PHY604 Lecture 12

September 30, 2021

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:

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

Review: Discrete Fourier transform

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

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

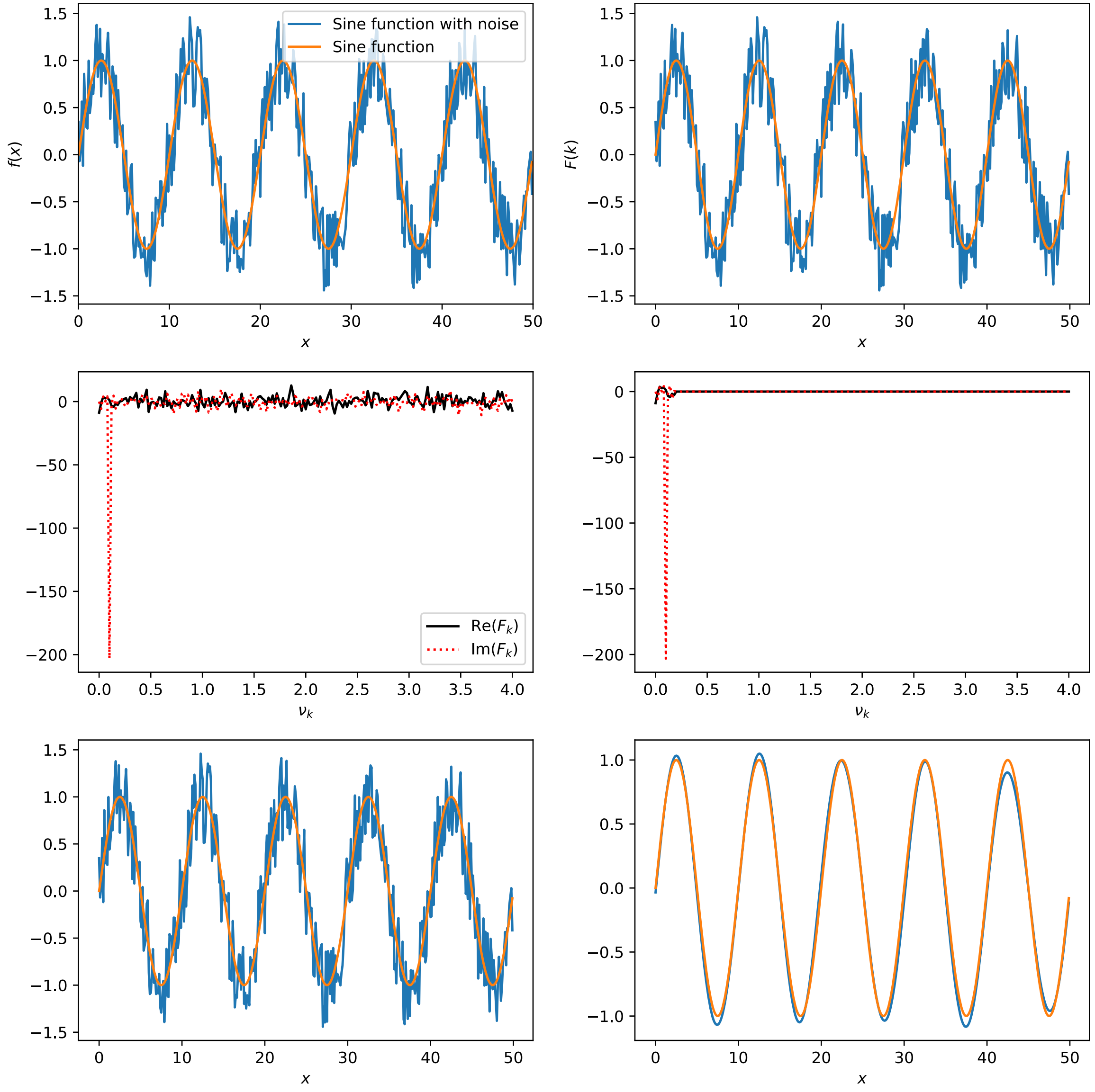
  - (dropped the 1/*N* from pervious slide, matter of convention)
  - This is the discrete Fourier transform (DFT)
  - Does not require us to know the positions x_n of sample points, or even width *L*
- We can define an inverse discrete Fourier transform to recover the initial function:

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

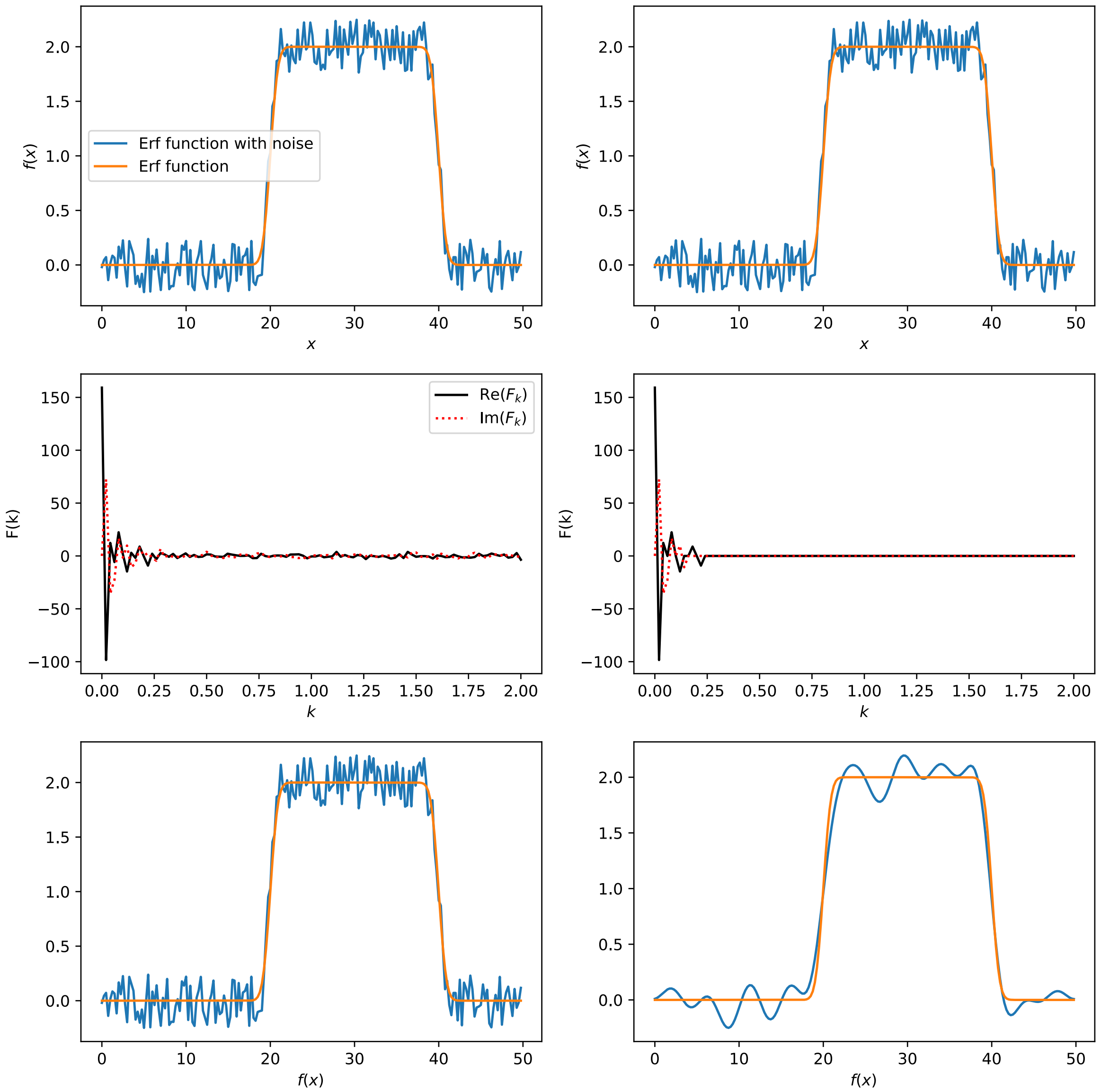
  - (1/*N* reappears)
- "Exact" (up to rounding errors), even though we used the trapezoid rule
  - see e.g., Newman Sec. 7.2

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

- Sin function with noise:



- Error function with noise:



Today's lecture:
FFTs and curve fitting

- More on Fourier Transforms
  - 2D FT
  - Cosine transformation
  - FFTs
- Curve fitting

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:

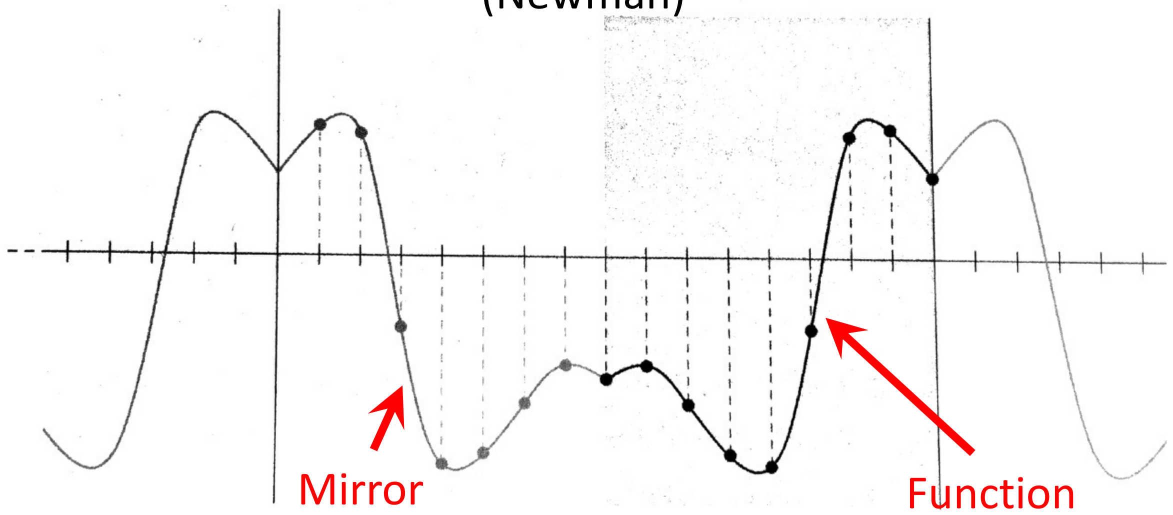
$$F_{kl} = \sum_{m=0}^{M-1} F'_{ml} \exp\left(-i\frac{2\pi km}{M}\right)$$

- 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)

(Newman)



- 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:

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

- Therefore:

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

- 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:

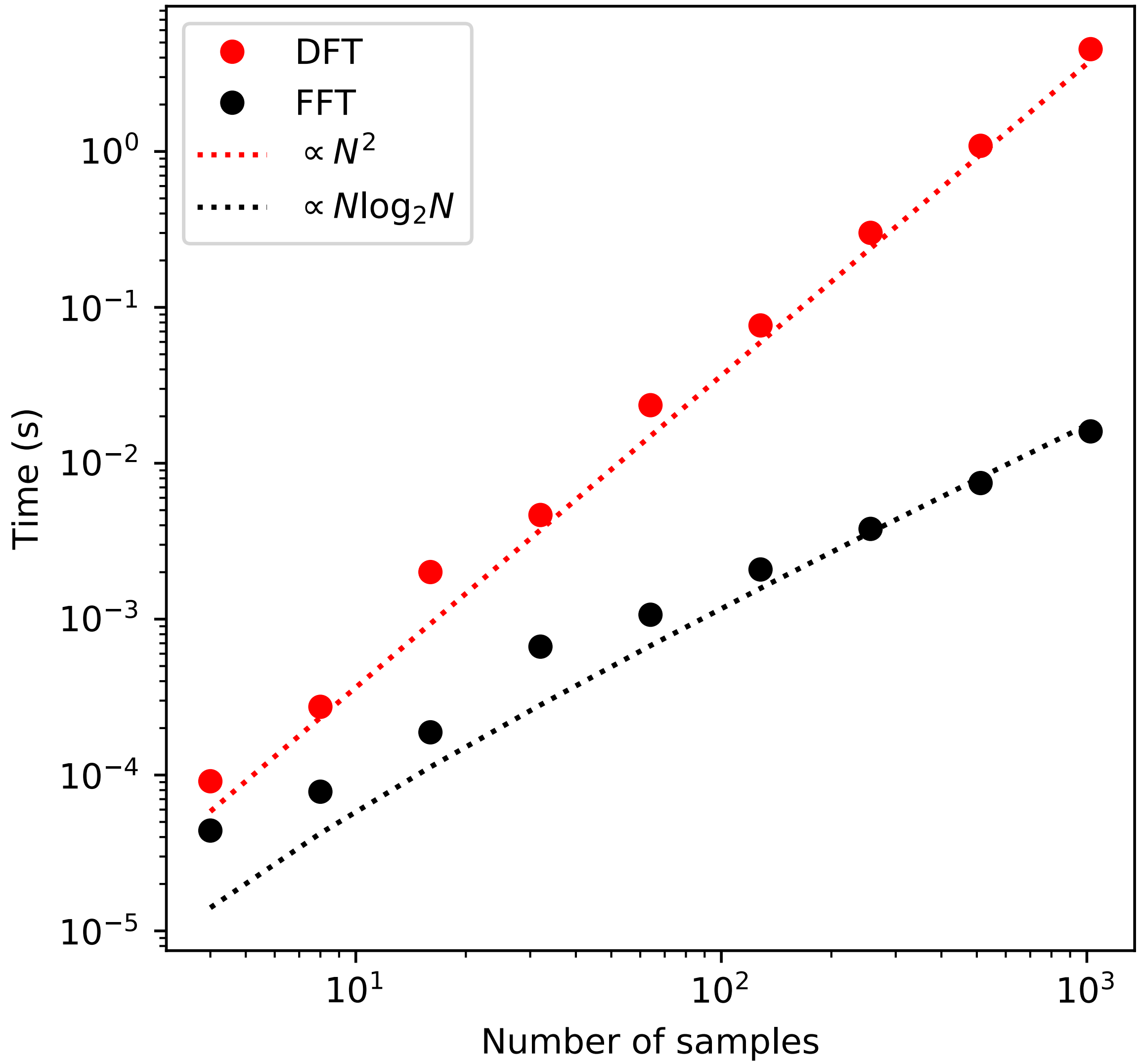
$$F_k = F_k^{\text{even}} + e^{-i2\pi k/N} F_k^{\text{odd}}$$

- 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

Today's lecture:
FFTs and curve fitting

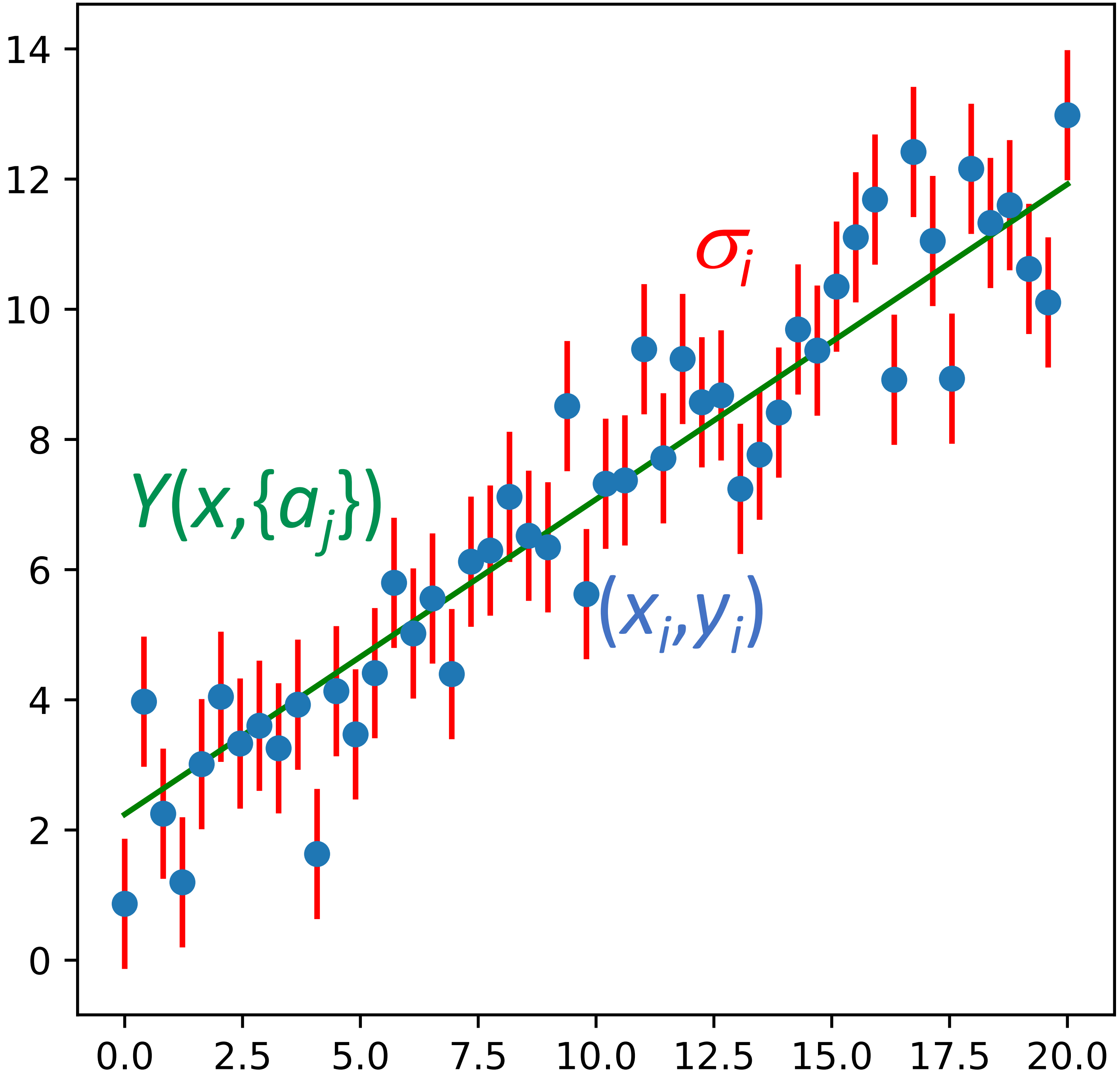
- More on Fourier Transforms
  - 2D FT
  - Cosine transformation
  - FFTs

- Curve fitting

Fitting data

- We have discussed interpolation, now we'll talk about fitting
  - *Interpolation* seeks to fill in missing information in some small region of the whole dataset
  - *Fitting* a function to the data seeks to produce a model (guided by physical intuition) so you can learn more about the global behavior of your data
- Goal is to understand data by finding a simple function that best represents the data
  - Previous discussion on linear algebra and root finding comes into play
- We will follow Garcia (Sec. 5.1)
  - Big topic, we'll just look at the basics

Notation



General theory of fitting

- We have a dataset of N points (x_i, y_i)
- Would like to "fit" this dataset to a function $Y(x,\{a_j\})$
  - $\{a_j\}$ is a set of M adjustable parameters
  - Find the value of these parameters that minimizes the distance between data points and curve: $$\Delta_i = Y(x_i, \{a_j\}) - y_i$$
- Curve-fitting criteria: Minimize the sum of the squares

$$D(\{a_j\}) = \sum_{i=0}^{N-1} \Delta_i^2 = \sum_{i=0}^{N-1} [Y(x_i, \{a_j\}) - y_i]^2$$

- "Least squares fit"
  - Not the only way, but the most common

General theory of fitting

- Often data points have estimated error bars/confidence intervals σ_i
- Modify fit criterion to give less weight to points with the most error

$$\chi^2(\{a_j\}) = \sum_{i=0}^{N-1} \left(\frac{\Delta_i}{\sigma_i}\right)^2 = \sum_{i=0}^{N-1} \frac{[Y(x_i, \{a_j\}) - y_i]^2}{\sigma_i^2}$$

- χ^2 most used fitting function
  - Errors have a Gaussian distribution

- We will not discuss “validation” of curve fitted to data
  - i.e., probability that the data is described by a given curve

Linear regression

- Now that we have criteria for a good fit, we need to find $\{a_i\}$
- First consider the simplest example: fitting data with a straight line

$$Y(x_i, \{a_0, a_1\}) = a_0 + a_1 x$$

- Such that χ^2 is minimized:

$$\chi^2(a_0, a_1) = \sum_{i=0}^{N-1} \frac{[a_0 + a_1 x_i - y_i]^2}{\sigma_i^2}$$

Linear regression: Finding coefficients

- Minimize χ^2 with respect to coefficients:

$$\frac{\partial \chi^2}{\partial a_0} = 2\sum_{i=0}^{N-1} \frac{a_0 + a_1 x_i - y_i}{\sigma_i^2} = 0, \qquad \frac{\partial \chi^2}{\partial a_0} = 2\sum_{i=0}^{N-1} x_i \frac{a_0 + a_1 x_i - y_i}{\sigma_i^2} = 0$$

- We can write as:

$$a_0 S + a_1 \Sigma_x - \Sigma_y = 0, \qquad a_0 \Sigma_x + a_1 \Sigma_{x^2} - \Sigma_{xy} = 0$$

- Where coefficients are known:

$$S \equiv \sum_{i=0}^{N-1} \frac{1}{\sigma_i^2}, \quad \Sigma_x \equiv \sum_{i=0}^{N-1} \frac{x_i}{\sigma_i^2}, \quad \Sigma_y \equiv \sum_{i=0}^{N-1} \frac{y_i}{\sigma_i^2}, \quad \Sigma_{x^2} \equiv \sum_{i=0}^{N-1} \frac{x_i^2}{\sigma_i^2}, \quad \Sigma_{xy} \equiv \sum_{i=0}^{N-1} \frac{x_i y_i}{\sigma_i^2}$$

Linear regression: Finding coefficients

- Solving for a_0 and a_1:

$$a_0 = \frac{\Sigma_y \Sigma_{x^2} - \Sigma_x \Sigma_{xy}}{S\Sigma_{x^2} - (\Sigma_x)^2}, \qquad a_1 = \frac{S\Sigma_{xy} - \Sigma_y \Sigma_x}{S\Sigma_{x^2} - (\Sigma_x)^2}$$

- Note that if σ_i is constant, it will cancel out
- Now let's define an error bar for the curve-fitting parameter a_j

$$\sigma_{a_j}^2 = \sum_{i=0}^{N-1} \left(\frac{\partial a_j}{\partial y_i} \right)^2 \sigma_i^2$$

  - See: https://en.wikipedia.org/wiki/Propagation_of_uncertainty

- For our linear case (after some algebra):

Both independent of y_i

$$\sigma_{a_0} = \sqrt{\frac{\Sigma_{x^2}}{S\Sigma_{x^2} - (\Sigma_x)^2}}, \qquad \sigma_{a_1} = \sqrt{\frac{S}{S\Sigma_{x^2} - (\Sigma_x)^2}}$$

Linear regression: Errors in coefficients

- If error bars are constant:

$$\sigma_{a_0} = \frac{\sigma_0}{\sqrt{N}}\sqrt{\frac{\langle x^2\rangle}{\langle x^2\rangle - \langle x\rangle^2}}, \qquad \sigma_{a_1} = \frac{\sigma_0}{\sqrt{N}}\sqrt{\frac{1}{\langle x^2\rangle - \langle x\rangle^2}}$$

variance

- Where:

$$\langle x\rangle = \frac{1}{N}\sum_{i=0}^{N-1} x_i, \quad \langle x^2\rangle = \frac{1}{N}\sum_{i=0}^{N-1} x_i^2$$

- If data does not have error bars, we can estimate ω_0 from the sample variance (https://en.wikipedia.org/wiki/Variance)

Sample std deviation

$$\sigma_0 \simeq s^2 = \frac{1}{N-2}\sum_{i=0}^{N-1}\left[y_i - (a_0 + a_1 x_i)\right]^2$$

N-2 since already extracted a0 and a1 from data

Nonlinear regression (with two variables)

- We have been discussing fitting a linear function, but many nonlinear curve-fitting problems can be transformed into linear problems

- Examples: $Z(x, \{\alpha, \beta\}) = \alpha e^{\beta x}$

- Rewrite with: $\ln Z = Y, \quad \ln \alpha = a_0, \quad \beta = a_1$

- Result: $Y = a_0 + a_1 x$

General least squares fit

- No analytic solution to general least squares problem, but can solve numerically
- Generalize to functions of the form:

$$Y(x_i, \{a_j\}) = a_0 Y_0(x) + a_1 Y_1(x) + \cdots + a_{M-1} Y_{M-1}(x) = \sum_{j=0}^{M-1} a_j Y_j(x)$$

- Now minimize χ^2: $$\frac{\partial \chi^2}{\partial \{a_j\}} = \frac{\partial}{\partial \{a_j\}} \sum_{i=0}^{N-1} \frac{1}{\sigma_i^2} \left[\sum_{k=0}^{M-1} a_k Y_k(x_i) - y_i \right]^2 = 0$$

$$= \sum_{i=0}^{N-1} \frac{Y_j(x_i)}{\sigma_i^2} \left[\sum_{k=0}^{M-1} a_k Y_k(x_i) - y_i \right] = 0$$

General least-squares fit

- From previous slide, we have:

$$\sum_{i=0}^{N-1} \sum_{k=0}^{M-1} \frac{Y_j(x_i) Y_k(x_i)}{\sigma_i^2} a_k = \sum_{i=0}^{N-1} \frac{Y_j(x_i) y_i}{\sigma_i^2}$$

- Set of *j* equations known as normal equations of the least-squares problem (*Y*'s may be nonlinear, but linear in *a*'s)
- Define design matrix with elements $A_{ij} = Y_j(x_i)/\sigma_i$:

$$\mathbf{A} = \begin{bmatrix} \frac{Y_0(x_0)}{\sigma_0} & \frac{Y_1(x_0)}{\sigma_0} & \cdots \\ \frac{Y_0(x_1)}{\sigma_1} & \frac{Y_1(x_1)}{\sigma_1} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

- Only depends on independent variables (not y_i)

General least-squares fit

- With design matrix, we can rewrite: $$\sum_{i=0}^{N-1}\sum_{k=0}^{M-1}\frac{Y_j(x_i)Y_k(x_i)}{\sigma_i^2}a_k = \sum_{i=0}^{N-1}\frac{Y_j(x_i)y_i}{\sigma_i^2}$$

- As: $$\sum_{i=0}^{N-1}\sum_{k=0}^{M-1}A_{ij}A_{ik}a_k = \sum_{i=0}^{N-1}A_{ij}\frac{y_i}{\sigma_i} \implies (\mathbf{A}^{\mathrm{T}}\mathbf{A})\mathbf{a} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$
  - Where $b_i = y_i/\sigma_i$

- Thus: $\mathbf{a} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$

- Or, we can solve for **a** via Gaussian elimination

Goodness of fit

- Usually, we have *N* >> *M*, the number of data points is much greater than the number of fitting variables
- Given the error bars, how likely is it that the curve actually describes the data?
- Rule of thumb: If the fit is good, on average the difference should be approximately equal to the error bars

$$|y_i - Y(x_i)| \simeq \sigma_i$$

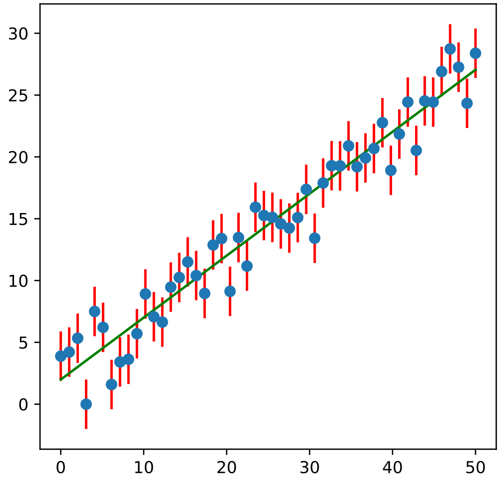
- Plugging in gives χ^2 equal to N. Since we know we can have a perfect fit for M=N, we postulate:

$$\chi^2 \simeq N - M$$

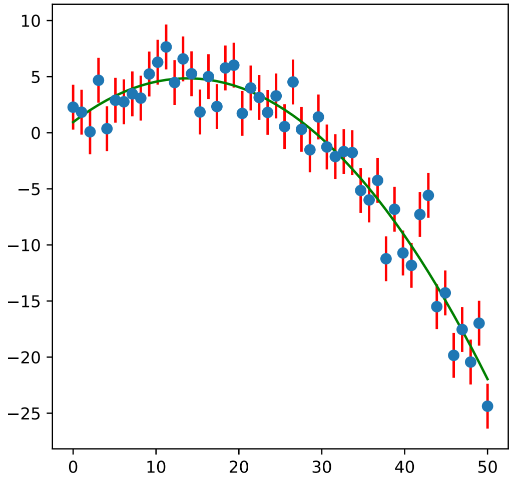
- If $\chi^2 \gg N - M$, probably not an appropriate function (or too small error bars
- If $\chi^2 \ll N - M$, fit is too good, error bars may be too large

Least squares fitting example:

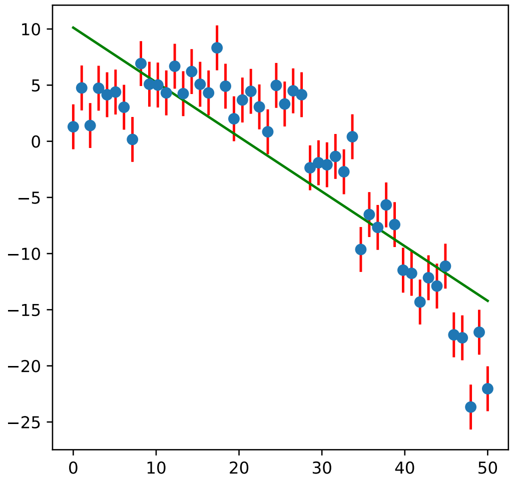
Linear regression, linear function



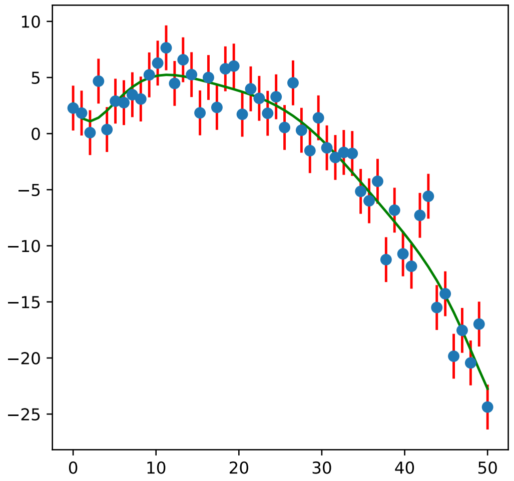
Polynomial regression (order 2), quadratic function



Linear regression, quadratic function



Polynomial regression (order 10), quadratic function



Comments on general least squares

- In the example, we used polynomials as our functions, but can use linear combinations of any functions we would like

- We choose functions strategically to get the best least squares fit
  - Often choosing orthogonal basis functions in the range of the fit will produce better fits

- The matrix $\mathbf{A}^T\mathbf{A}$ is notoriously ill conditioned especially for increased number of basis functions
  - Gaussian substitution will have problems solving (numpy solve uses singular-value decomposition)

- Procedure can be generalized if we also have errors in x

Nonlinear least-squares fitting

- Even in the polynomial case, we were using linear combinations of functions
- We can also directly fit a function whose parameters enter nonlinearly
- Consider the function: $f(a_0, a_1) = a_0 e^{a_1 x}$
- Want to minimize: $Q \equiv \sum_{i=1}^{N} (y_i - a_0 e^{a_1 x_i})^2$
- Take derivatives:

$$f_0 = \frac{\partial Q}{\partial a_0} = \sum_{i=1}^{N} e^{a_1 x_i} (a_0 e^{a_1 x_i} - y_i) = 0,$$

$$f_1 = \frac{\partial Q}{\partial a_1} = \sum_{i=1}^{N} x_i e^{a_1 x_i} (a_0 e^{a_1 x_i} - y_i) = 0$$

Nonlinear least-squares fitting

- Produces a nonlinear system—we can use the multivariate root-finding techniques we learned earlier:
  - Compute the Jacobian
  - Take an initial guess for unknown coefficients
  - Use Newton-Raphson techniques to compute the correction:

$$\mathbf{a}_1 = \mathbf{a}_0 - \mathbf{J}^{-1}\mathbf{f}$$

  - Iterate

- Can be very difficult to converge, and highly dependent on the initial guess

Fitting packages

- Fitting is a very sensitive procedure—especially for nonlinear cases
- Lots of minimization packages exist that offer robust fitting procedures

- MINUIT2: the standard package in high-energy physics (Python version: PyMinuit and Iminuit)
- MINPACK: Fortran library for solving least squares problems—this is what is used under the hood for the built in SciPy least squares routine
    - http://www.netlib.org/minpack/
- SciPy optimize: https://docs.scipy.org/doc/scipy/reference/optimize.html

After class tasks

- Homework 2 due today
- Homework 3 will be posted today or tomorrow

- Readings
  - FFTs:
    - Newman Ch. 7
    - https://en.wikipedia.org/wiki/Discrete_Fourier_transform
  - Linear regression:
    - Wikipedia page on varience
    - Wikipedia page on propagation of errors
    - Garcia Sec. 5.1