

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Inverse DFT

$$f_m = \frac{1}{N} \sum_{n=0}^{N-1} \bar{f}_n \, e^{2\pi i m n/N}$$

$$|f\rangle = (f_1, f_2, f_3 \cdots f_N) \;,\; f_n = f(n\Delta)$$

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \cdots \bar{f}_N) \;,$$

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

There is a critical frequency associated with the sampling rate

$f_c = \frac{1}{2\Delta}$ Nyquist frequency (Nyquist rate)

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$|f\rangle = (f_1, f_2, f_3 \ldots f_N) \;,\; f_n = f(n\Delta)$$

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \ldots \bar{f}_N) \;,$$

There is a critical frequency associated with the sampling rate

$f_c = \frac{1}{2\Delta}$ Nyquist frequency (Nyquist rate)

$$f(t) = \sin(2\pi f_c t)$$

Discretize and build the vector

$$f_n = \sin\left(2\pi f_c \times n\Delta\right)$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$|f\rangle = (f_1, f_2, f_3 \ldots f_N) \;,\; f_n = f(n\Delta)$$

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \ldots \bar{f}_N) \;,$$

There is a critical frequency associated with the sampling rate

$f_c = \frac{1}{2\Delta}$ Nyquist frequency (Nyquist rate)

$$f(t) = \sin(2\pi f_c t)$$

Discretize and build the vector

$$f_n = \sin\left(2\pi f_c \times n\Delta\right) = \sin n\pi = 0$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

$$|f\rangle = (f_1, f_2, f_3 \cdots f_N) \;,\; f_n = f(n\Delta)$$

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \cdots \bar{f}_N) \;,$$

There is a critical frequency associated with the sampling rate

$f_c = \frac{1}{2\Delta}$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***

$e^{2\pi i f_1 t}$ $e^{2\pi i f_2 t}$ The total signal is being sampled at intervals Δ

3. The Discreet Fourier Transform (DFT)

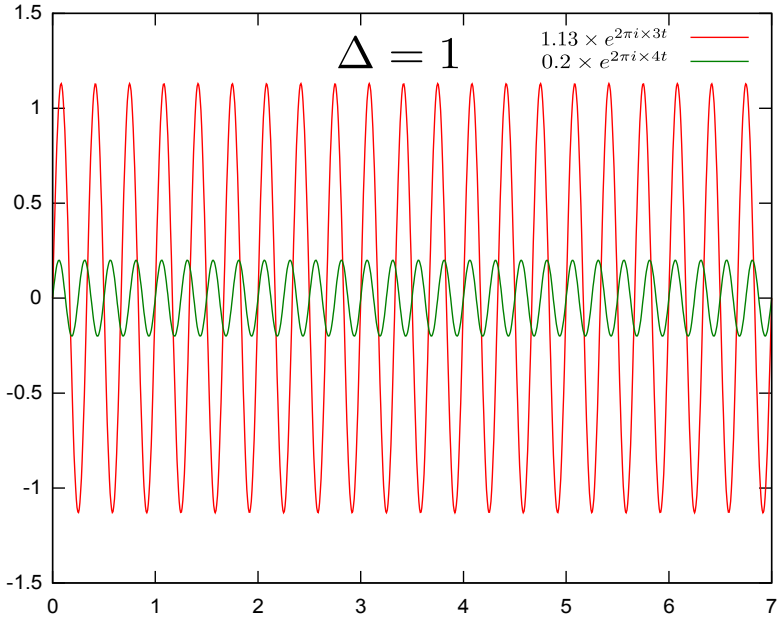
$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***



The total signal is being sampled at intervals Δ

3. The Discreet Fourier Transform (DFT)

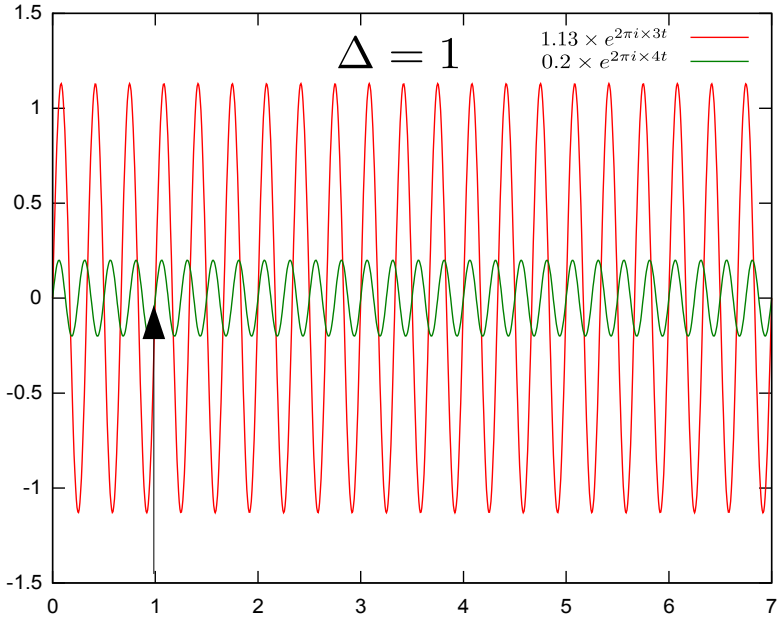
$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***



The total signal is being sampled at intervals Δ

3. The Discreet Fourier Transform (DFT)

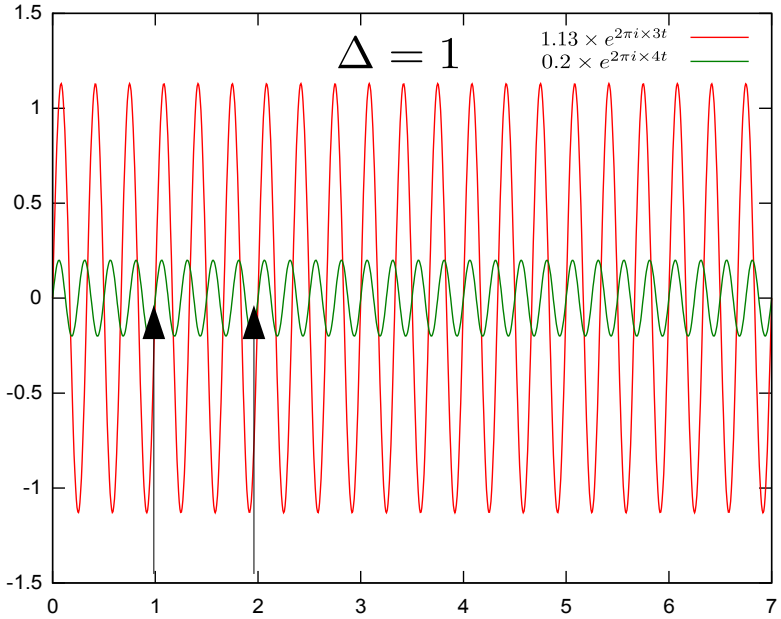
$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***



The total signal is being sampled at intervals Δ

3. The Discreet Fourier Transform (DFT)

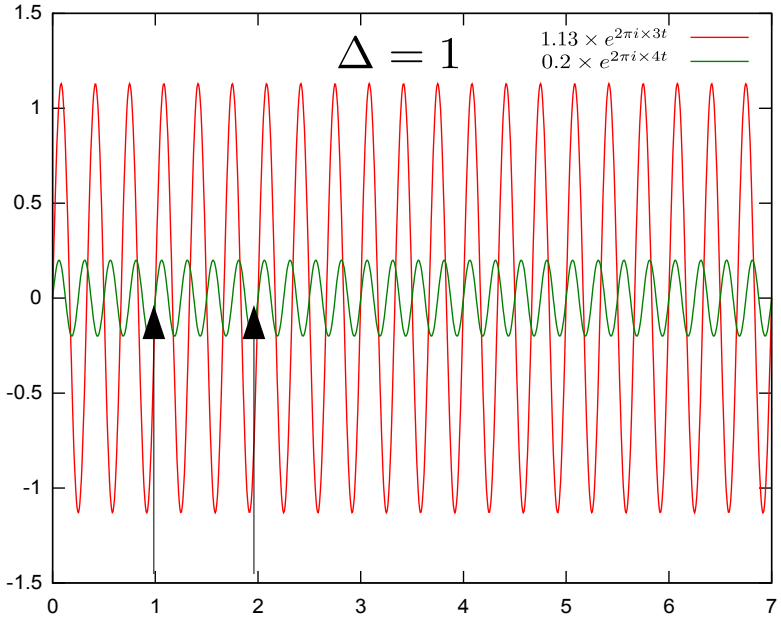
$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***



The total signal is being sampled at intervals Δ

If the two frequencies differ by a multiple of the sampling rate, Then they give the same samples!

Sampling rate is just length of the interval $(-f_c, f_c)$

Any frequency outside this interval gets aliased into it

3. The Discreet Fourier Transform (DFT)

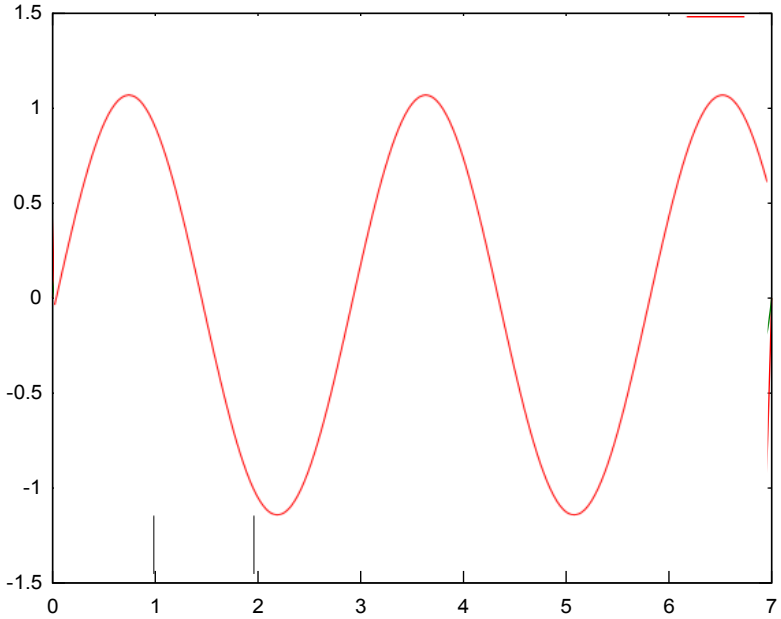
$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

There is a critical frequency associated with the sampling rate

Δ^{-1} = Sampling Rate
Δ = Sampling Interval

$$f_c = \frac{1}{2\Delta}$$ Nyquist frequency (Nyquist rate)

- Any frequency in your data that is at Nyquist rate does not get detected !!
- Worse, any frequency that is more than the Nyquist rate **gives you a false frequency,** a phenomenon called ***aliasing***



The total signal is being sampled at intervals Δ

If the two frequencies differ by a multiple of the sampling rate, Then they give the same samples!

Sampling rate is just length of the interval $(-f_c, f_c)$

Any frequency outside this interval gets aliased into it

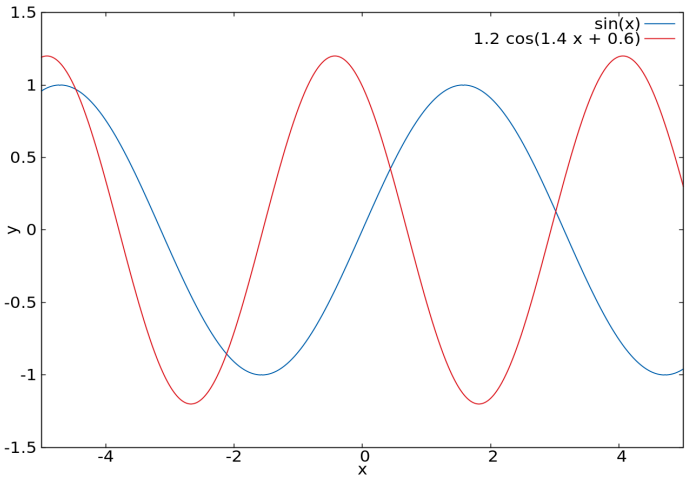
How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency

3. The Discreet Fourier Transform (DFT)

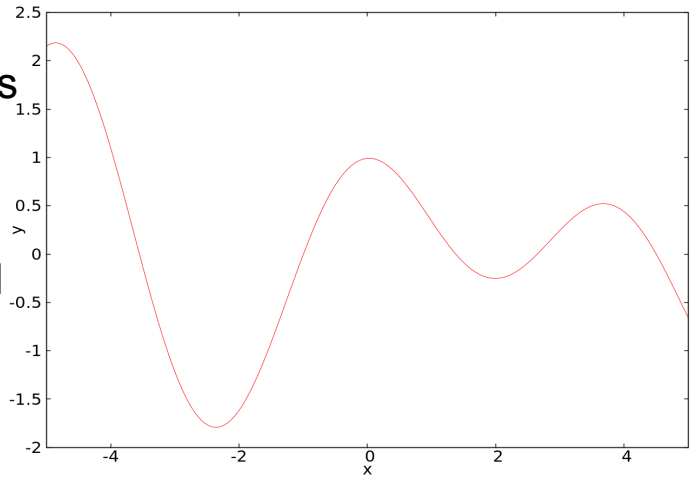
How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency



Add the components

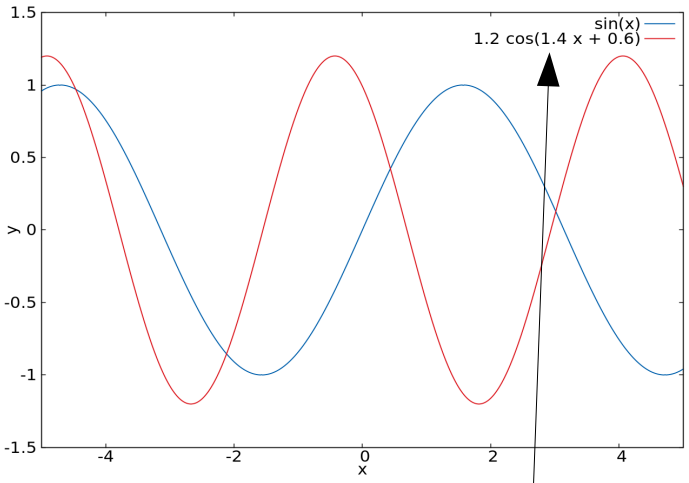
Resolve the signal



3. The Discreet Fourier Transform (DFT)

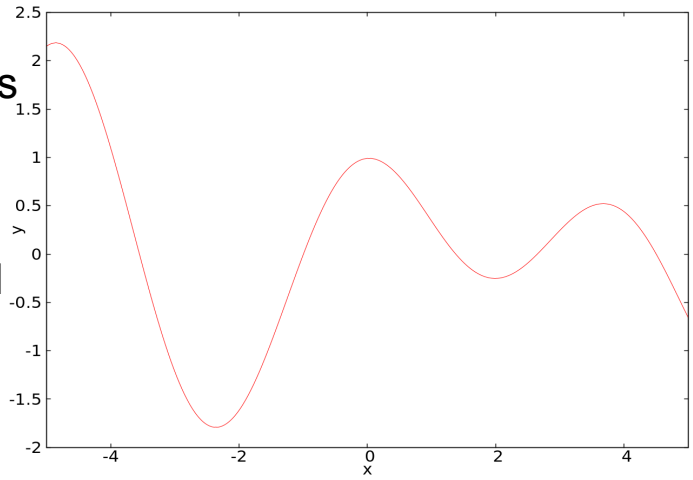
How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency



Add the components

Resolve the signal



$$\frac{1}{2\Delta} = f_c \gg \frac{1.4}{2\pi}$$

$$\Delta \ll \frac{\pi}{1.4} \approx 2.2$$

How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency

- Choose a Nyquist rate, and sample the data accordingly
  If the FFT is too large at the Nyquist edges, then you're aliasing high frequencies
  Increase the Nyquist rate and re-sample
  Keep doing this until the FFT at the Nyquist edges is small enough to ignore.

3. The Discreet Fourier Transform (DFT)

How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency

- Choose a Nyquist rate, and sample the data accordingly
  If the FFT is too large at the Nyquist edges, then you're aliasing high frequencies
  Increase the Nyquist rate and re-sample
  Keep doing this until the FFT at the Nyquist edges is small enough to ignore.

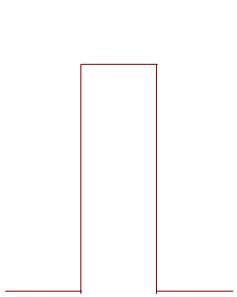
Infinitely extended signals with (quasi) periodic behavior are not expected to have too many frequencies (here, only 2)

3. The Discreet Fourier Transform (DFT)

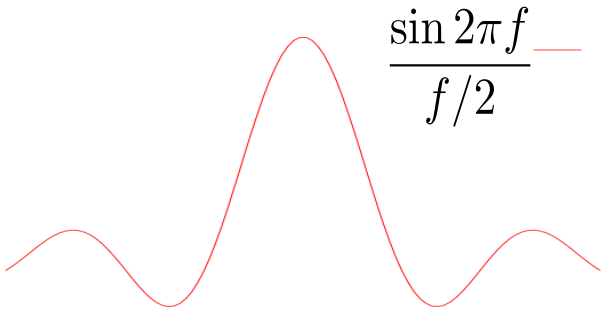
How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency

- Choose a Nyquist rate, and sample the data accordingly
  If the FFT is too large at the Nyquist edges, then you're aliasing high frequencies
  Increase the Nyquist rate and re-sample
  Keep doing this until the FFT at the Nyquist edges is small enough to ignore.



Brightness Image



Fourier Transform

Infinitely extended signals with (quasi) periodic behavior are not expected to have too many frequencies (here, only 2)

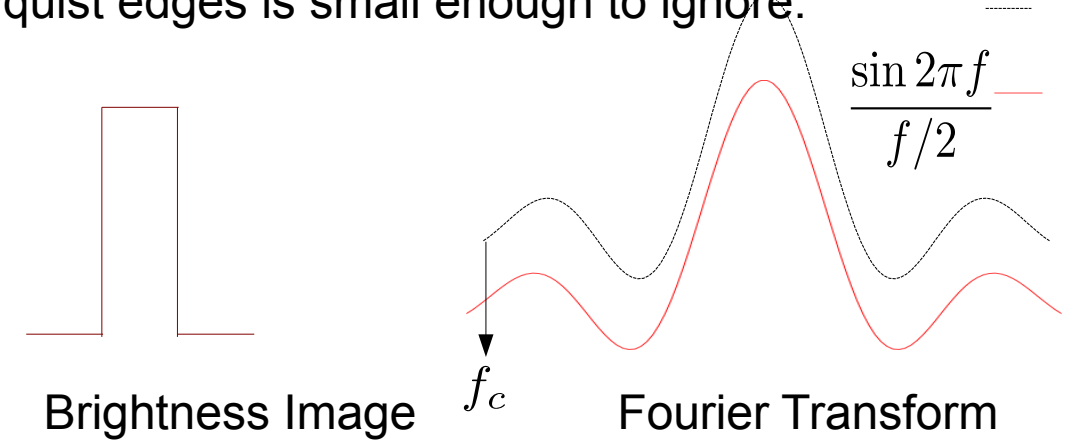
Finite signals, will have many many frequencies (here, a full continuum) - larger frequencies have smaller amplitudes

3. The Discreet Fourier Transform (DFT)

How to handle aliasing?

- Know the natural bandwidth of the signal
  If you already know something about what frequencies are there
  Make sure that your Nyquist rate is faster than the largest expected frequency

- Choose a Nyquist rate, and sample the data accordingly
  If the FFT is too large at the Nyquist edges, then you're aliasing high frequencies
  Increase the Nyquist rate and re-sample
  Keep doing this until the FFT at the Nyquist edges is small enough to ignore.



Infinitely extended signals with (quasi) periodic behavior are not expected to have too many frequencies (here, only 2)

Finite signals, will have many many frequencies (here, a full continuum) - larger frequencies have smaller amplitudes

Choose a sample rate. Sample. Take FFT and re - sample until aliasing falls off

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{aligned}
\bar{f}_0 &= e^{-2\pi i\, 0\times 0/N} f_0 + e^{-2\pi i\, 0\times 1/N} f_1 + e^{-2\pi i\, 0\times 2/N} f_2 + \ldots e^{-2\pi i\, 0\times (N-1)/N} f_{N-1} \\
\bar{f}_1 &= e^{-2\pi i\, 1\times 0/N} f_0 + e^{-2\pi i\, 1\times 1/N} f_1 + e^{-2\pi i\, 1\times 2/N} f_2 + \ldots e^{-2\pi i\, 1\times (N-1)/N} f_{N-1} \\
\bar{f}_2 &= e^{-2\pi i\, 2\times 0/N} f_0 + e^{-2\pi i\, 2\times 1/N} f_1 + e^{-2\pi i\, 2\times 2/N} f_2 + \ldots e^{-2\pi i\, 2\times (N-1)/N} f_{N-1} \\
\vdots &\quad \vdots \qquad\qquad\qquad\qquad \ddots \qquad\qquad\qquad\qquad \vdots
\end{aligned}$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{N-1} \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \dots & e^{-2\pi i \, 0\times (N-1)/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \dots & e^{-2\pi i \, 1\times (N-1)/N} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-2\pi i \, (N-1)\times 0/N} & e^{-2\pi i \, (N-1)\times 1/N} & \dots & e^{-2\pi i \, (N-1)\times (N-1)/N} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

$$U$$

$$|\bar{f}\rangle = U \cdot |f\rangle \qquad U_{mn} = \exp\left(-2\pi i m n/N\right)$$

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{N-1} \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \dots & e^{-2\pi i \, 0\times (N-1)/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \dots & e^{-2\pi i \, 1\times (N-1)/N} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-2\pi i \, (N-1)\times 0/N} & e^{-2\pi i \, (N-1)\times 1/N} & \dots & e^{-2\pi i \, (N-1)\times (N-1)/N} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

$$U$$

$$|\bar{f}\rangle = U \cdot |f\rangle \qquad U_{mn} = \exp\left(-2\pi i m n/N\right)$$

- Thus, DFT is basically just a Linear Algebra problem!

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{N-1} \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \dots & e^{-2\pi i \, 0\times (N-1)/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \dots & e^{-2\pi i \, 1\times (N-1)/N} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-2\pi i \, (N-1)\times 0/N} & e^{-2\pi i \, (N-1)\times 1/N} & \dots & e^{-2\pi i \, (N-1)\times (N-1)/N} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

$$U$$

$$|\bar{f}\rangle = U \cdot |f\rangle \qquad U_{mn} = \exp\left(-2\pi i m n/N\right)$$

- Thus, DFT is basically just a Linear Algebra problem!
- The numpy and scipy modules already have routines for matrices.

3. The Discreet Fourier Transform (DFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{N-1} \end{pmatrix} = \begin{pmatrix} e^{-2\pi i\, 0\times 0/N} & e^{-2\pi i\, 0\times 1/N} & \dots & e^{-2\pi i\, 0\times (N-1)/N} \\ e^{-2\pi i\, 1\times 0/N} & e^{-2\pi i\, 1\times 1/N} & \dots & e^{-2\pi i\, 1\times (N-1)/N} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-2\pi i\, (N-1)\times 0/N} & e^{-2\pi i\, (N-1)\times 1/N} & \dots & e^{-2\pi i\, (N-1)\times (N-1)/N} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

$$U$$

$$|\bar{f}\rangle = U \cdot |f\rangle \qquad U_{mn} = \exp\left(-2\pi i m n/N\right)$$

- Thus, DFT is basically just a Linear Algebra problem!
- The numpy and scipy modules already have routines for matrices.
- So, are we done? Not quite.

4. The Fast Fourier Transform (FFT)

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

$$|f\rangle = (f_1, f_2, f_3 \; \cdots \; f_N) \;\; , \;\; f_n = f(n\Delta)$$

$\Delta^{-1} =$ Sampling Rate

DFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \; \cdots \; \bar{f}_N) \;\; , \;\; \bar{f}_n = \sum_{n=1}^{N} f_n \; e^{-2\pi i m n/N} \quad ???$$

$$|\bar{f}\rangle = U \cdot |f\rangle$$

Is there a faster way than to do this brute force matrix-vector operation?

4. The Fast Fourier Transform (FFT)

$$\bar{f}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \; f(t) \; e^{-i\omega t}$$

$$|f\rangle = (f_1, f_2, f_3 \; \cdots \; f_N) \;\; , \;\; f_n = f(n\Delta)$$

$\Delta^{-1} =$ Sampling Rate

FFT

$$|\bar{f}\rangle = (\bar{f}_1, \bar{f}_2, \bar{f}_3 \; \cdots \; \bar{f}_N) \;\; , \;\; \bar{f}_n = \sum_{n=1}^{N} f_n \; e^{-2\pi i m n/N} \;\; ???$$

$$|\bar{f}\rangle = U \cdot |f\rangle$$

Is there a faster way than to do this brute force matrix-vector operation?

Yes, there is! It's called the FFT algorithm.

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \cdots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \cdots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \cdots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

As a special case, consider N=4

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} =$$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \cdots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \cdots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \cdots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

As a special case, consider N=4

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} f_0 + f_1 + f_2 + f_3 \\ f_0 - if_1 - f_2 + if_3 \\ f_0 - f_1 + f_2 - f_3 \\ f_0 + if_1 - f_2 - if_3 \end{pmatrix}$$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \cdots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \cdots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \cdots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

As a special case, consider N=4

16 multiplications and 12 additions!

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} f_0 + f_1 + f_2 + f_3 \\ f_0 - if_1 - f_2 + if_3 \\ f_0 - f_1 + f_2 - f_3 \\ f_0 + if_1 - f_2 - if_3 \end{pmatrix}$$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \cdots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \cdots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \cdots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

As a special case, consider N=4

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} (f_0 + f_2) + (f_1 + f_3) \\ (f_0 - f_2) - i(f_1 - f_3) \\ (f_0 + f_2) - (f_1 + f_3) \\ (f_0 - f_2) + i(f_1 - f_3) \end{pmatrix}$$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \cdots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \cdots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \cdots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

As a special case, consider N=4

2 multiplications and 8 additions!
Much faster this way!!!

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} (f_0 + f_2) + (f_1 + f_3) \\ (f_0 - f_2) - i(f_1 - f_3) \\ (f_0 + f_2) - (f_1 + f_3) \\ (f_0 - f_2) + i(f_1 - f_3) \end{pmatrix}$$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

As a special case, consider N=4

2 multiplications and 8 additions!
Much faster this way!!!

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} (f_0 + f_2) + (f_1 + f_3) \\ (f_0 - f_2) - i(f_1 - f_3) \\ (f_0 + f_2) - (f_1 + f_3) \\ (f_0 - f_2) + i(f_1 - f_3) \end{pmatrix}$$

This observation may reduce the computational effort from $\mathcal{O}(N^2)$ into $\mathcal{O}(N \log_2 N)$

Much, much faster! Because $\lim_{N\to\infty} \frac{\log_2 N}{N} = 0$

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\begin{pmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} e^{-2\pi i \, 0\times 0/N} & e^{-2\pi i \, 0\times 1/N} & \dots & e^{-2\pi i \, 0\times N-1/N} \\ e^{-2\pi i \, 1\times 0/N} & e^{-2\pi i \, 1\times 1/N} & \dots & e^{-2\pi i \, 1\times N-1/N} \\ e^{-2\pi i \, 2\times 0/N} & e^{-2\pi i \, 2\times 1/N} & \dots & e^{-2\pi i \, 2\times N-1/N} \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

- The U matrix above consists only of roots of 1 *i.e.* 1, -1 & $\pm i$
- Thus, all we need to do is group those terms in f that multiply with $1, -1, i, \& -i$ respectively
- Add (or subtract) them as need be, then multiply them with $i, \& -i$ and add/subtract the results

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

- The U matrix above consists only of 1s and roots of -1 i.e. $\pm i$
- Thus, all we need to do is group those terms in f that multiply with $1, -1, i, \& -i$ respectively

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m \, (2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m \, (2j+1)/N}$$

1. Separate the odd and even indices in the sum

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

- The U matrix above consists only of 1s and roots of -1 i.e. $\pm i$
- Thus, all we need to do is group those terms in f that multiply with $1, -1, i, \& -i$ respectively

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m \, (j)/(N/2)} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m \, (j)/(N/2)} \left(e^{-2\pi i/N}\right)^m$$

2. Rearrange terms in the exponent so N changes to N/2 and a root of -1 comes out

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

- The U matrix above consists only of 1s and roots of -1 i.e. $\pm i$
- Thus, all we need to do is group those terms in f that multiply with $1, -1, i, \& -i$ respectively

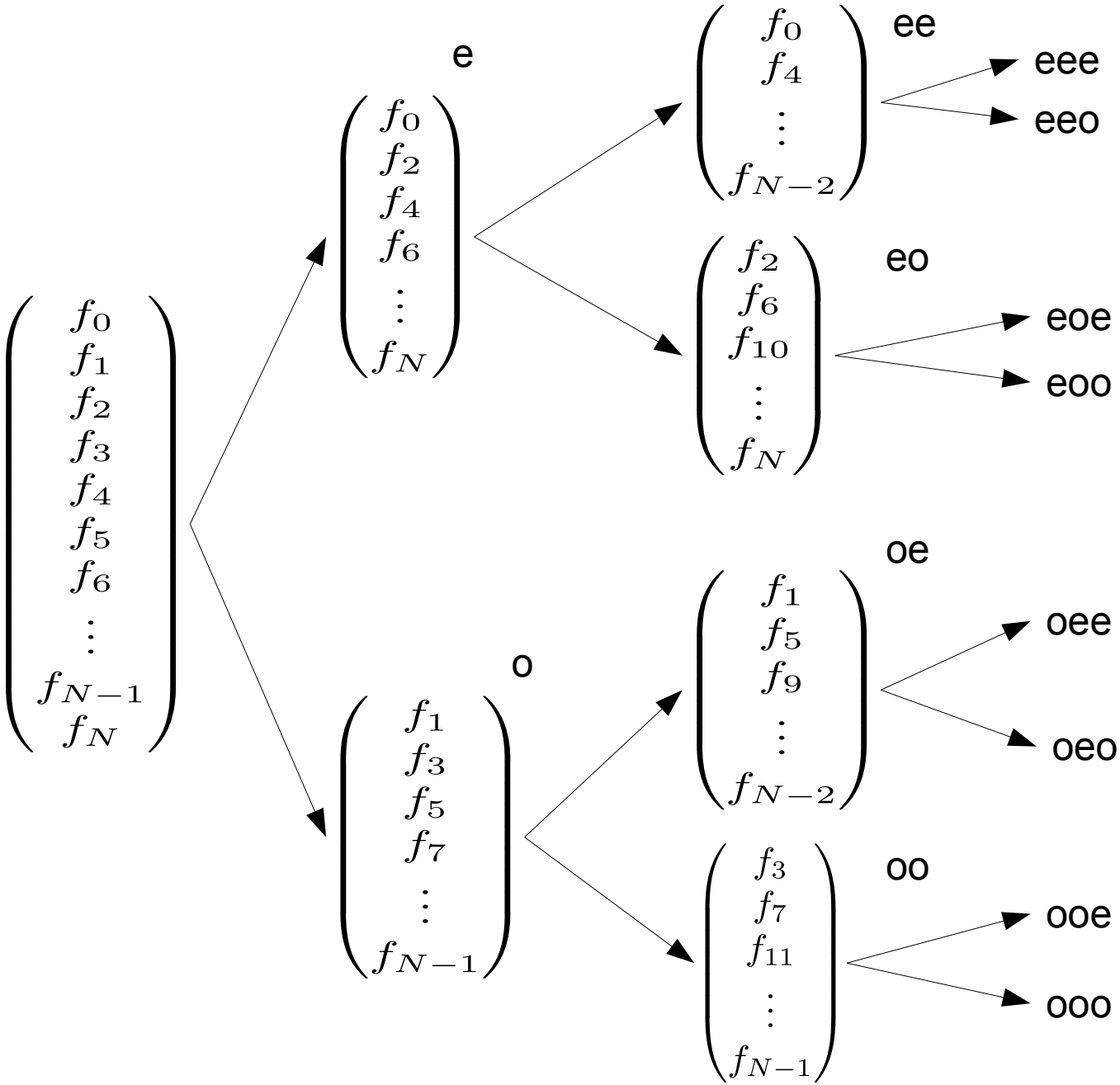
Two smaller DFT's: **Decimation in Time**
Danielson Lanczos lemma (1942!)
Based on earlier works by Gauss (1805!!)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m\,(j)/(N/2)} + \left(e^{-2\pi i/N}\right)^m \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m\,(j)/(N/2)}$$

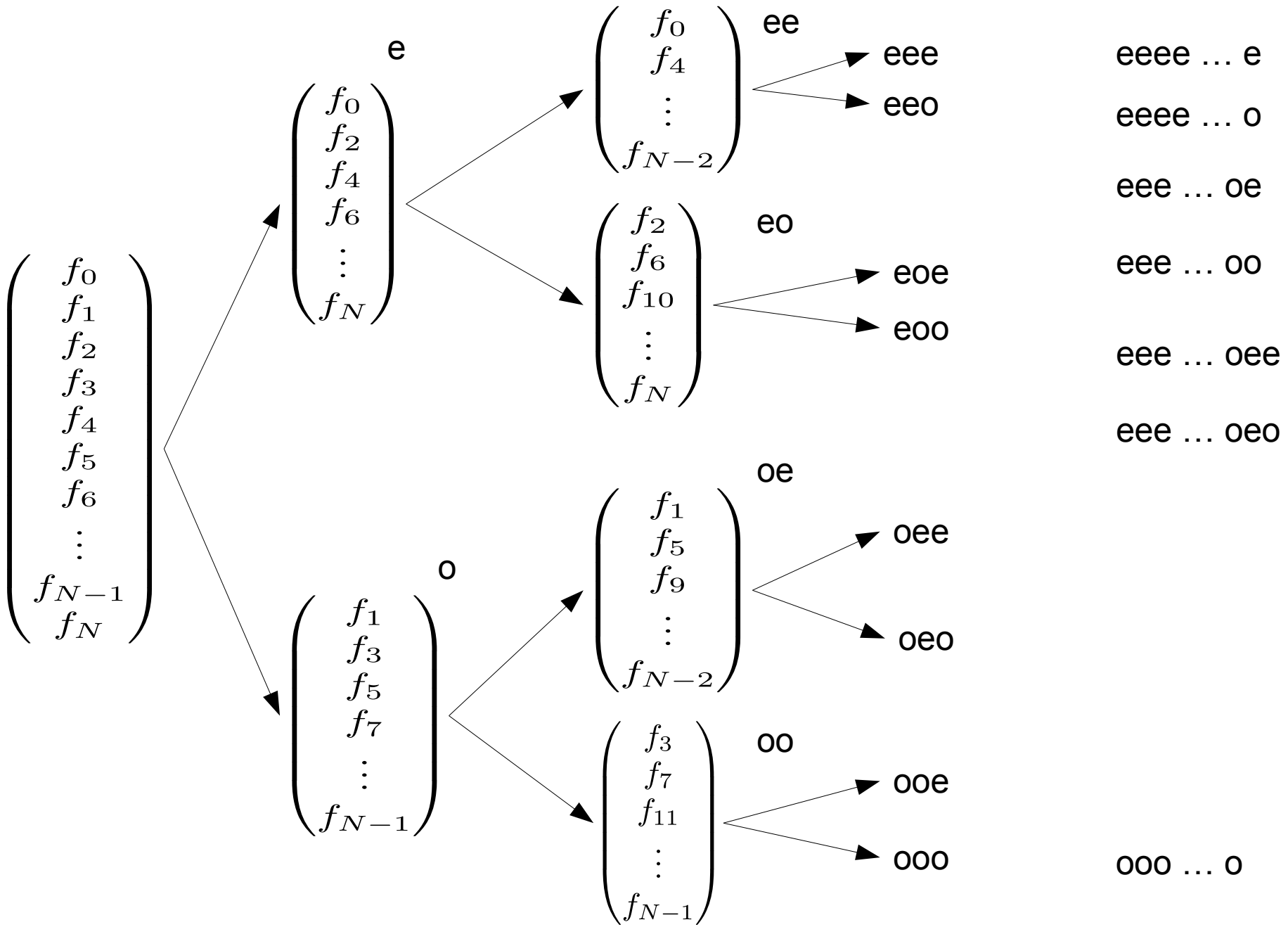
3. Split the FFT to a combination of two smaller FFT's. This'll work so long as N is a power of 2

4. The Fast Fourier Transform (FFT)



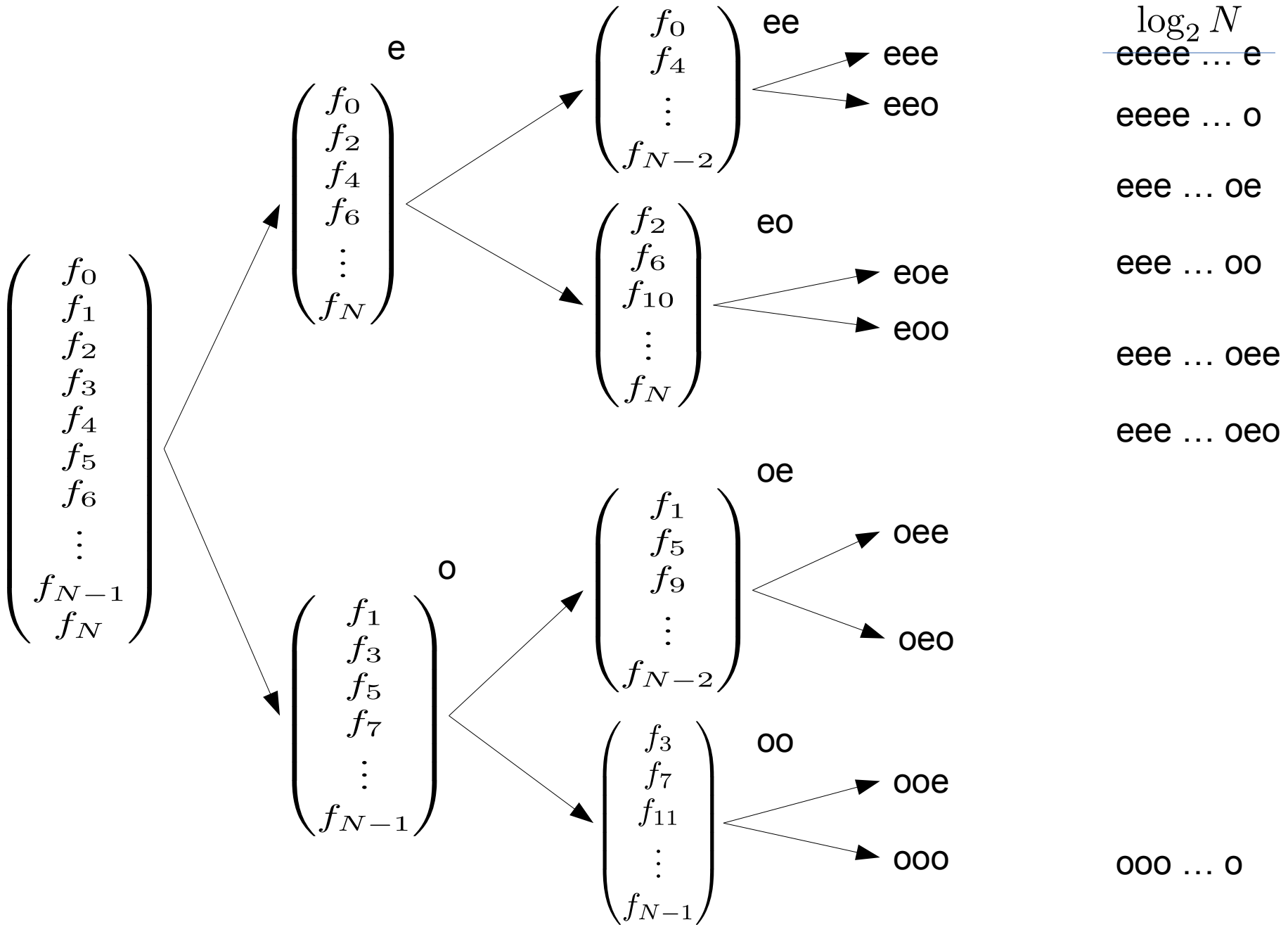
4. Repeat recursively until you wind up with only vectors of length 1by rearranging in bit reversed order of index

4. The Fast Fourier Transform (FFT)



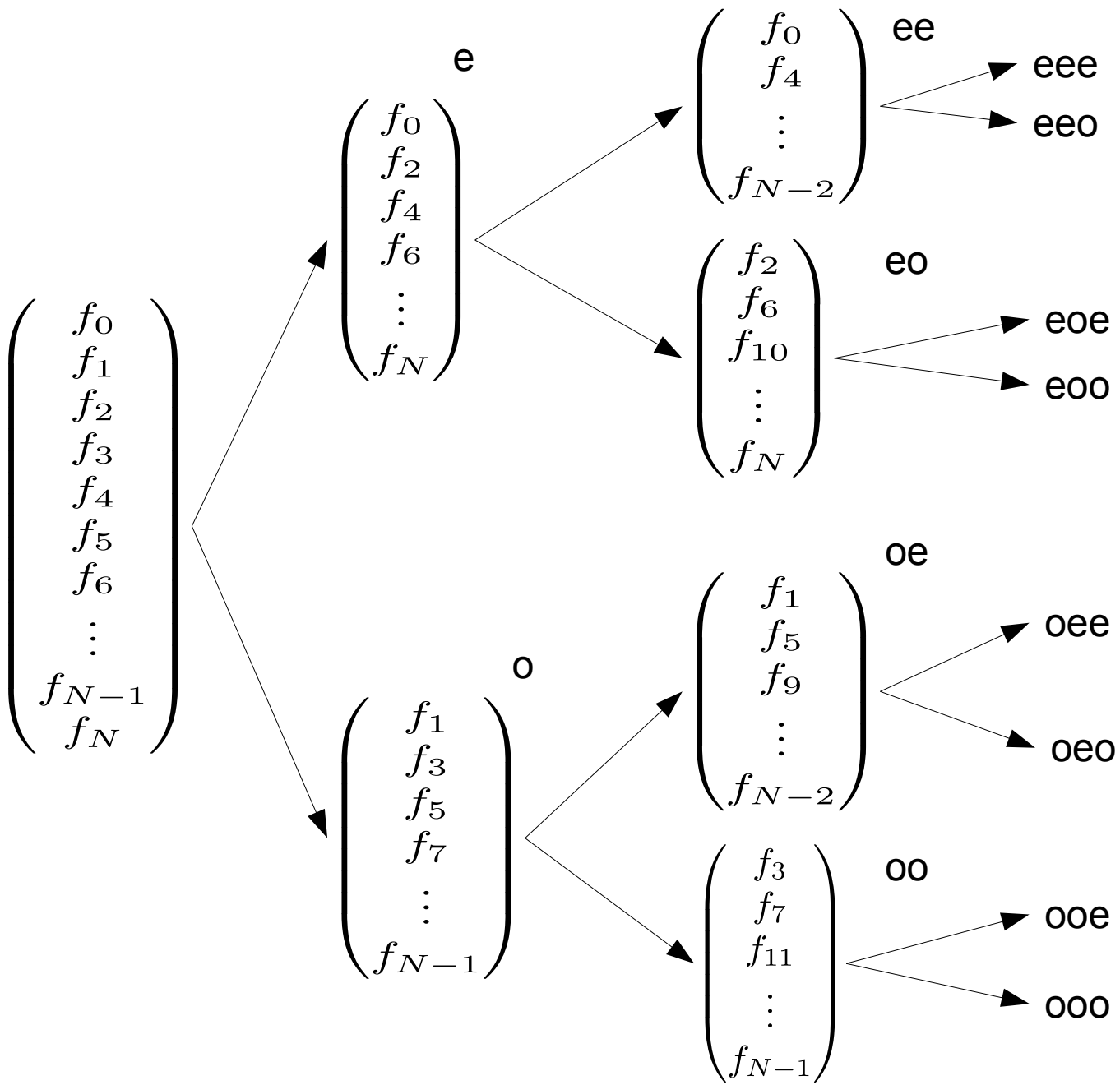
4. Repeat recursively until you wind up with only vectors of length 1by rearranging in bit reversed order of index

4. The Fast Fourier Transform (FFT)



4. Repeat recursively until you wind up with only vectors of length 1by rearranging in bit reversed order of index

4. The Fast Fourier Transform (FFT)



$\log_2 N$

eeee … e

eeee … o

eee … oe

eee … oo

eee … oee

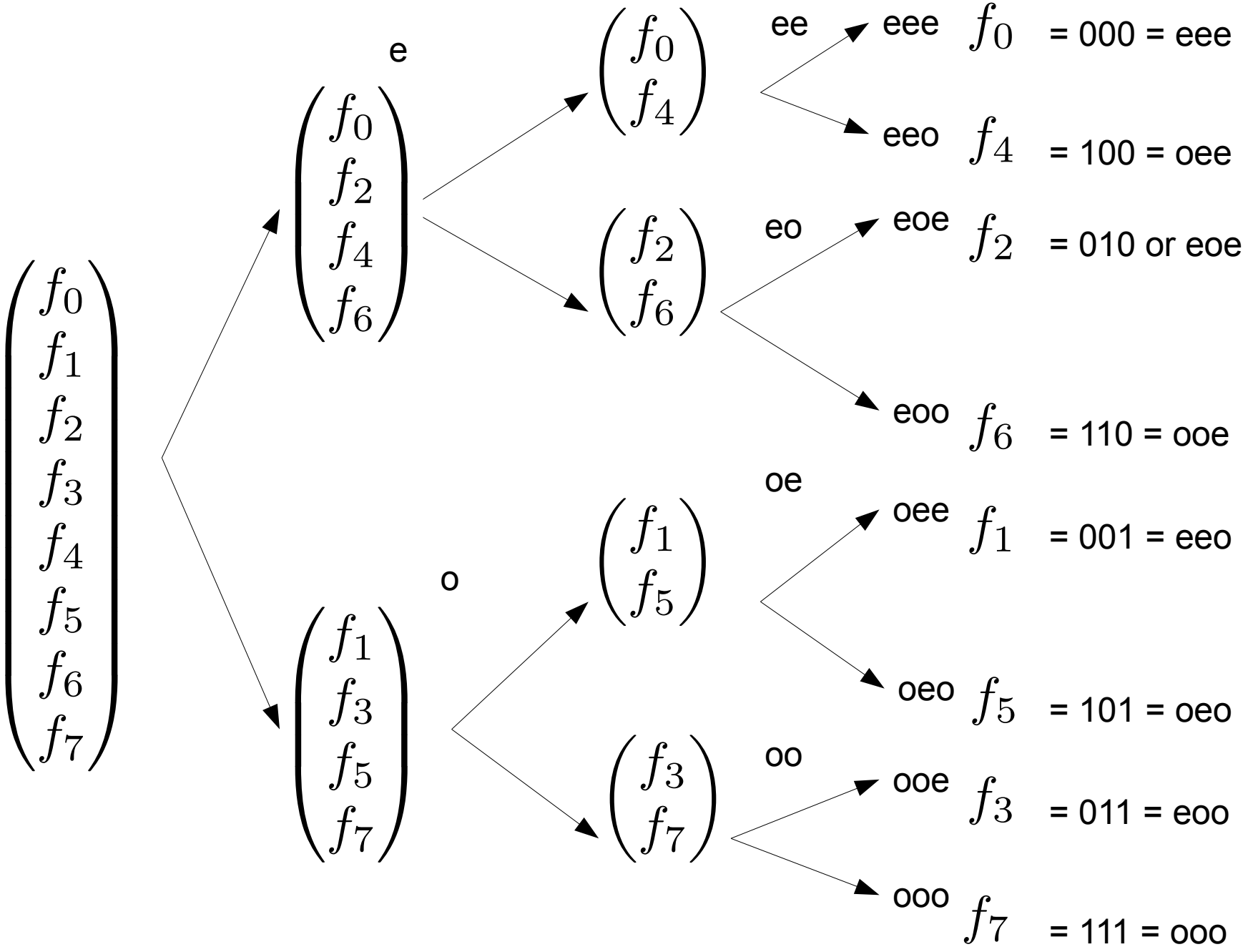
eee … oeo

ooo … o

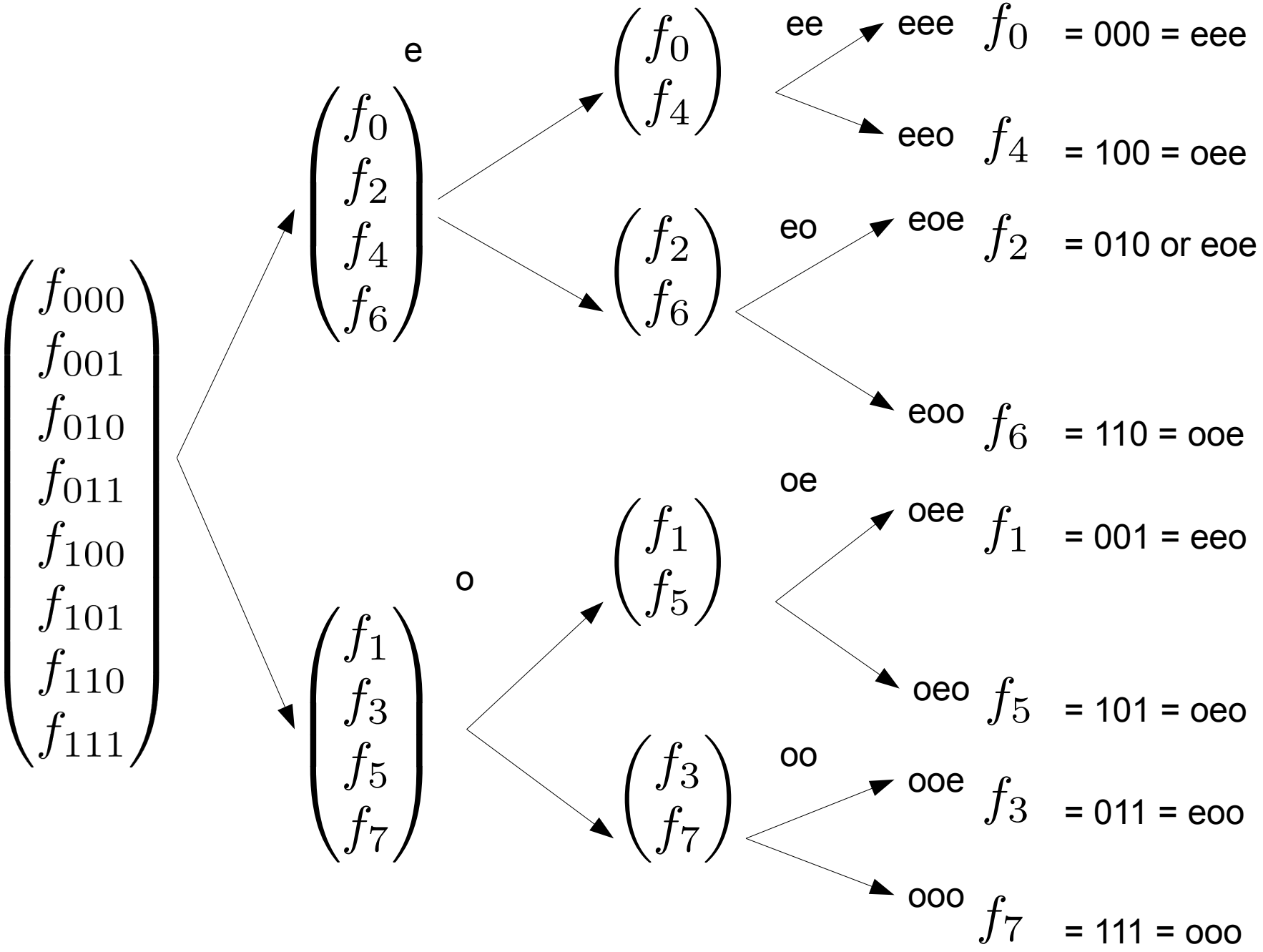
BIT REVERSED ORDER OF THE ORIGINAL INDICES!!! The famous Cooley-Tukey Algorithm

4. Repeat recursively until you wind up with only vectors of length 1by rearranging in bit reversed order of index

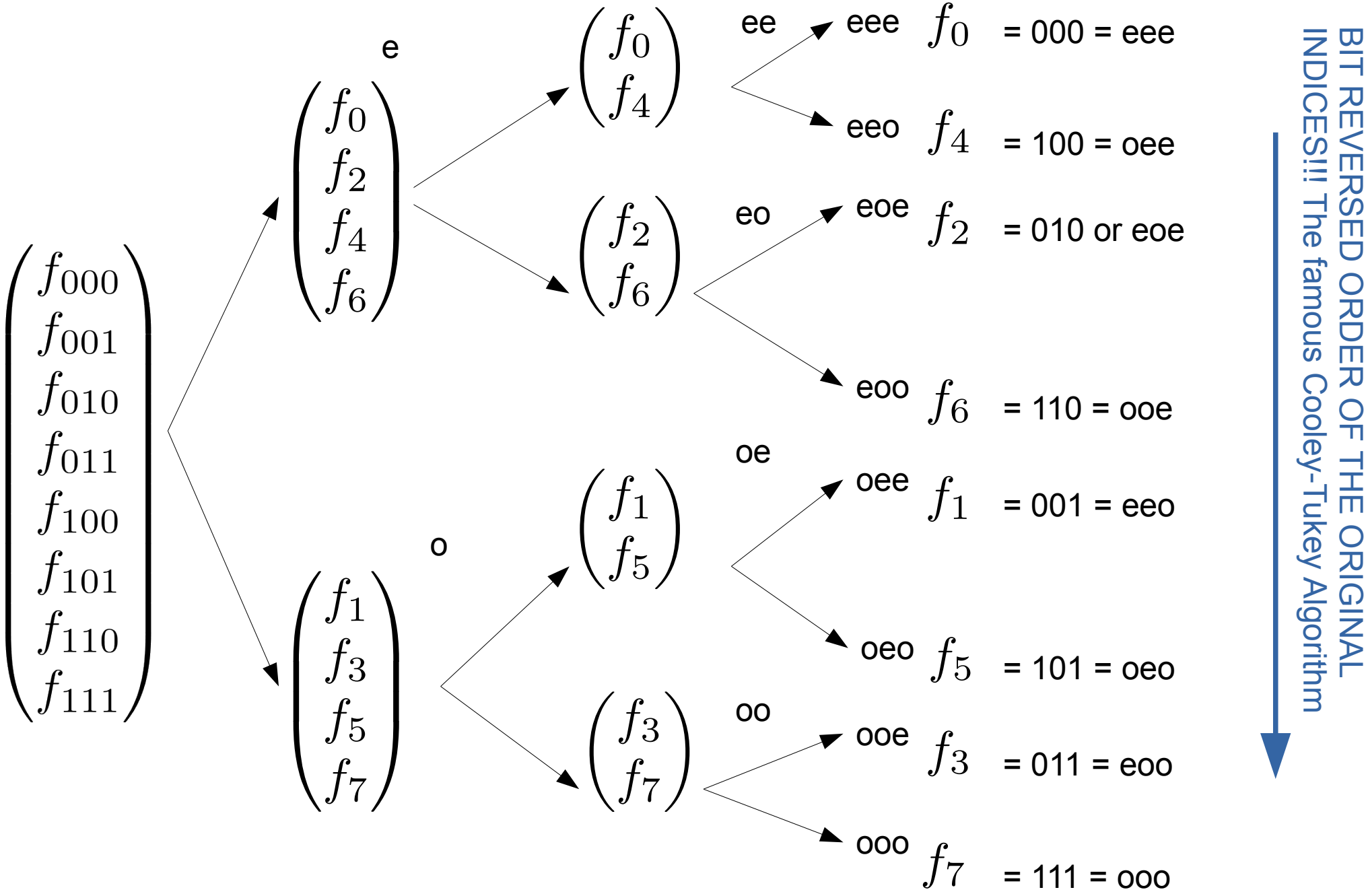
4. The Fast Fourier Transform (FFT)



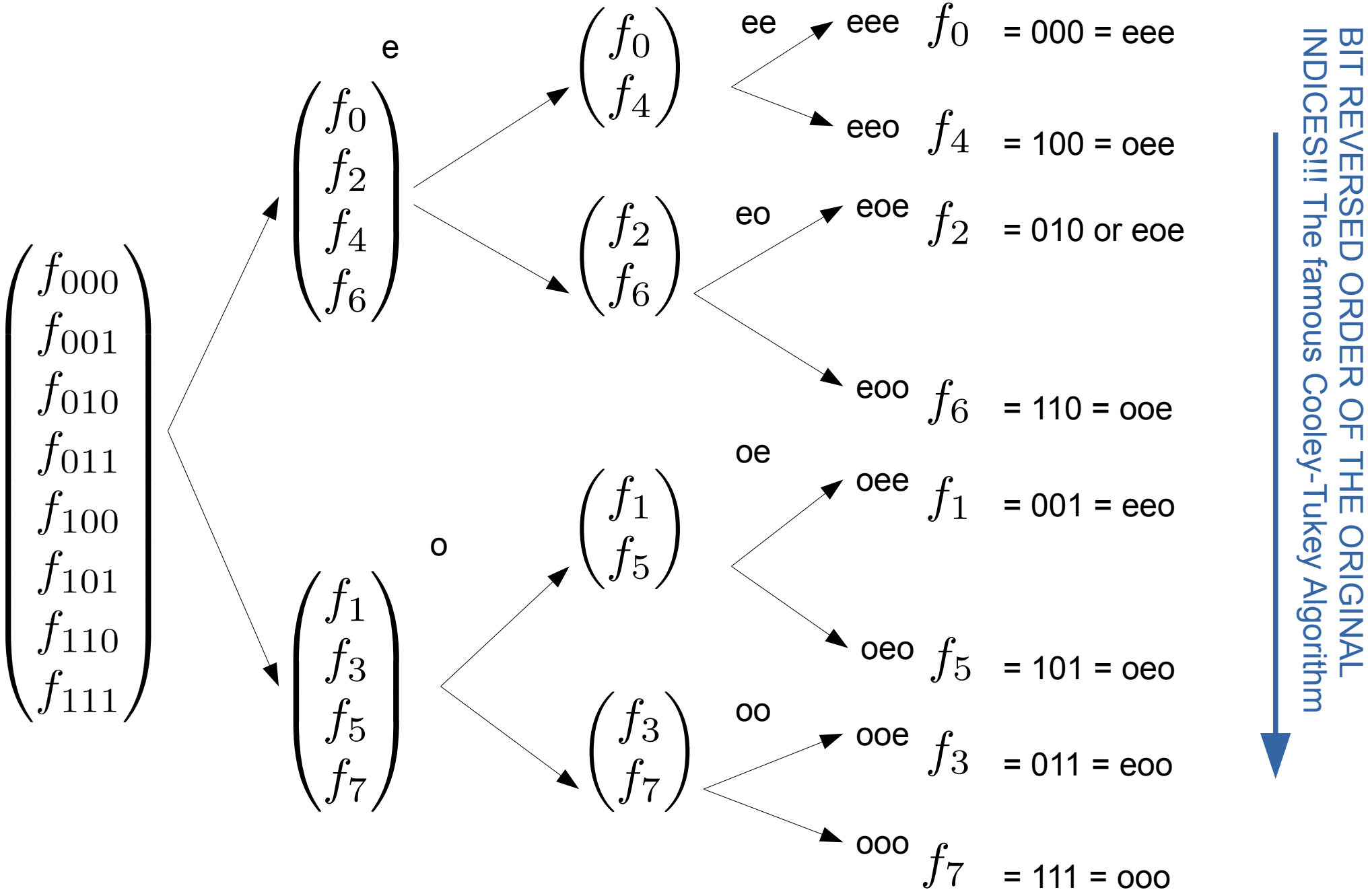
4. The Fast Fourier Transform (FFT)



4. The Fast Fourier Transform (FFT)



4. The Fast Fourier Transform (FFT)



It works because the successive subdivisions into even & odd are tests of successive low-order (least significant) bits of the original index.

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

Danielson-lanczos lemma (recursive)

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m\,(j)/(N/2)} + \left(e^{-2\pi i/N}\right)^m \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m\,(j)/(N/2)}$$

What have we done to obtain FFT analytically?

1. Separate the odd and even indices in the sum
2. Rearrange terms in the exponent so N changes to N/2 and a root of -1 comes out
3. Split the FFT to a combo of two smaller FFT's (Danielson – Lanczos lemma)
4. Repeat recursively until you wind up with only vectors of length 1 by rearranging in bit reversed order of index

How should FFT be done numerically ? The Cooley-Tukey Algorithm

1. Rearrange the vector in bit reversed order of index
2. Apply the Danielson - Lanzcos lemma to get the previous combination
3. Repeat this $\log_2 N$ times until full FFT vector is obtained. Only $N \log_2 N$ multiplications!

4. The Fast Fourier Transform (FFT)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

Danielson-lanczos lemma (recursive)

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m \, (j)/(N/2)} + \left(e^{-2\pi i/N}\right)^m \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m \, (j)/(N/2)}$$

FFT Algorithm in pseudocode (without the bit reversal):

Radix-2 decimation in time (DIT) form of Cooley-Tukey algorithm (1965)
Replace the input with the output (in-place)

```
algorithm iterative-fft is
    input: Array a of n complex values where n is a power of 2.
    output: Array A the DFT of a.

    bit-reverse-copy(a, A)
    n ← a.length
    for s = 1 to log(n) do
        m ← 2s
        ωm ← exp(−2πi/m)
        for k = 0 to n-1 by m do
            ω ← 1
            for j = 0 to m/2 − 1 do
                t ← ω A[k + j + m/2]
                u ← A[k + j]
                A[k + j] ← u + t
                A[k + j + m/2] ← u − t
                ω ← ω ωm

    return A
```

Cooley-Tukey Algorithms

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m \, (2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m \, (2j+1)/N}$$

1. Separate the odd and even indices in the sum (Decimation in Time)

4. The Fast Fourier Transform (FFT)

Cooley-Tukey Algorithms

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m\,(2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m\,(2j+1)/N}$$

1. Separate the odd and even indices in the sum (Decimation in Time)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2} \, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

2. Separate the first and second half (Decimation in Frequency)

4. The Fast Fourier Transform (FFT)

Cooley-Tukey Algorithms

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m\,(2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m\,(2j+1)/N}$$

1. Separate the odd and even indices in the sum (Decimation in Time)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

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

$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2} \, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

2. Separate the first and second half (Decimation in Frequency)

4. The Fast Fourier Transform (FFT)

Cooley-Tukey Algorithms

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} f_{2j} \, e^{-2\pi i m\,(2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} \, e^{-2\pi i m\,(2j+1)/N}$$

1. Separate the odd and even indices in the sum (Decimation in Time)

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2} \, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

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

$$\begin{cases} 1 & m \text{ is even} \\ -1 & m \text{ is odd} \end{cases}$$

2. Separate the first and second half (Decimation in Frequency)

4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

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

$$\Omega_N^{(N/2)m} = \begin{cases} 1 & m \text{ is even} \\ -1 & m \text{ is odd} \end{cases}$$

$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2} \, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

Now, split the $\bar{f}_m$ into odd-and-even (so, “Decimation in Frequency”)

4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

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

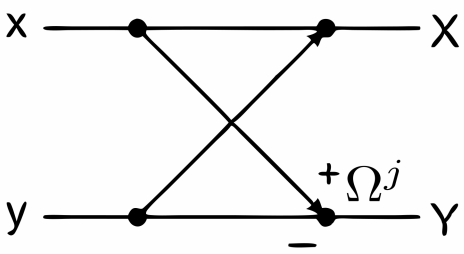
$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2}\, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

$$\Omega_N^{(N/2)m} = \begin{cases} 1 & m \text{ is even} \\ -1 & m \text{ is odd} \end{cases}$$

Now, split the $\bar{f}_m$ into odd-and-even (so, “Decimation in Frequency”)

$$X \to x + y$$

$$Y \to (x - y)\,\Omega^j$$



4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

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

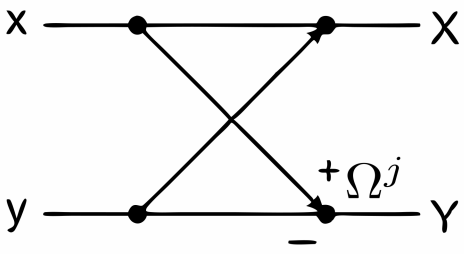
$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2}\, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

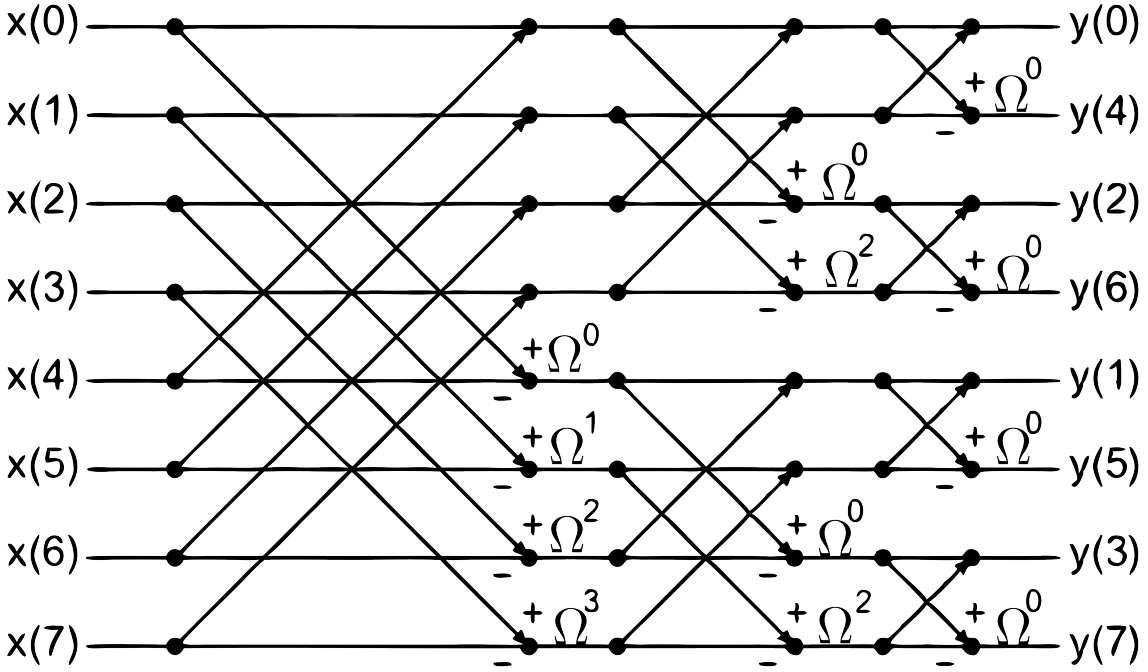
$$\Omega_N^{(N/2)m} = \begin{cases} 1 & m \text{ is even} \\ -1 & m \text{ is odd} \end{cases}$$

Now, split the $\bar{f}_m$ into odd-and-even (so, “Decimation in Frequency”)

$$X \to x + y$$

$$Y \to (x - y)\,\Omega^j$$





4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n/N}$$

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

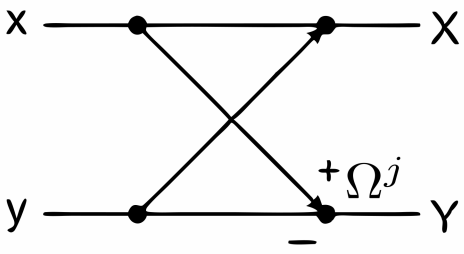
$$\sum_{j=0}^{N/2-1} \left[f_j \, e^{-2\pi i m\,(2j)/N} + f_{j+N/2}\, \Omega_N^{(N/2)m} \right] \Omega_N^{jm}$$

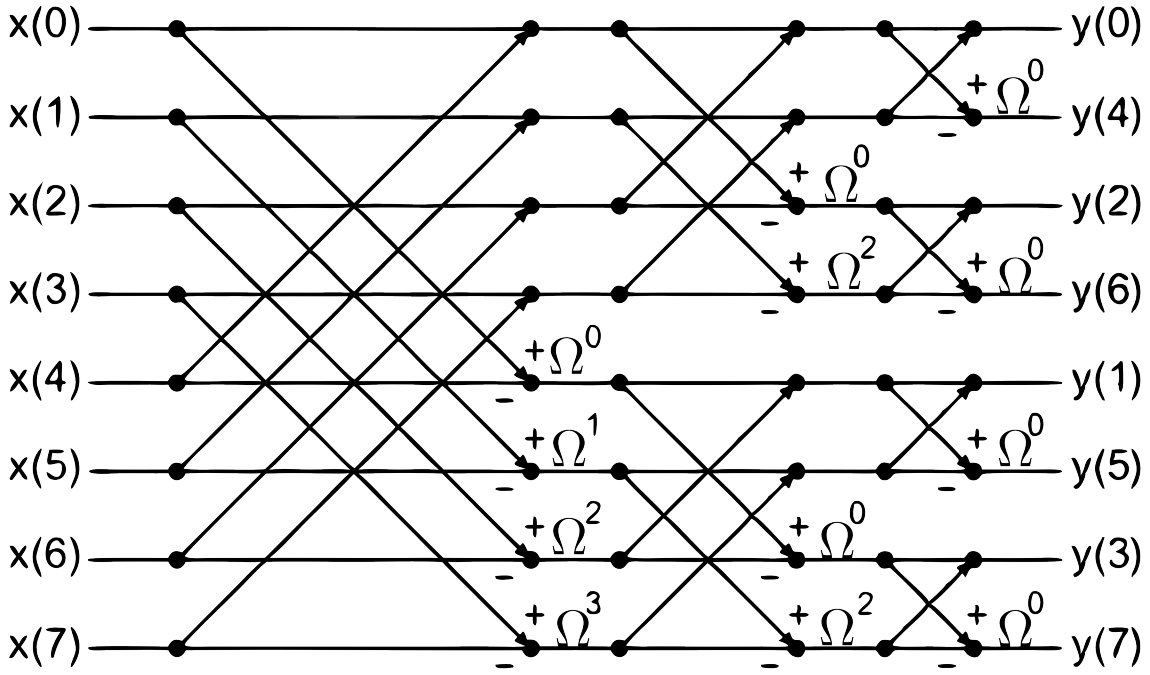
$$\Omega_N^{(N/2)m} = \begin{cases} 1 & m \text{ is even} \\ -1 & m \text{ is odd} \end{cases}$$

Now, split the $\bar{f}_m$ into odd-and-even (so, “Decimation in Frequency”)

$$X \to x + y$$

$$Y \to (x - y)\,\Omega^j$$



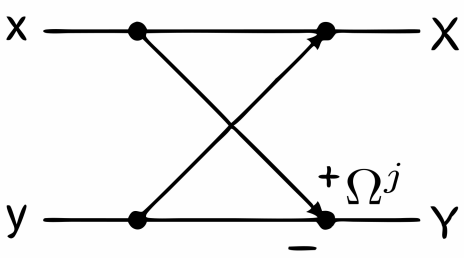


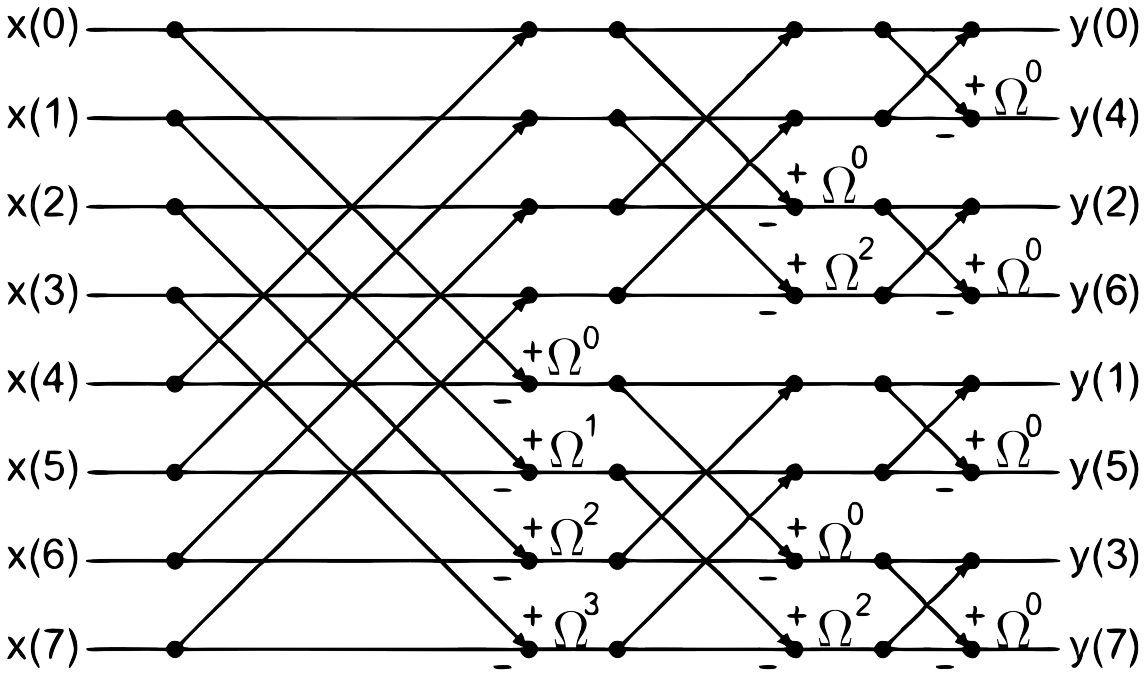
4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$X \rightarrow x + y$$

$$Y \rightarrow (x - y)\,\Omega^{j}$$





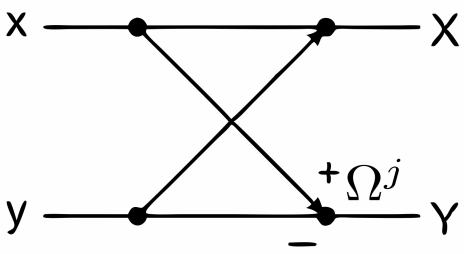
also bit-reversed

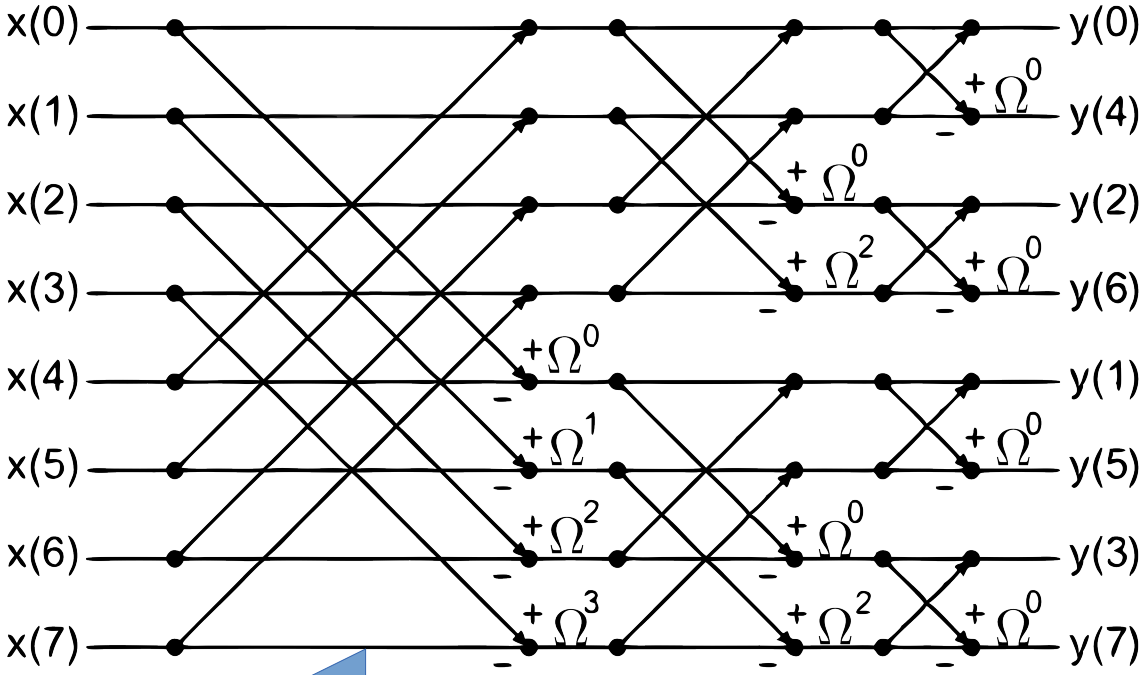
4. The Fast Fourier Transform (FFT)

Decimation in Frequency

$$X \rightarrow x + y$$

$$Y \rightarrow (x - y)\,\Omega^{j}$$

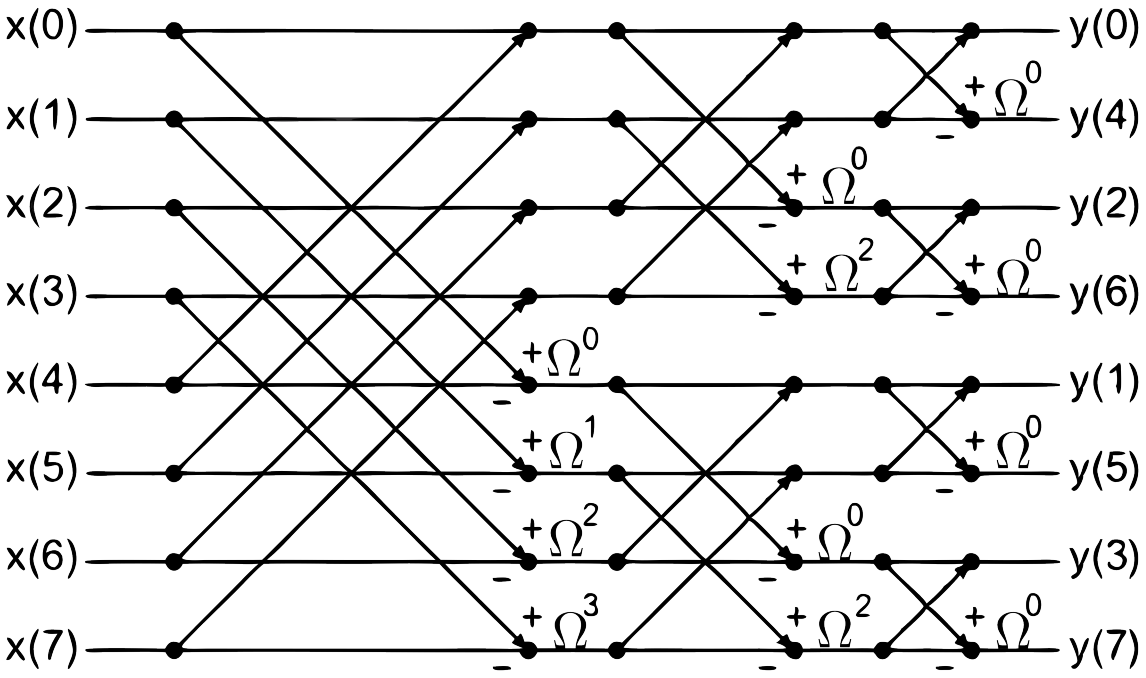




Cooley-Tukey Algorithm
bit-reversed and updated in-place (original data lost)

Stockham Algorithm
without bit-reversal updated in separate array

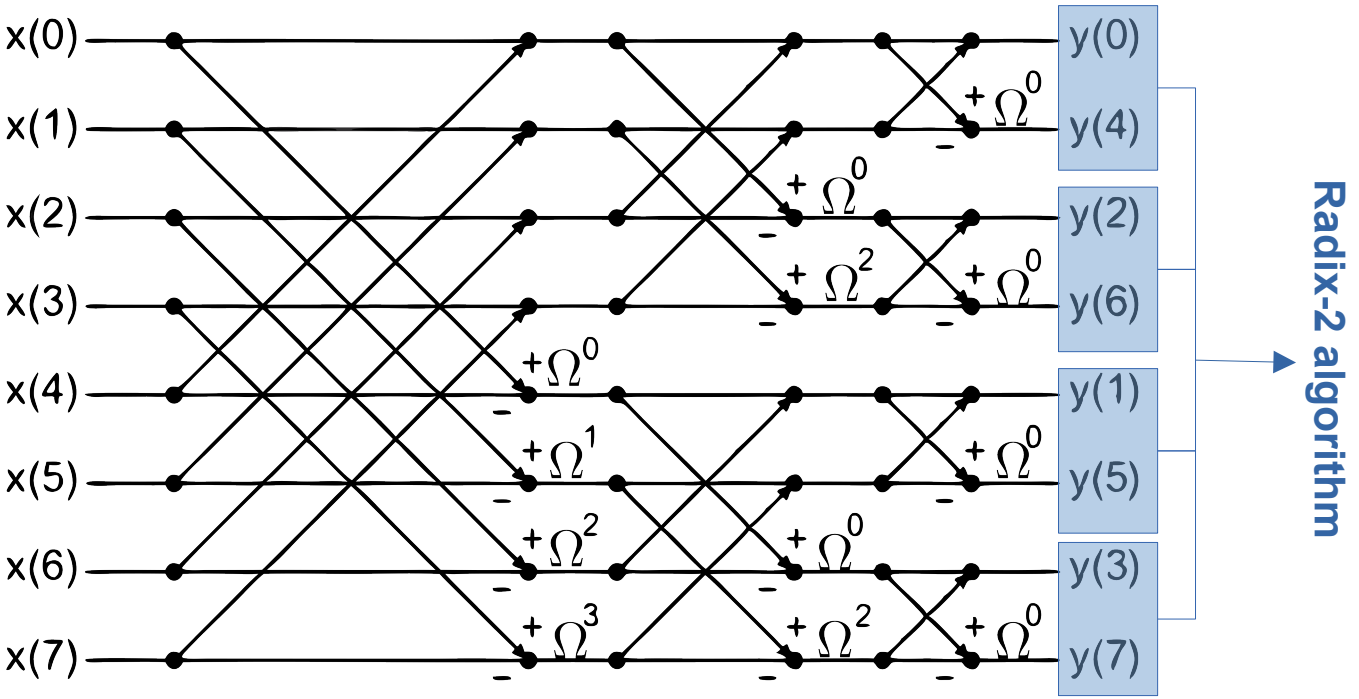
The Radix of an FFT algorithm:



Decimation in Frequency: Cooley-Tukey or Stockham

The number of data points to which the DFT is finally reduced in the FFT decomposition is called the radix.

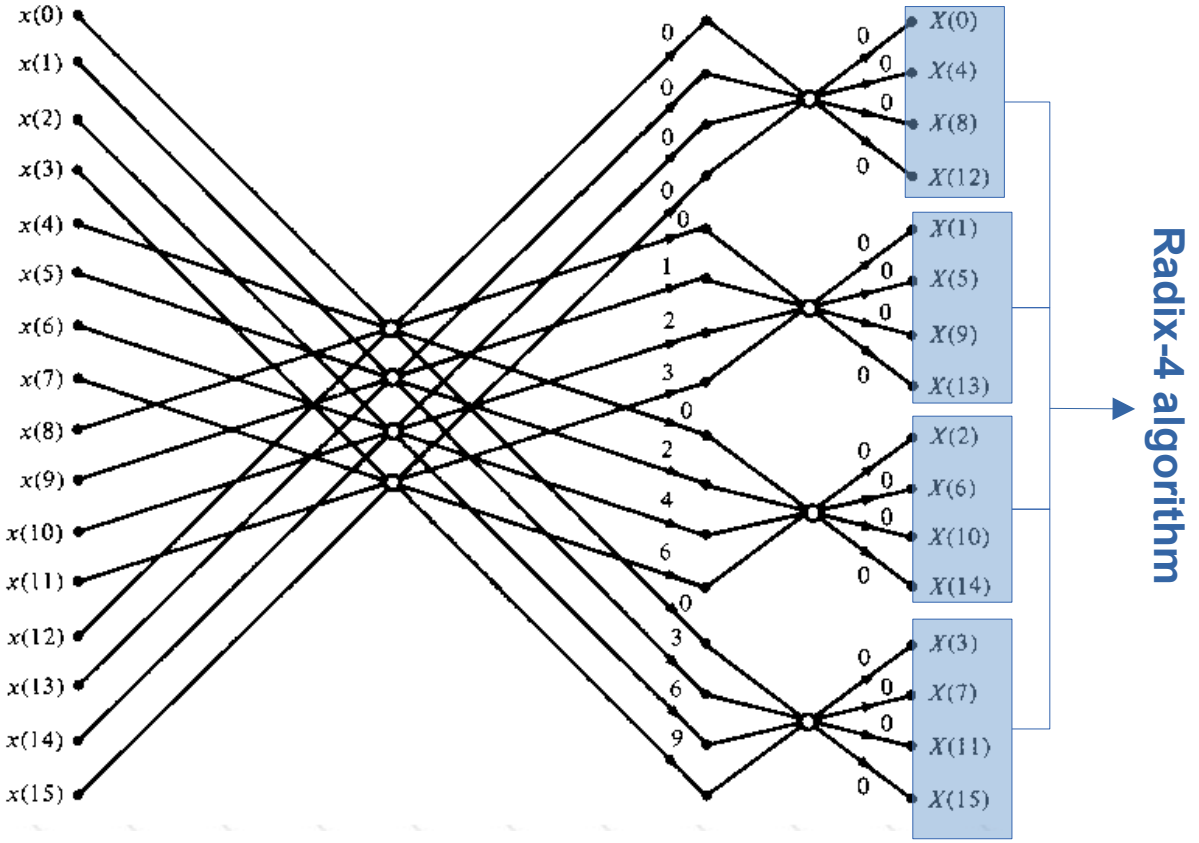
The Radix of an FFT algorithm:



Decimation in Frequency: Cooley-Tukey or Stockham

The number of data points to which the DFT is finally reduced in the FFT decomposition is called the radix.

The Radix of an FFT algorithm:

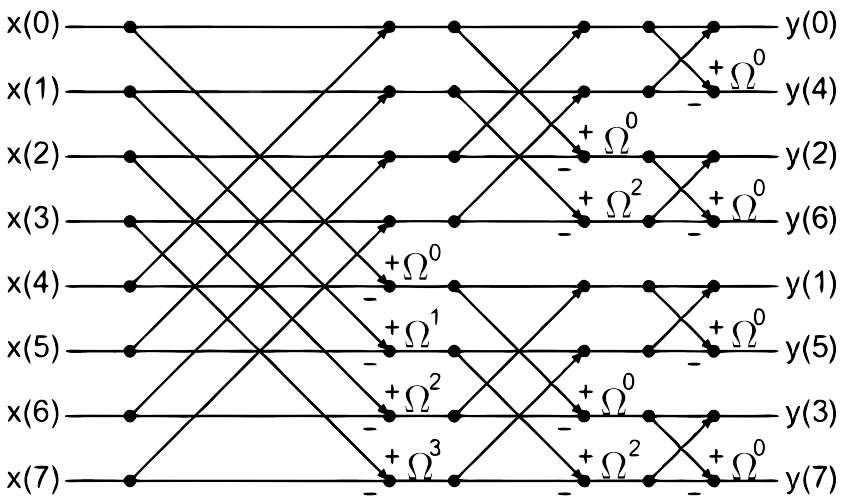


Decimation in Frequency

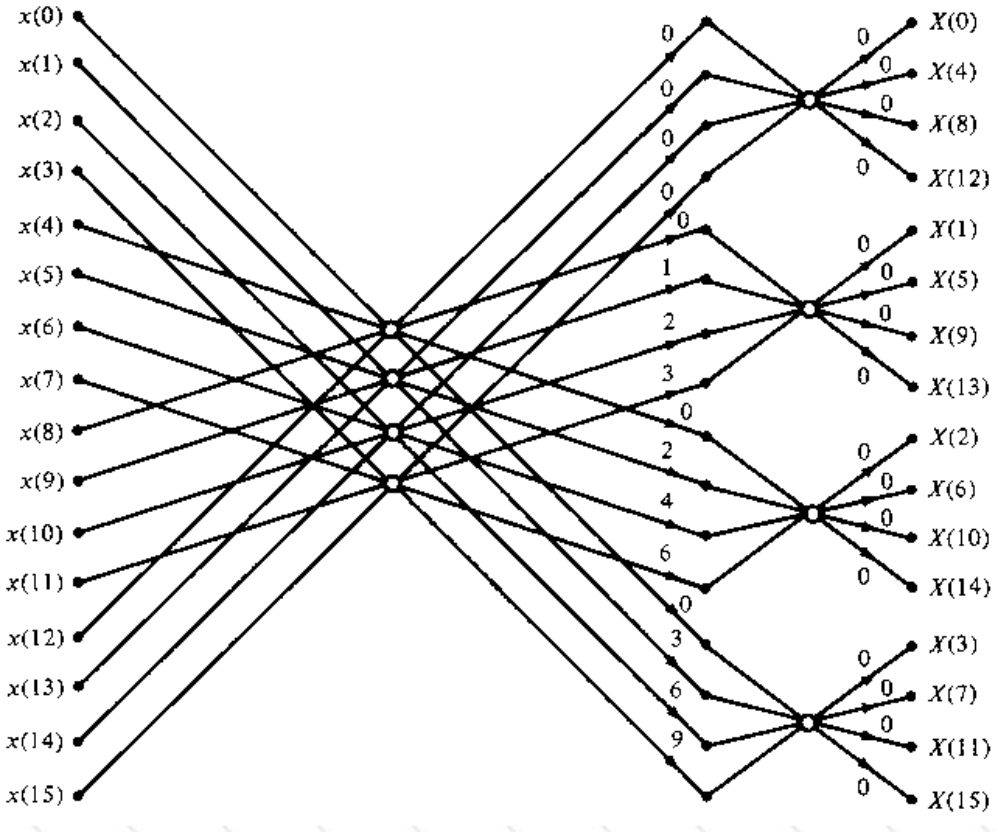
The number of data points to which the DFT is finally reduced in the FFT decomposition is called the radix.

4. The Fast Fourier Transform (FFT)

The Radix of an FFT algorithm:



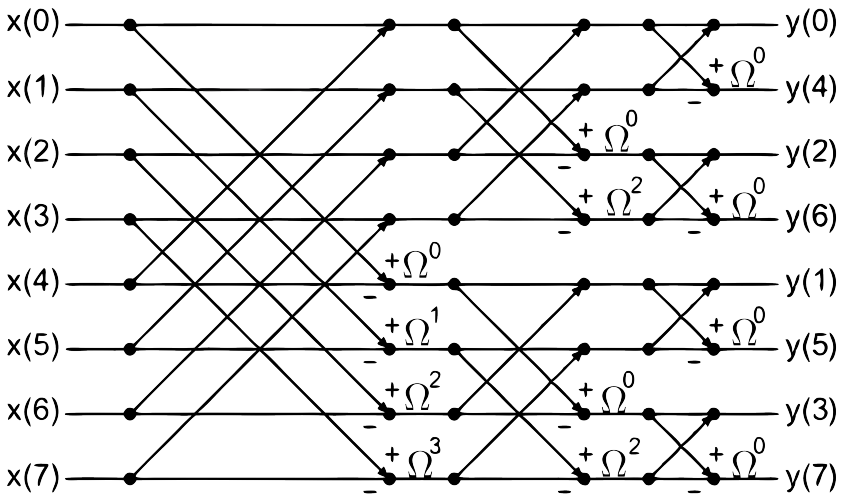
Radix-2, $N = 2^p$**, # recs** $\log_2 N$



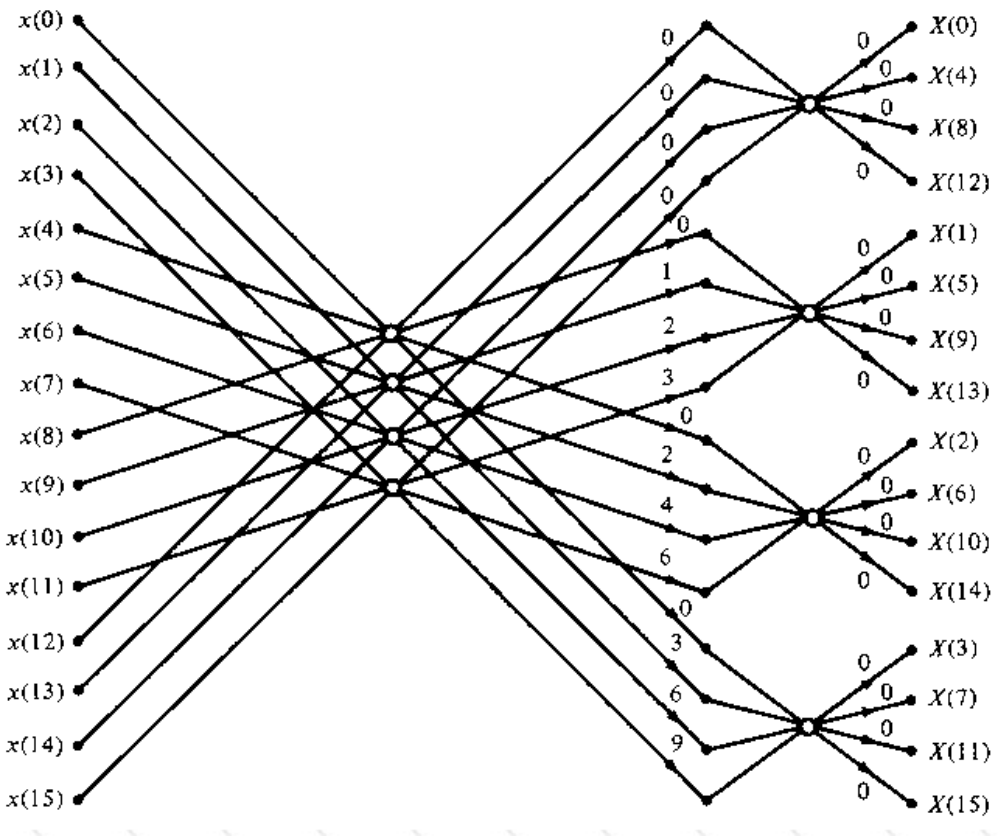
Radix-4, $N = 4^p$**, # recs** $\log_4 N$

4. The Fast Fourier Transform (FFT)

The Radix of an FFT algorithm:



Radix-2, $N = 2^p$**, # recs** $\log_2 N$



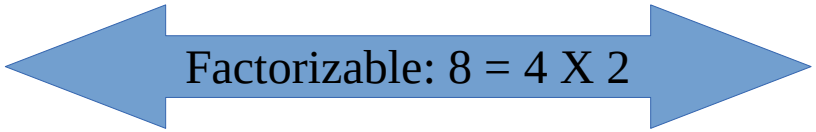
Radix-4, $N = 4^p$**, # recs** $\log_4 N$

- Higher radix means fewer recursions and more calculations per recursion
- These can **sometimes** be faster in modern CPUs with SIMD instruction sets like Intel/AMD AVX-512 and later
- SIMD CPUs have hardware that allow simultaneous floating point ops over 4+ contiguous elements in memory
- FFT backends like PocketFFT (NumPy), FFTPACK (SciPy) & FFTW-3 (PyFFTW) automagically do this

4. The Fast Fourier Transform (FFT)

- Software FFT parallelization: Multithreaded support in FFTW (PyFFTW)

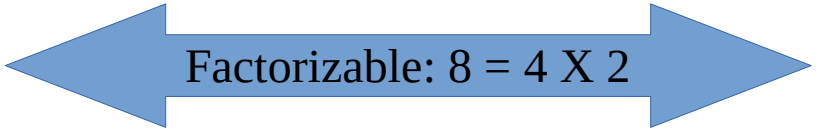
$$|f\rangle = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$$



4. The Fast Fourier Transform (FFT)

- Software FFT parallelization: Multithreaded support in FFTW (PyFFTW)

$$|f\rangle = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$$

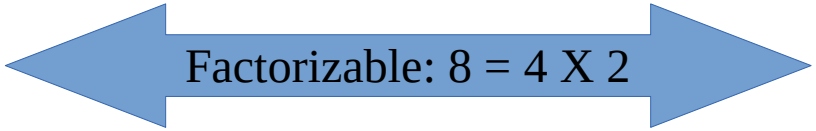


$$f = \begin{pmatrix} f_1 & f_5 \\ f_2 & f_6 \\ f_3 & f_7 \\ f_4 & f_8 \end{pmatrix}$$

4. The Fast Fourier Transform (FFT)

- Software FFT parallelization: Multithreaded support in FFTW (PyFFTW)

$$|f\rangle = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$$



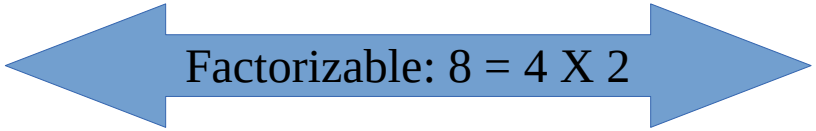
$$f = \begin{pmatrix} f_1 & f_5 \\ f_2 & f_6 \\ f_3 & f_7 \\ f_4 & f_8 \end{pmatrix}$$

$$N = M \times m$$

4. The Fast Fourier Transform (FFT)

- Software FFT parallelization: Multithreaded support in FFTW (PyFFTW)

$$|f\rangle = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$$



$$f = \begin{pmatrix} f_1 & f_5 \\ f_2 & f_6 \\ f_3 & f_7 \\ f_4 & f_8 \end{pmatrix}$$

$$N = M \times m$$

$$\bar{f}_{KM} = \sum_j \left\{ e^{2\pi ijk/m} \left[e^{2\pi ijK/(Mm)} \left(\sum_J e^{2\pi iJK/M} f_{Jj} \right) \right] \right\}$$

4. The Fast Fourier Transform (FFT)

- Software FFT parallelization: Multithreaded support in FFTW (PyFFTW)

$$|f\rangle = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8)$$

Factorizable: 8 = 4 X 2

$$f = \begin{pmatrix} f_1 & f_5 \\ f_2 & f_6 \\ f_3 & f_7 \\ f_4 & f_8 \end{pmatrix}$$

$$N = M \times m$$

$$\bar{f}_{KM} = \sum_j \left\{ e^{2\pi i j k/m} \left[e^{2\pi i j K/(Mm)} \left(\sum_J e^{2\pi i J K/M} f_{Jj} \right) \right] \right\}$$

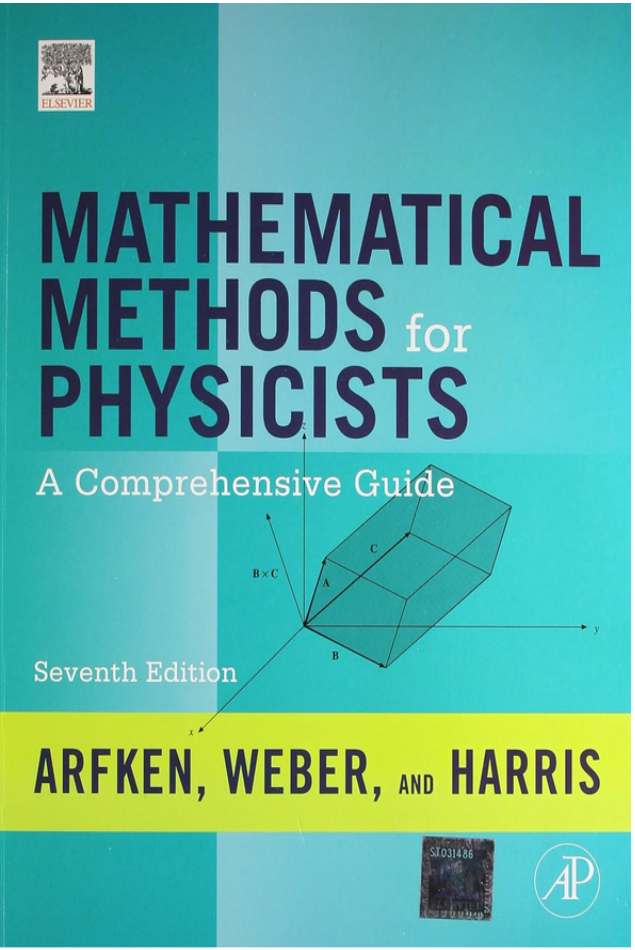
Smaller DFT independent of j
Can be done in separate procs
Not always the best way

4. The Fast Fourier Transform (FFT)

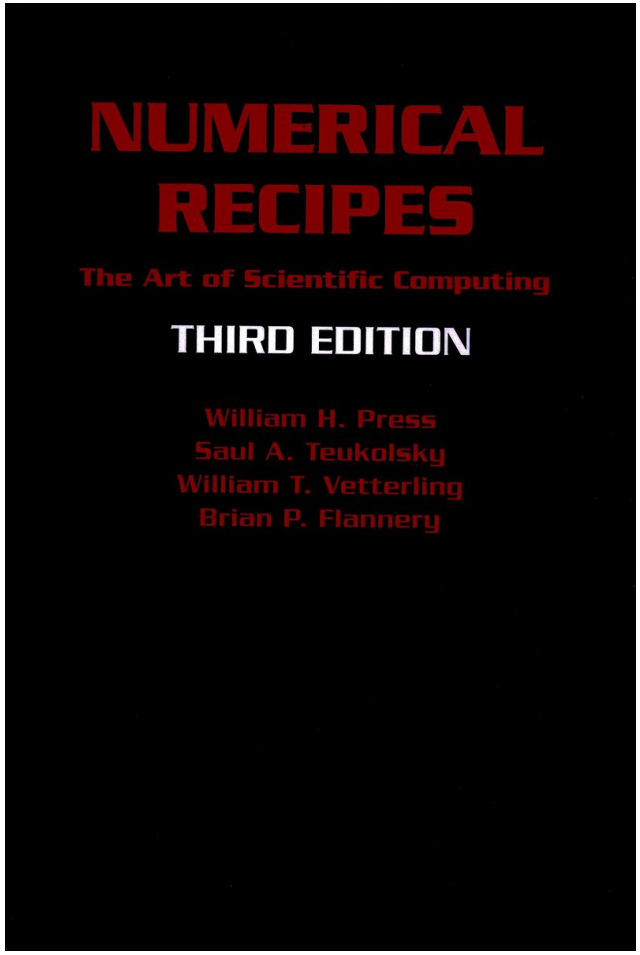
Library	Python Binding	Features			Learning Resources
		Multiple-Radix	Multithreading	SIMD Support	
FFTW (C)	pyFFTW	✓	✓	✓	https://pypi.org/project/pyFFTW/
PFFT (C)	pfft-python	✓	✓	✗	https://github.com/MP-Gadget/pfft-python
libfqfft (C++)	N/A	✓	✓	✗	https://github.com/scipr-lab/libfqfft
muFFT (C)	Python	✓	✓	✓	https://github.com/muSpectre/muFFT
PocketFFT (C++)	NumPy	✓	✗	✗	https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html
FFTPACK (FORTRAN)	SciPy	✓	✓	✓	https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html
CuFFT (nvidia GPU only)	N/A	✓	✓	✓	https://docs.nvidia.com/cuda/cufft/index.html
CuPy.fft (nvidia GPU only)	CuPy	✓	✓	✓	https://docs.cupy.dev/en/stable/user_guide/fft.html

Table 1: Comparison of FFT Libraries and their Features

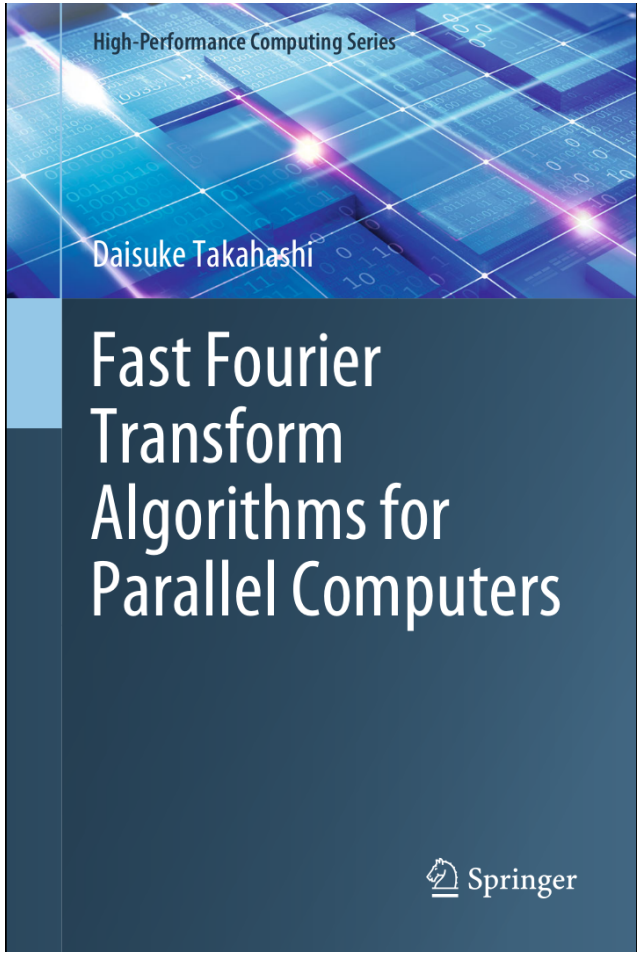
References:



ISBN: 978-9381269558



https://numerical.recipes/



ISBN: 978-9811399640

https://bit.ly/fft_lab

https://numpy.org/doc/stable/reference/routines.fft.html

5. Exercises

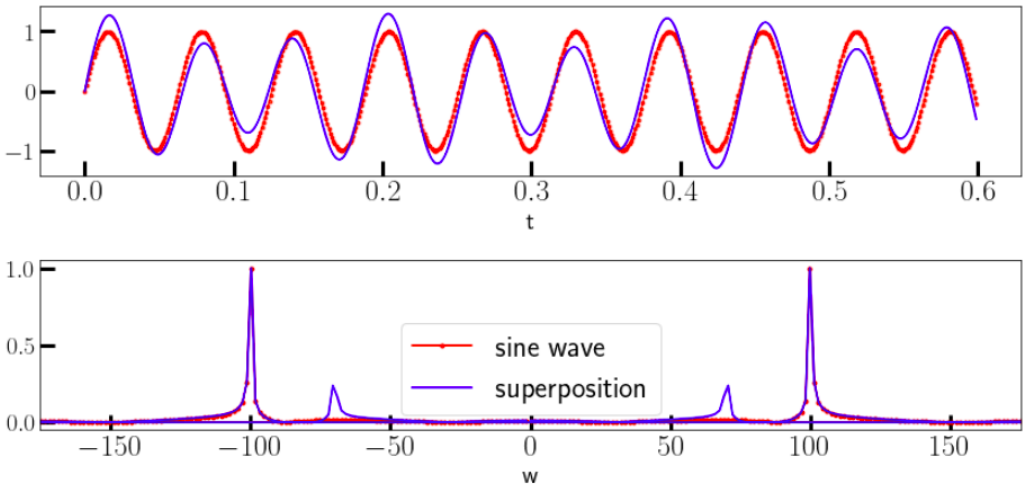
1a. FFT in Python:

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

- **Choose a sampling rate (say, 1000)**
- **Evaluate the corresponding sample interval**
- **In Python, build a 'time vector' of desired interval size in steps of the sample interval**
- **In Python, generate data of signal as a function of time where the signal is:**
  I. A sine wave of frequency 100
  II. A superposition of sine waves of different amplitudes (say, 1 and 0.3) and different frequencies (say 100 and 70 respectively)
- **For both sets of data, evaluate the FFT**
- **Pad the signal data to a size of the nearest power of 2 to the original size:**
  The 'numpy.fft.fft' routine has a kwarg of 'n=padded_size' for this
  To get the padded size, use 'padded_size = 2^nextpow2(original_size)' .
  The exponent of the nearest power of 2 to 'n' can be obtained as the ceiling of log(n) with base 2
- **Generate the frequencies where the FFTs were evaluated.**
  Use the 'numpy.fft.fftfreq' routine to do this automatically
- **Plot the absolute value of the FFTs as functions of the frequencies.**
  What can you infer from the FFT data?

5. Exercises

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12,6)
plt.rcParams['font.size'] = 20

sample_rate = 1000

Delta = 1/sample_rate
sample_size = 4000
t = np.arange(sample_size) * Delta
padded_size = 2**np.ceil(np.log2(sample_size)).astype(int)

s1 = np.sin(100*t)
s2 = s1 + 0.3 * np.sin(70*t)

sp1 = np.fft.fft(s1, n=padded_size)
sp2 = np.fft.fft(s2, n=padded_size)

freq = np.fft.fftfreq(padded_size, d=t[1]-t[0])
f, (ax1, ax2) = plt.subplots(2,1)

ax1.plot(t[0:600], s1[0:600], "r.-", label='sine wave', alpha=0.6)
ax1.plot(t[0:600], s2[0:600], "b", label='superposition')
ax1.set_xlabel('t')

ax2.plot(2*np.pi*freq, np.abs(sp1)/np.amax(np.abs(sp1)), "r.-", label='sine wave')
ax2.plot(2*np.pi*freq, np.abs(sp2)/np.amax(np.abs(sp2)), "b", label='superposition')
ax2.legend()
ax2.set_xlabel('w')
ax2.set_xlim(-175,175)
plt.show()
```



5. Exercises

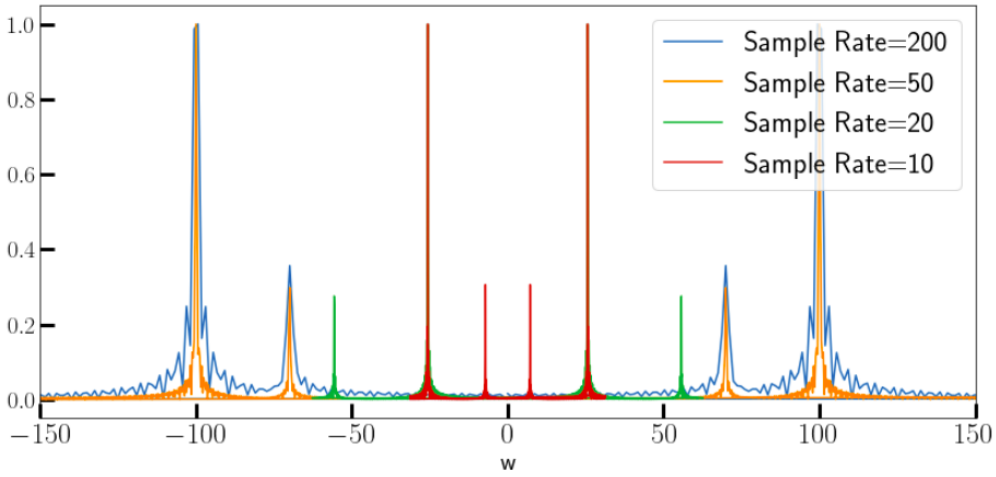
1b. FFT and Aliasing:

$$\bar{f}_m = \sum_{n=0}^{N-1} f_n \, e^{-2\pi i m n / N}$$

- **Take the previous signal and plot the FFT for a range of sample rates**
- **Can you see the effects of aliasing if the sample rate is too low? How low?**

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12,6)
plt.rcParams['font.size'] = 20

sample_rates = [200, 50, 20, 10]
sample_size = 600

for s in sample_rates:
    Delta = 1/s
    t = np.arange(sample_size) * Delta
    padded_size = 2**np.ceil(np.log2(sample_size)).astype(int)
    signal = np.sin(100*t) + 0.3 * np.sin(70*t)
    ft = np.fft.fft(signal, n=padded_size)
    freq = np.fft.fftfreq(padded_size, d=t[1]-t[0])
    plt.plot(2*np.pi*freq, np.abs(ft)/np.amax(np.abs(ft)),\
            label=f'Sample Rate = {s}')

plt.xlabel('t')
plt.legend()
plt.xlabel('w')
plt.xlim(-150,150)
plt.show()
```

$$x(t) = \sin(100t) + 0.3\sin(70t)$$



5. Exercises

2. Using FFT to filter noisy signals:

- **Goto the GitHub repository for this course and download the following files:**

  i. ‘fftdata/times.npy’,
  ii. ‘fftdata/signal.npy’,
  iii. ‘fftdata/noisy_signal.npy’

- **Load these files into numpy arrays**

- **Plot the data from the signal files as a function of the data in the times file**

- **Now, do FFTs of the signal data and plot them as functions of frequency**
  For now, assume that the time data is regularly spaced, otherwise you’ll have to regularize it by interpolation

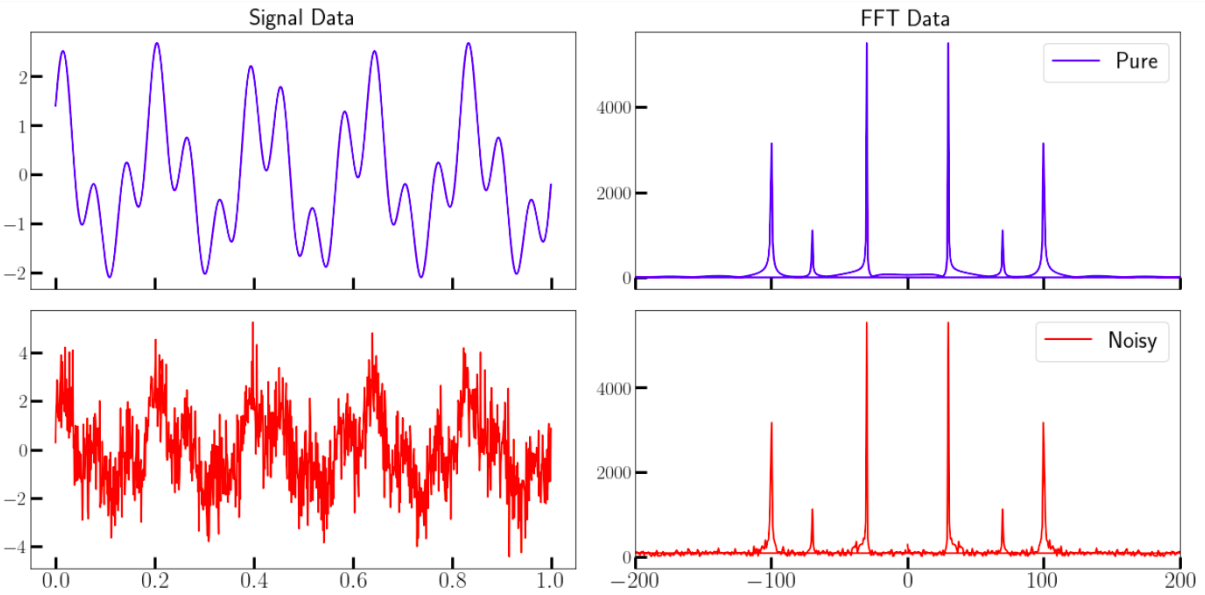
- **Is the FFT data what you expected?**

- **Clean up the FFT data of the noisy signal by removing the ‘noisy frequencies’ .**
  Visually estimate the noise threshold in the amplitudes and set all data below that to 0.

- **Finally, take an inverse FFT of the masked data and see if you succeeded in cleaning it up**

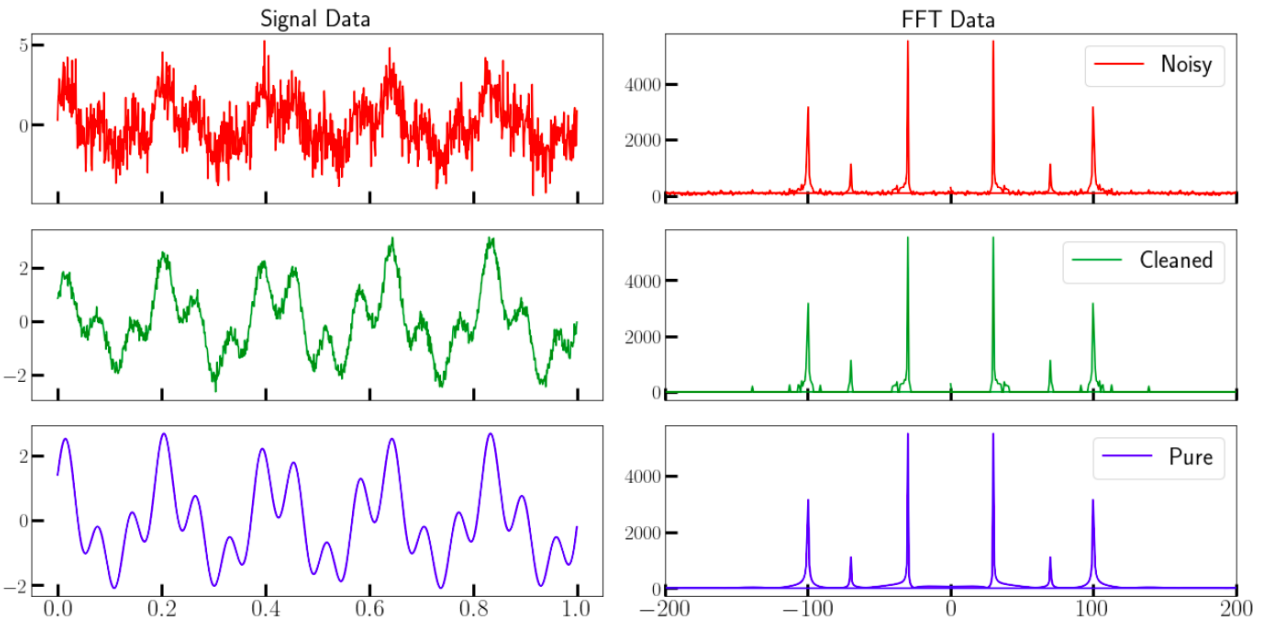
5. Exercises

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (18,9)
plt.rcParams['font.size'] = 20

# Load the datasets from files
times = np.load('fftdata/times.npy')
signal = np.load('fftdata/signal.npy')
noisy_signal = np.load('fftdata/noisy_signal.npy')

# Create a 2X2 grid of axes
f, axs = plt.subplots(2,2, sharex='col')

#Plot the raw datasets on the first column axes
axs[0,0].plot(times[0:1000], signal[0:1000], color="blue")
axs[0,0].set_title("Signal Data")
axs[1,0].plot(times[0:1000], noisy_signal[0:1000],color="red")

# Do the FFT of the datasets
sample_size = times.shape[-1]
padded_size = 2**np.ceil(np.log2(sample_size)).astype(int)
ft_sig = np.fft.fft(signal, n=padded_size)
ft_noisy = np.fft.fft(noisy_signal, n=padded_size)
freq = np.fft.fftfreq(padded_size, d=t[1]-t[0])

#Plot the fft datasets on the corresponding rows of the last column
axs[0,1].plot(2*np.pi*freq, np.abs(ft_sig), color='blue', label='Pure')
axs[0,1].set_title("FFT Data")
axs[1,1].plot(2*np.pi*freq, np.abs(ft_noisy), color='red', label='Noisy')
for ax in axs[:,1]:
    ax.set_xlim(-200,200)
    ax.legend()

plt.legend()
plt.show()
```



5. Exercises

```
import numpy as np
import matplotlib.pyplot as plt

threshold = 200
ft_abs      = np.abs(ft_noisy)
indices     = ft_abs > threshold   # filter out those value under 300
ft_clean    = indices * ft_noisy # noise frequency will be set to 0
cleaned_signal = np.fft.ifft(ft_clean) # Do inverse FFT

# Create a 2X2 grid of axes
f, axs = plt.subplots(3,2, sharex='col')

#Plot the raw datasets on the first column axes
axs[0,0].plot(times[0:1000], noisy_signal[0:1000], color='red')
axs[0,0].set_title("Signal Data")
axs[1,0].plot(times[0:1000], cleaned_signal[0:1000].real, color='green')
axs[2,0].plot(times[0:1000], signal[0:1000].real, color='blue')

#Plot the fft datasets on the corresponding rows of the last column
axs[0,1].plot(2*np.pi*freq, np.abs(ft_noisy), color='red', label='Noisy')
axs[0,1].set_title("FFT Data")
axs[1,1].plot(2*np.pi*freq, np.abs(ft_clean), color='green',
label='Cleaned')
axs[2,1].plot(2*np.pi*freq, np.abs(ft_sig), color='blue', label='Pure')
for ax in axs[:,1]:
    ax.set_xlim(-200,200)
    ax.legend()

plt.show()
```



Homework: Use a similar technique to separate the two frequencies in the cleaned data and plot them Independently.

5. Exercises

3. Compare FFT speed with DFT speed on random arrays of varying sizes.

- **Write (or find) a Python routine that:**

  i. Takes a numpy array **f** as its argument.
  ii. Evaluates the DFT vector **F** using the formula given on the right.
  iii. Evaluates the execution time.
  iv. Returns both of the above.

$$F = U \cdot f$$

$$U_{mn} = e^{-2\pi imn/N}$$

- **Generate random column vectors of sizes 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096.**

- **Run the above subroutine and plot the execution time as a function of data size. Interpret the curve.**

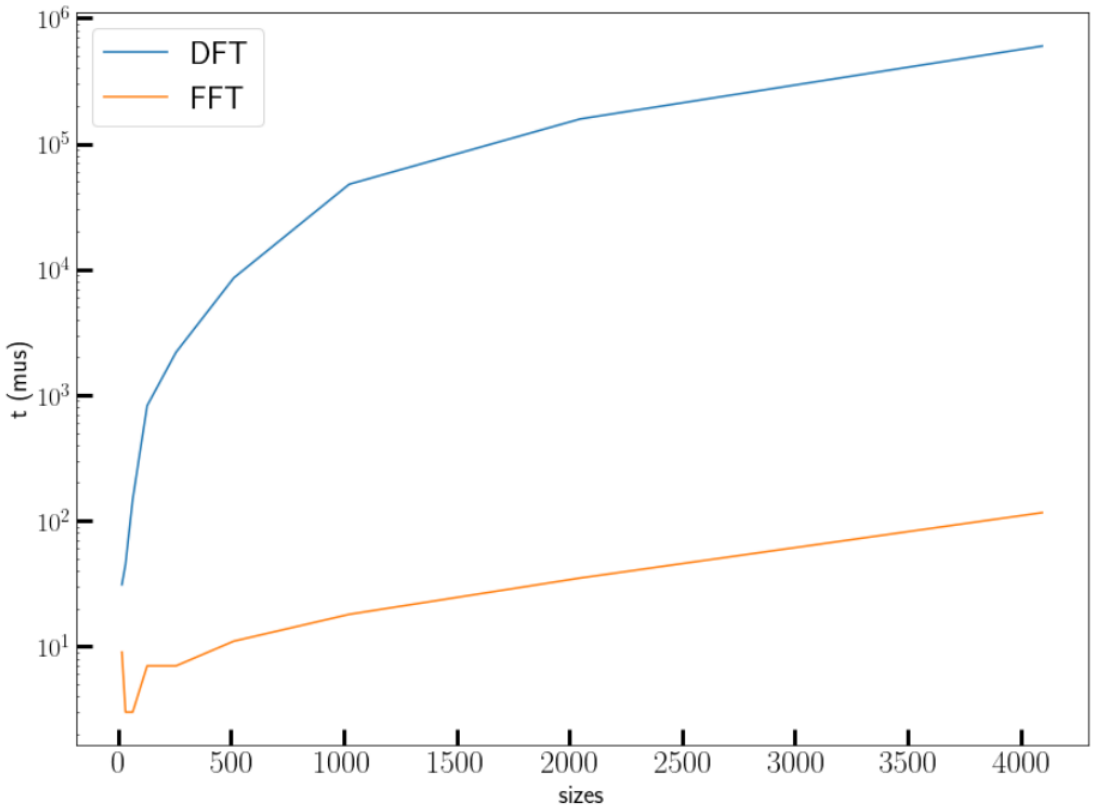
- **Repeat the steps above with the builtin FFT routine.**

- **In both cases, plot the execution time as a function of size and compare, then:**

  I. Plot the execution time for the DFT case as a function of size^2
  II. Plot the execution time for the FFT case as a function of size*log2(size)
  III. What can you infer from these graphs?

5. Exercises

```
import numpy as np
def dft(x):
    """
    Function to calculate the
    discrete Fourier Transform
    of a 1D real-valued signal x
    """
    N = x.shape[-1]
    n = np.arange(N)
    k = n.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * n / N)
    return e @ x

x = np.random.random(100)
print(np.allclose(dft(x), np.fft.fft(x)))

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (8,6)
plt.rcParams['font.size'] = 20

from timeit import Timer
defnum = 20
mysetup = 'import numpy as np\n'
mysetup += '''def dft(x):
                N = x.shape[-1]
                n = np.arange(N)
                k = n.reshape((N, 1))
                e = np.exp(-2j * np.pi * k * n / N)
                return e @ x'''

sizes = 2**np.arange(4,13)
times_dft = np.zeros_like(sizes)
times_fft = np.zeros_like(sizes)
for i,s in enumerate(sizes):
    dft = Timer(setup=mysetup, stmt="dft(np.random.random(%d))" % (s,))
    times_dft[i] = min(dft.repeat(number=defnum)) * 1e6 / defnum
    fft = Timer(setup=mysetup, stmt="np.fft.fft(np.random.random(%d))" % (s,))
    times_fft[i] = min(fft.repeat(number=defnum)) * 1e6 / defnum

plt.plot(sizes, times_dft,label="DFT")
plt.plot(sizes, times_fft, label="FFT")
plt.xlabel("sizes")
plt.ylabel("t (mus)")
plt.yscale('log')
plt.legend()
plt.show()
```



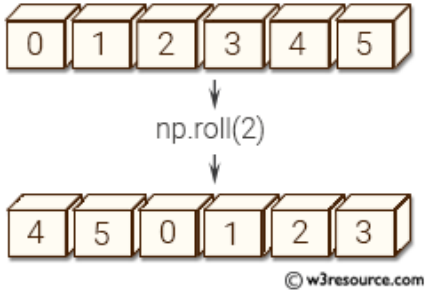
5. Exercises

4. Convolutions via FFT

- **Generate two vectors ‘f’ and ‘g’ of equal size (choose a large size, preferably a power of 2), consisting of random numbers, and evaluate the convolution vector $c = f \otimes g$, where the elements of c are given by**

$$c_l = \sum_j f_{l-j} \times g_j$$

i. Imagine that the elements of ‘f’ lie on a ring. Then, the l^{th} component of ‘c involves the components of ‘f’ after ‘rolling’ this ring by l places.
ii. In numpy, there is a function ‘numpy.roll’ that does this. Use that function.



- **Now, get the FFTs of the two vectors, compute their element-wise product, and do the IFFT of the result. Sum over this.**

- **Is this sum equal to the first result? Why?**

i. Repeat for a few different sets of two random vectors.
ii. Recall the convolution theorem

- **If they are indeed equal as expected, then which method is faster and why?**

Use the %timeit magic in Jupyter to measure runtimes

5. Exercises

4. Convolutions via FFT

```
import numpy as np

N = 2**15
f = np.random.random(N)
g = np.random.random(N)

print("Normal Convolution:")
%timeit c = np.array([np.sum(np.roll(f,-l-1)[::-1]*g) for l in range(N)])
c = np.array([np.sum(np.roll(f,-l-1)[::-1]*g) for l in range(N)])

print("\nConvolution by FFT:")
%timeit c_fft = np.fft.ifft(np.fft.fft(f) * np.fft.fft(g))
c_fft = np.fft.ifft(np.fft.fft(f) * np.fft.fft(g))

print("\nAre they equal?", np.allclose(c, c_fft))
```

Normal Convolution:
1.91 s ± 10.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Convolution by FFT:
1.4 ms ± 23.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

Are they equal? True