PHY604 Lecture 13

October 19, 2023

Review: QR decomposition algorithm

- For a give N x N starting matrix **A**:

- 1. Create an N x N array to hold **V**; initialize as identity
- 2. Calculate QR decomposition **A** = **QR**
- 3. Update **A** with new value **A** = **RQ**
- 4. Multiply **V** on the RHS with **Q**
- 5. Check off-diagonal elements of **A**. If they are less than some tolerance, we are done. Otherwise go back to 2.

Review: Multivariate Newton's method

- We can generalize Newton's method for equations with several variables
  - Can be used when we no longer have a linear system
  - Cast the problem as one of root finding
- Consider the vector function: $\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) & f_1(\mathbf{x}) & \dots & f_N(\mathbf{x}) \end{bmatrix}$
- Where the unknowns are: $\mathbf{x} = \begin{bmatrix} x_1 & x_1 & \dots & x_N \end{bmatrix}$

- Revised guess from initial guess $\mathbf{x}^{(0)}$: $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{f}(\mathbf{x}_0)\mathbf{J}^{-1}(\mathbf{x}_0)$
  - $\mathbf{J}^{-1}$ is the inverse of the Jacobian matrix:

$$J_{ij}(\mathbf{x}) = \frac{\partial f_i(\mathbf{x})}{\partial x_i}$$

- To avoid taking the inverse at each step, solve with Gaussian substitution:

$$\mathbf{J}\delta\mathbf{x}^k = -\mathbf{f}(\mathbf{x}^k)$$

Review: Steepest descent

- Used for finding roots, minima, or maxima of functions of several variables
- Based on the idea of moving downhill with each iteration, i.e., opposite to the gradient
  - If current position is $\mathbf{x}_n$, next step is:

$$x_{n+1} = x_n - \alpha_n \nabla f(x_n)$$

- Determine the step size α such that we reach the line minimum in direction of the gradient:

$$\frac{d}{d\alpha_n} f[x_{n+1}(\alpha_n)] = -\nabla f(x_{n+1}) \cdot \nabla f(x_n) = 0$$

- Find root of function of α :

$$g(\alpha) = \nabla f[x_{n+1}(\alpha)] \cdot \nabla f(x_n) = 0$$

Today’s lecture: FFTs and curve fitting

- Fourier Transforms

Fourier analysis

- Study of the way general functions can be represented by sums of trigonometric functions

- Applications in: Signal processing, solving of PDEs, interpolations,…

- In condensed matter/solid state physics, we often make use of reciprocal space because of Bloch’s theorem
    - Certain operators like spatial derivatives and convolutions are simpler in reciprocal space
    - Plane waves are often used as a basis to represent functions

Fourier Series

- A periodic function defined on an interval $0 \leq x < L$ can be written as a Fourier series:

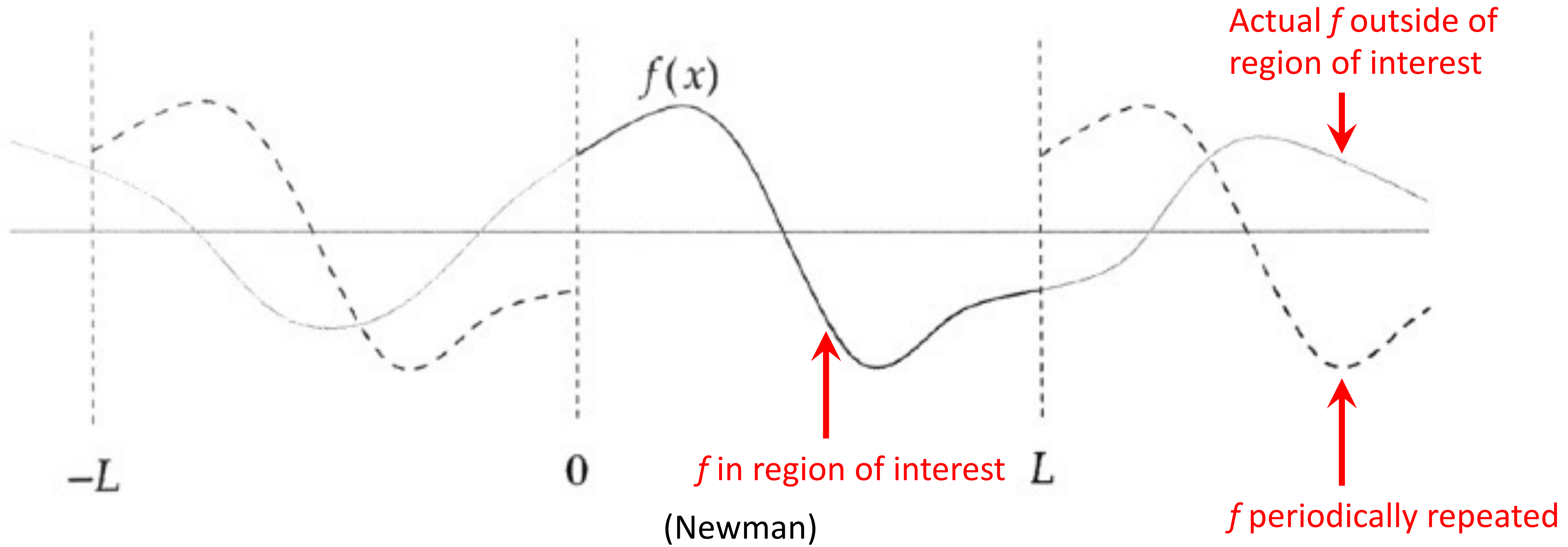
$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\pi kx}{L}\right) + \sum_{k=0}^{\infty} \beta_k \sin\left(\frac{2\pi kx}{L}\right)$$

$$= \sum_{k=-\infty}^{\infty} \gamma_k e^{2\pi ikx/L}$$

- Where:

$$\gamma_k = \begin{cases} \frac{1}{2}(\alpha_{-k} + i\beta_{-k}) & \text{if } k < 0 \\ \alpha_0 & \text{if } k = 0 \\ \frac{1}{2}(\alpha_k - i\beta_k) & \text{if } k > 0 \end{cases}$$

Fourier series for nonperiodic functions



(Newman)

- If function is not periodic, we can take the portion over the range of interest (0 to L) and repeat it
- Fourier series will give correct result from 0 to L

Fourier series coefficients

- Formally, the coefficients are: $\gamma_k = \dfrac{1}{L}\displaystyle\int_0^L f(x) e^{-2\pi i k x/L}$
- Usually, we are dealing with *f*(*x*) that is discrete data
- Use the trapezoid rule to calculate the integral:

$$\gamma_k = \frac{1}{N}\left[\frac{1}{2}f(0) + \frac{1}{2}f(L) + \sum_{n=1}^{N-1} f(x_n)\exp\left(-i\frac{2\pi k x_n}{L}\right)\right]$$

- Where sample points are $x_n = n\, L/N$
- Since we assume periodicity, *f*(0)=*f*(*L*) so:

$$\gamma_k = \frac{1}{N}\sum_{n=0}^{N-1} f(x_n)\exp\left(-i\frac{2\pi k x_n}{L}\right)$$

Discrete Fourier transform

- Assume function evaluated on equally-spaced points *n*:

$$F_k = \sum_{n=0}^{N-1} f_n \exp\left(-i\frac{2\pi nk}{N}\right)$$

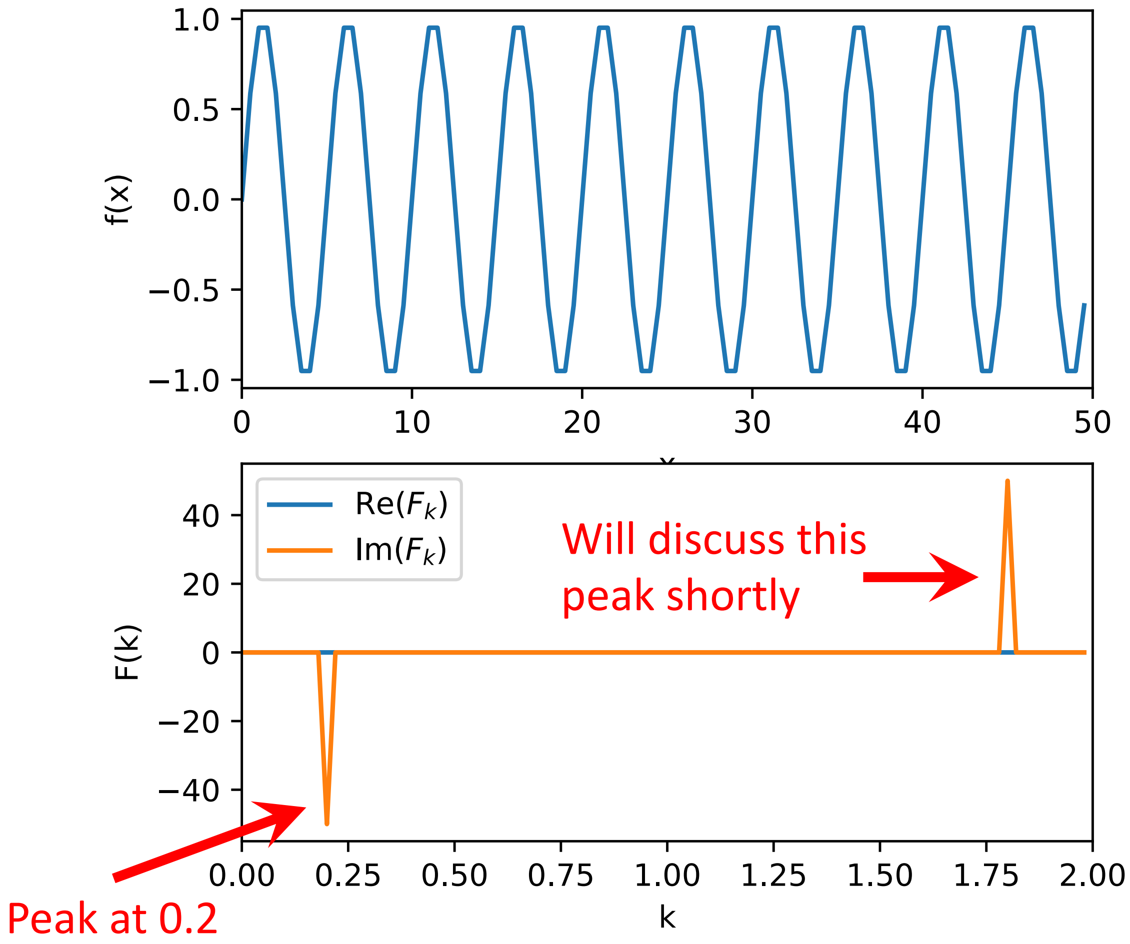
  - (dropped the 1/*N* from pervious slide, matter of convention)
  - This is the discrete Fourier transform (DFT)
  - Does not require us to know the positions x_n of sample points, or even width *L*
- We can define an inverse discrete Fourier transform to recover the initial function:

$$f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k \exp\left(i\frac{2\pi nk}{N}\right)$$

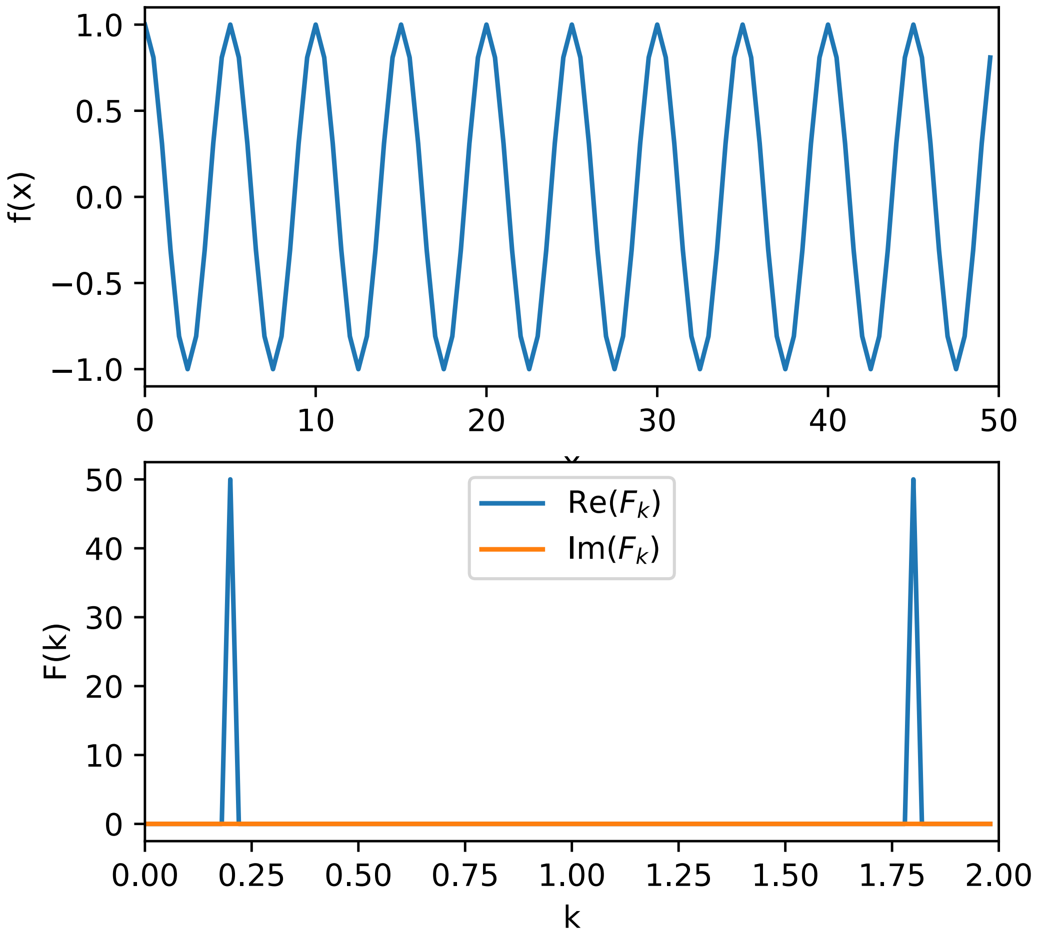
  - (1/*N* reappears)
- “Exact” (up to rounding errors), even though we used the trapezoid rule
  - see e.g., Newman Sec. 7.2

Example: Fourier transform of monochromatic functions

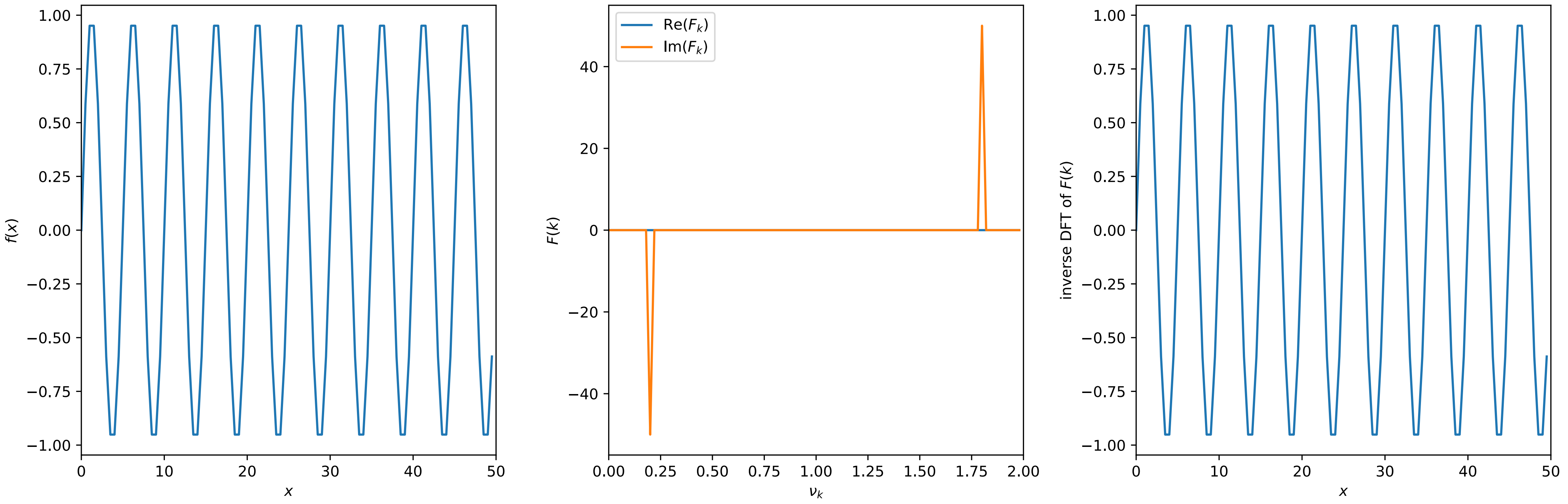
- $f(x) = \sin(2\pi\nu_0 x)$ with $\nu_0 = 0.2$:
  - Peak in the imaginary part will appear at the characteristic frequency ν_0



- $f(x) = \cos(2\pi\nu_0 x)$ with $\nu_0 = 0.2$:
  - Peak in the real part will appear at the characteristic frequency ν_0



“Exact” in that inverse DFT gives the same function back up to rounding errors



Real and imaginary parts

- Real parts represent even functions (e.g., Cosine)
- Imaginary parts represent odd functions (e.g., Sine)

- Could also think in terms of amplitude and phase

- For real f_n:

$$\mathrm{Re}(F_k) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{2\pi nk}{N}\right)$$

$$\mathrm{Im}(F_k) = \sum_{n=0}^{N-1} f_n \sin\left(\frac{2\pi nk}{N}\right)$$

Frequencies in DFTs

- In the DFT, the physical coordinate value, x_n , does not enter—instead, we just look at the index n itself
  - Assumes data is regularly gridded

- Many FFT routines will return frequencies in "index" space, e.g., $k_{\mathrm{freq}} = 0, 1/N, 2/N, 3/N, \ldots$

- Lowest frequency: $1/L$ (corresponds largest wavelength, $\lambda = L$: entire domain)
- Highest frequency ~ $N/L \sim 1/\Delta x$ (corresponds to shortest wavelength, $\lambda = \Delta x$)

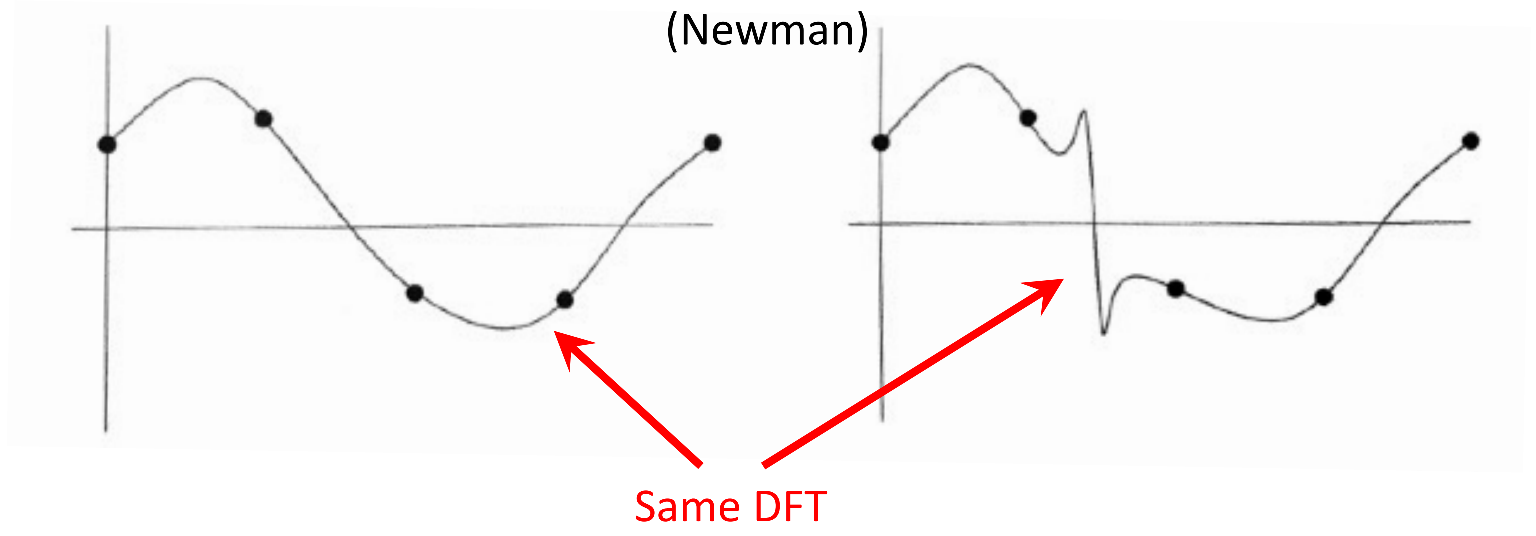
k=0 is the DC offset

- Real part is the average:

$$\mathrm{Re}(F_0) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{2\pi n 0}{N}\right) = \sum_{n=0}^{N-1} f_n$$

$$\mathrm{Im}(F_0) = \sum_{n=0}^{N-1} f_n \sin\left(\frac{2\pi n 0}{N}\right) = 0$$

Caveat: DFT exact only for sampled points



- Functions with the same values at the sample points will have the same DFT

DFTs of real functions

- Works for real or complex functions, but most of the time, we have real data
- If f_n is real, we can simplify further:
- Consider F_k for k in the upper half of the range: $k = N\text{-}r$ where: $1 \leq r < \frac{1}{2}N$

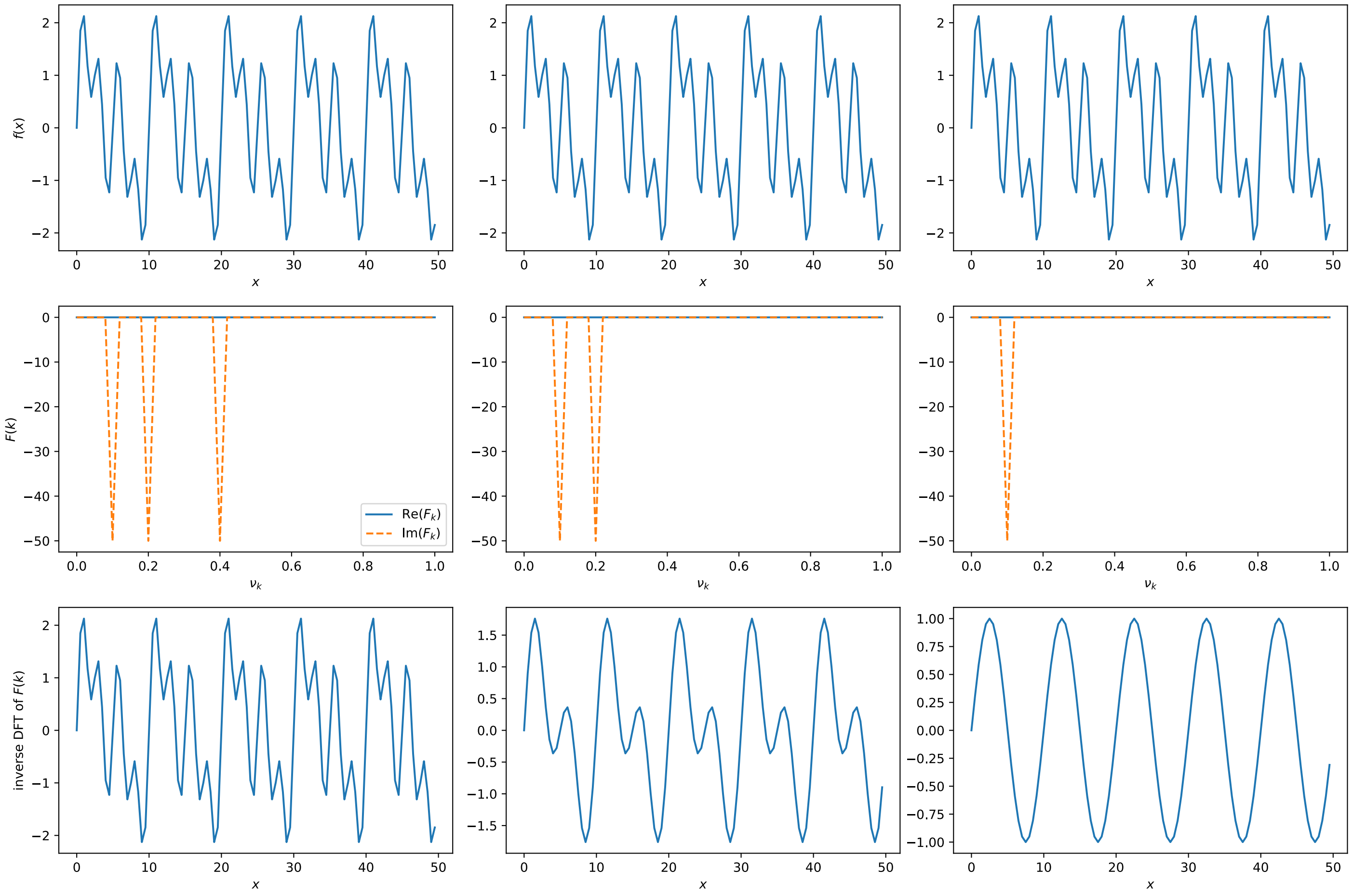
$$F_{N-r} = \sum_{n=0}^{N-1} f_n \exp\left(-i\frac{2\pi(N-r)n}{N}\right) = \sum_{n=0}^{N-1} f_n \exp\left(i\frac{2\pi rn}{N}\right) = F_r^*$$

- Therefore, for real functions, only need to calculate F_k for $0 \leq k \leq \frac{1}{2}N$

What can we do with the DFT? E.g., filtering

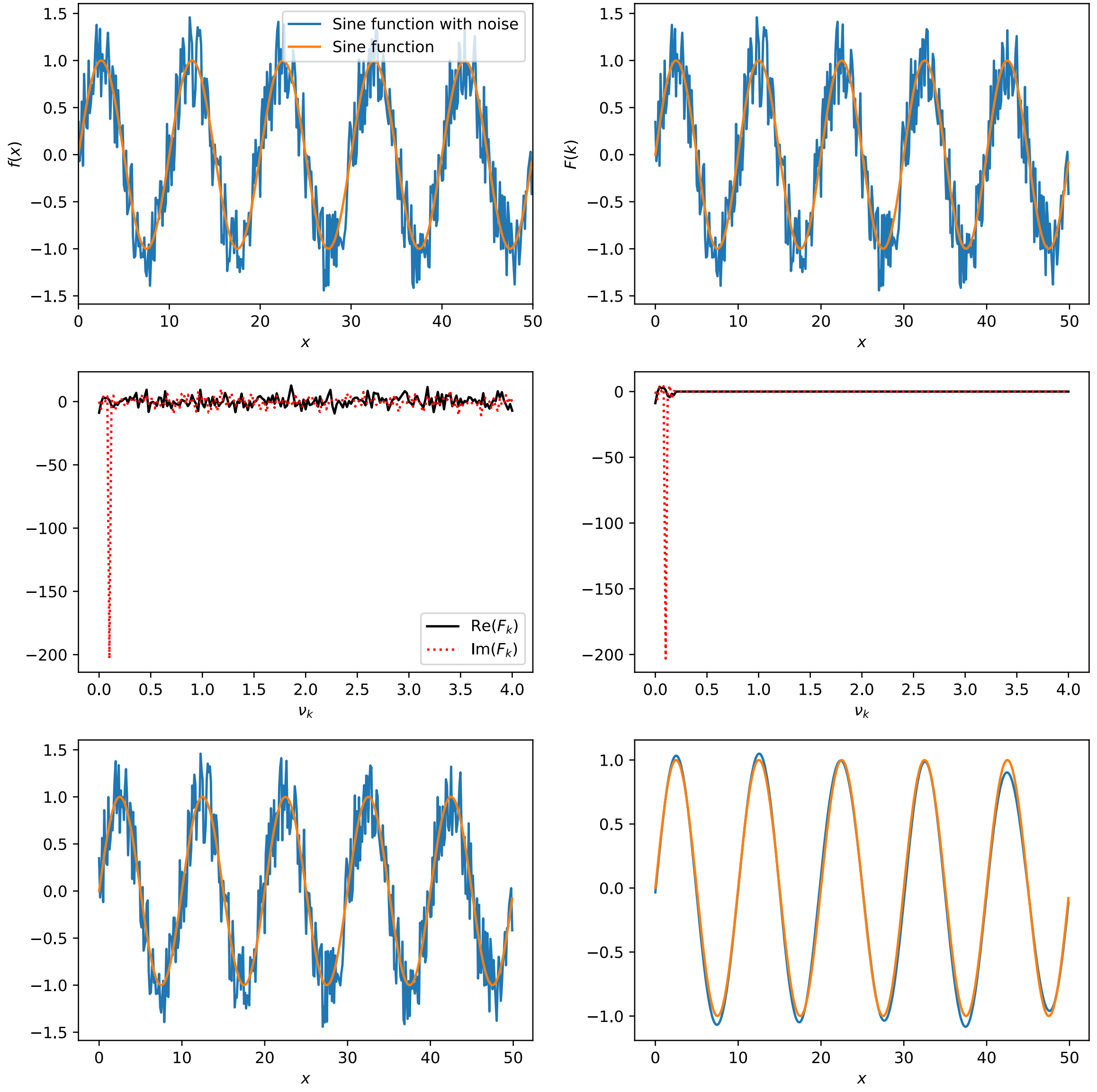
- Can use DFT to remove wither high or low frequency “noise” from a signal
- E.g., three sine functions:

Remove frequencies in DFT one at a time:

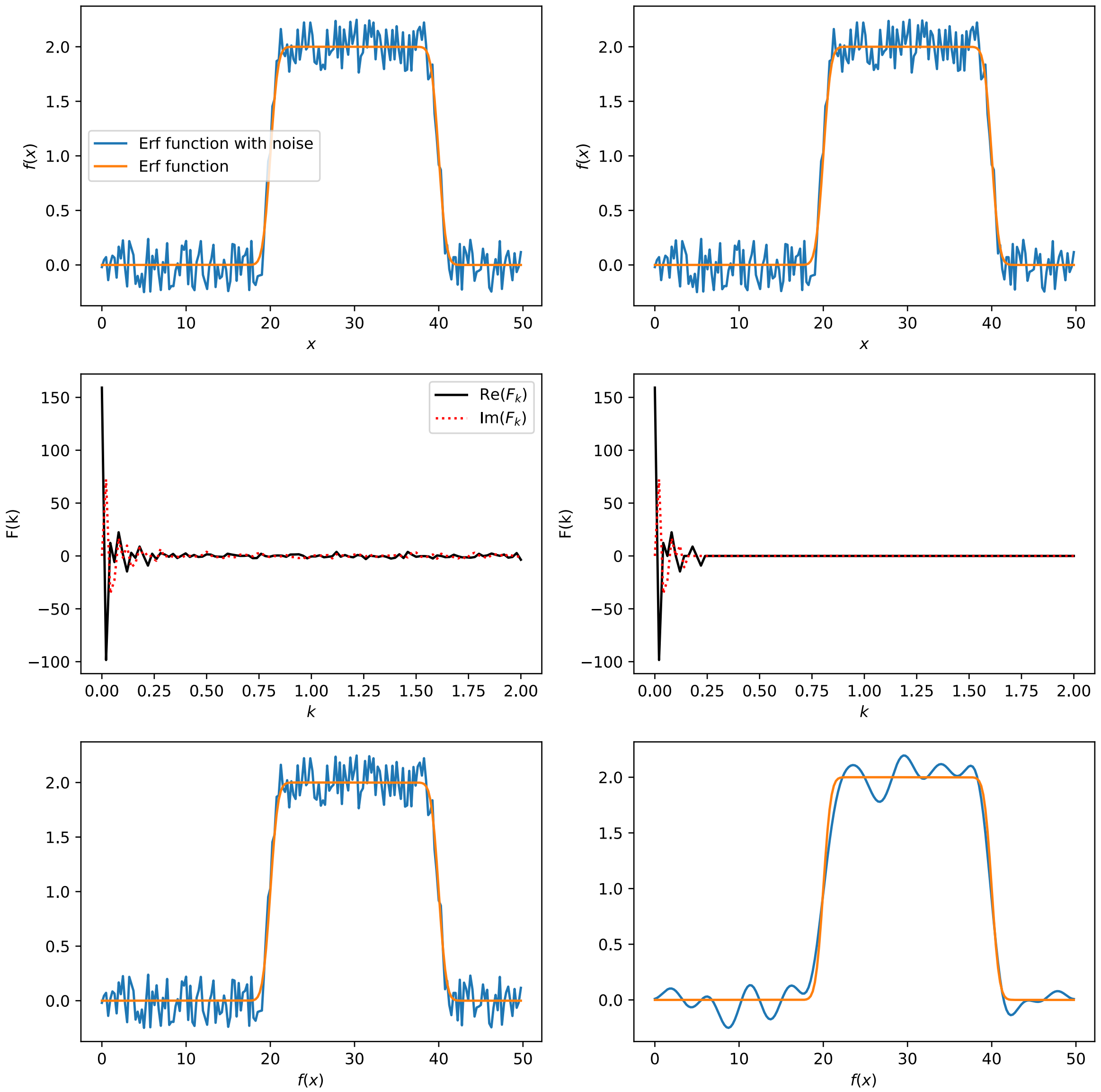


What can we do with the DFT? E.g., filtering

- Sin function with noise:



- Error function with noise:



Two-dimensional Fourier transforms

- Simply transform with respect to one variable and then the other
- Consider function on *M* x *N* grid
  - 1. Perform DFT on each of the *M* rows:

$$F'_{ml} = \sum_{n=0}^{N-1} f_{mn} \exp\left(-i\frac{2\pi ln}{N}\right)$$

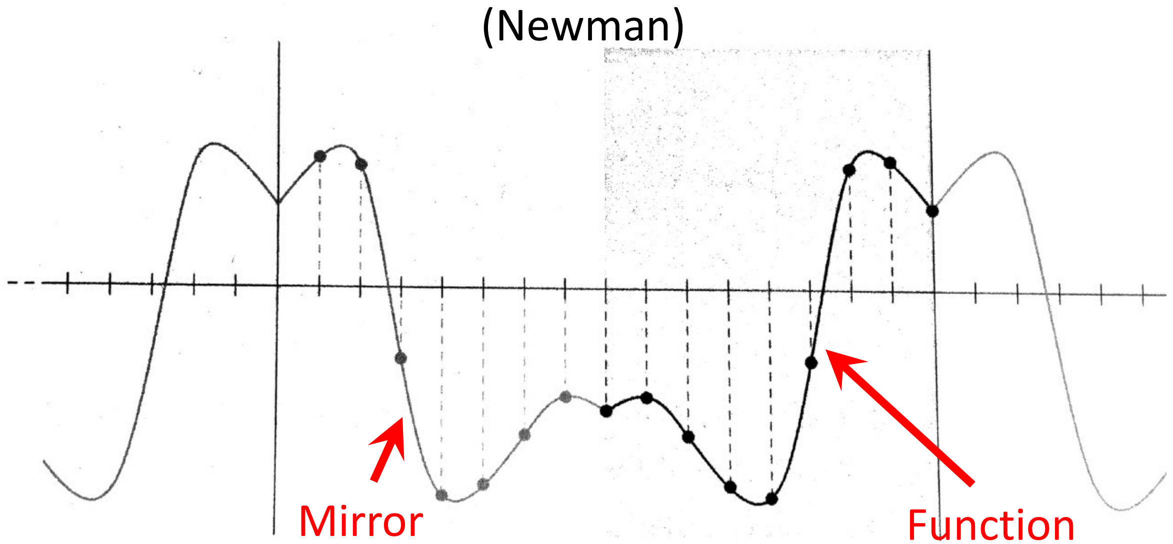
  - 2. Take *l*th coefficient in each of the *M* rows and DFT:

$$F_{kl} = \sum_{m=0}^{M-1} F'_{ml} \exp\left(-i\frac{2\pi km}{M}\right)$$

- Combining these gives:

$$F_{kl} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \exp\left[-i2\pi\left(\frac{km}{M} + \frac{ln}{N}\right)\right]$$

Cosine transformation (see Newman Sec. 7.3)



- Can also construct Fourier series from using sine and cosine functions instead of complex exponentials
- Cosine series: Can only represent functions symmetric about the midpoint of the interval
  - Can enforce this for any function by mirroring it, and then repeating the mirrored function
- Different ways of writing it (see Newman):

$$F_k = \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right), \quad f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k \cos\left(\frac{\pi k(n+\frac{1}{2})}{N}\right)$$

Benefits of the cosine transformation

- Only involves real functions
- Does not assume samples are periodic (i.e., first point and last point are the same)
  - Avoids discontinuities from periodically repeating function over interval
  - Often preferable for data that is not intrinsically periodic

- Used for compressing images and other media
  - JPEG, MPEG

- Can also define a sine transformation
  - Requires that function vanish at either end of its range

Fast Fourier transforms

- DFTs shown before have a double sum, so scale something like N^2 operations
  - We can do it in much less
- Consider the DFT: $F_k = \sum_{n=0}^{N-1} f_n \exp\left(-i\frac{2\pi nk}{N}\right)$
- Take the number of samples to be a power of 2: $N = 2^m$
- Break F_k into n even and n odd. For the even terms:

$$F_k^{\text{even}} = \sum_{r=0}^{\frac{1}{2}N-1} f_{2r} \exp\left(-i\frac{2\pi k(2r)}{N}\right) = \sum_{r=0}^{\frac{1}{2}N-1} f_{2r} \exp\left(-i\frac{2\pi kr}{N/2}\right)$$

- Just another Fourier transform, but with $N/2$ samples

Fast Fourier transforms continued

- For the odd terms:

$$\sum_{r=0}^{\frac{1}{2}N-1} f_{2r+1} \exp\left(-i\frac{2\pi k(2r+1)}{N}\right) = e^{-i2\pi k/N} \sum_{r=0}^{\frac{1}{2}N-1} f_{2r+1} \exp\left(-i\frac{2\pi kr}{N/2}\right) = e^{-i2\pi k/N} F_k^{\text{odd}}$$

- Therefore:

$$F_k = F_k^{\text{even}} + e^{-i2\pi k/N} F_k^{\text{odd}}$$

- So full DFT is sum of two DFTs with half as many points

- Now repeat the process until we get down to a single sample where:

$$F_0 = \sum_{n=0}^{0} f_n e^0 = f_0$$

Procedure for FFT

- 1. Start with (trivial) FT of single samples:

$$F_0 = \sum_{n=0}^{0} f_n e^0 = f_0$$

- 2. Combine them in pairs using:

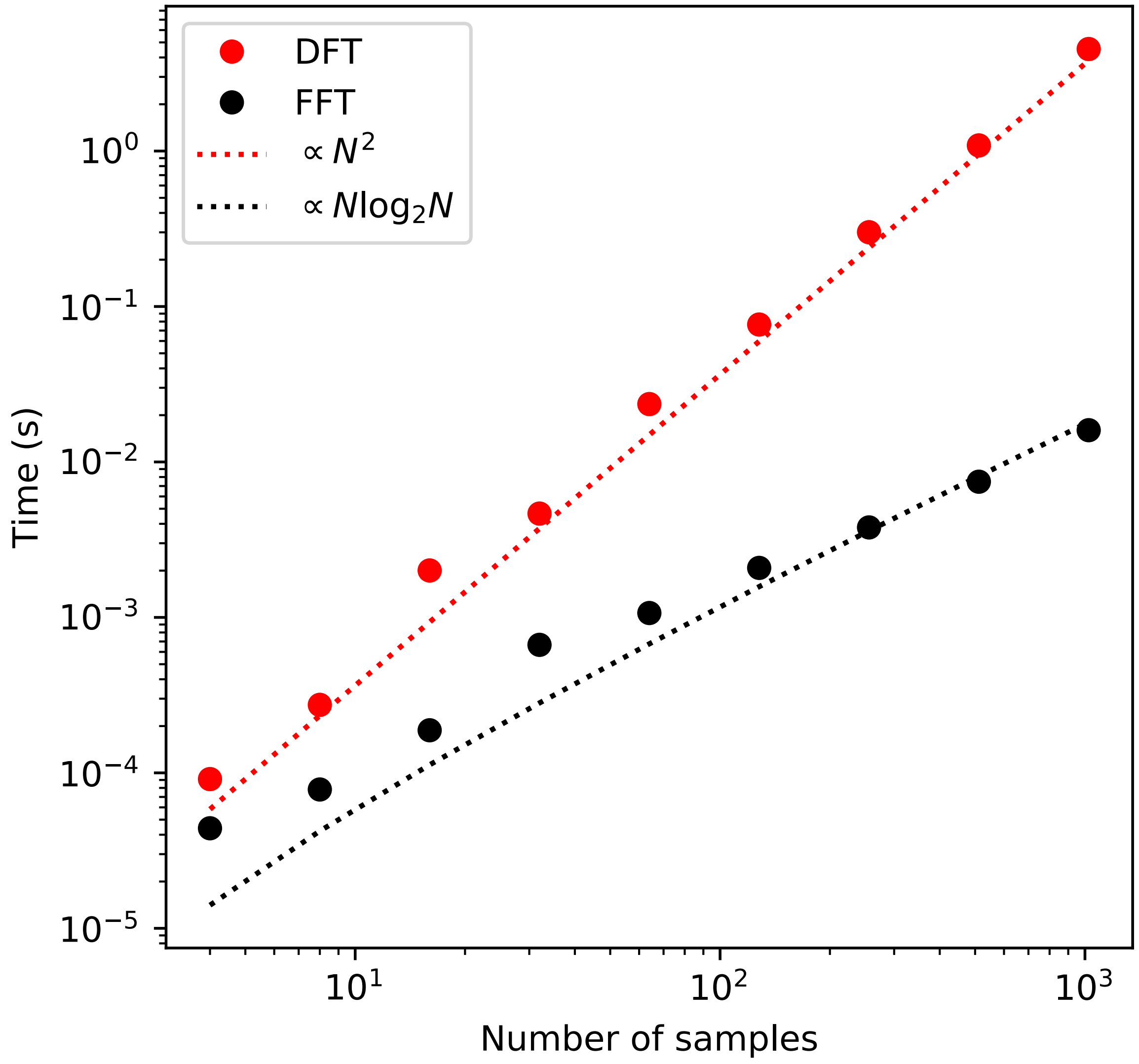
$$F_k = F_k^{\text{even}} + e^{-i2\pi k/N} F_k^{\text{odd}}$$

- 3. Continue combining into fours, eights, etc. until the full transform on the full set of samples is reconstructed

Speed up

- First "round" we have N samples
- Next round we combine these into pairs to make $N/2$ transforms with two coefficients each: N coefficients
- Next round we combine these into fours to make $N/4$ transforms with four coefficients each: N coefficients
- …
- For 2^m samples we have $m = \log_2 N$ levels, so the number of coefficients we have to calculate is $N \log_2 N$
- Way better scaling than N^2!

Speed up of FFT vs DFT



Libraries for FFT

- FFTW (fastest Fourier transform in the west)
  - https://www.fftw.org/
  - C subroutine library
  - Open source

- Intel MKL (math kernel library)
  - https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html#gs.bu9rfp
  - Written in C/C++, fortran
  - Also involves linear algebra routines
  - Not open source, but freely available
  - Often very fast, especially on intel processors

Python’s fft

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

- fft/ifft: 1-d data
  - By design, the k=0, ... N/2 data is first, followed by the negative frequencies. These later are not relevant for a real-valued $f(x)$
  - k's can be obtained from fftfreq(n)
  - fftshift(x) shifts the k=0 to the center of the spectrum
- rfft/irfft: for 1-d real-valued functions. Basically the same as fft/ifft, but doesn't return the negative frequencies
- 2-d and n-d routines analogously defined

After class tasks

- Homework 3 due Oct. 26

- Readings
  - FFTs:
    - Newman Ch. 7
    - https://en.wikipedia.org/wiki/Discrete_Fourier_transform