Fitting Data

- We get experimental/observational data as a sequence of times (or positions) and associate values
  - N points: \((x_i, y_i)\)
  - Often we have errors in our measurements at each of these values: \(\sigma_i\) for each \(y_i\)

- To understand the trends represented in our data, we want to find a simple functional form that best represents the data—this is the fitting problem
  - We'll follow the discussion in Garcia to get a basic feel for the problem (the discussion in Numerical Recipes is quite similar too)

- This is a big topic—we'll just look at the basics here
  - We'll see that our previous work on linear algebra and root finding comes back into play...
Fitting Data

- We want to fit our data to a function: \( Y(x, \{a_j\}) \)
  - Here, the \( a_j \) are a set of parameters that we can adjust
  - We want to find the optimal set of \( a_j \) that make \( Y \) best represent our data
- The distance between a point and the representative curve is
  \[
  \Delta_i = Y(x_i, \{a_j\}) - y_i
  \]
  - Least squares fit minimizes the sum of the squares of all these errors
  - With error bars, we weight each distance error by the uncertainty in that measurement, giving:
  \[
  \chi^2(\{a_j\}) = \sum_{i=1}^{N} \left( \frac{\Delta_i}{\sigma_i} \right)^2
  \]
  This is what we minimize
Linear Regression

- Linear regression: use a line as our model:
  \[ Y = a_1 + a_2 x \]

  - Our fit appears as:
    \[
    \chi^2(a_1, a_2) = \sum_{i=1}^{N} \left( \frac{(a_1 + a_2 x_i - y_i)^2}{\sigma_i^2} \right)
    \]

  - Finding the parameters requires minimization → generates a linear system to solve
Linear Regression

- Minimization: derivative of $\chi^2$ with respect to all parameters is zero:

$$\frac{\partial \chi^2}{\partial a_1} = 2 \sum_{i=1}^{N} \frac{a_1 + a_2 x_i - y_i}{\sigma_i^2} = 0$$

$$\frac{\partial \chi^2}{\partial a_2} = 2 \sum_{i=1}^{N} \frac{a_1 + a_2 x_i - y_i}{\sigma_i^2} x_i = 0$$

- Define:

$$S = \sum_{i=1}^{N} \frac{1}{\sigma_i^2}$$

$$\xi_1 = \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}$$

$$\xi_2 = \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}$$

$$\eta = \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2}$$

$$\mu = \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2}$$
Linear Regression

- We then have a linear system: 2 equations + 2 unknowns

\[
a_1 S + a_2 \xi_1 - \eta = 0
\]
\[
a_1 \xi_1 + a_2 \xi_2 - \mu = 0
\]

- We can solve this analytically

\[
a_1 = \frac{\eta \xi_2 - \mu \xi_1}{\xi_2 S - \xi_1^2}
\]
\[
a_2 = \frac{S \mu - \xi_1 \eta}{\xi_2 S - \xi_1^2}
\]
Goodness of the Fit

- Typically, if $M$ is the number of parameters (2 for linear), then $N \gg M$
  - Average pointwise error should be $|y_i - Y(x_i)| \sim \sigma_i$
  - Number of degrees of freedom is $N - M$
    - i.e. larger $M$ makes it easier to fit all the points
    - See discussion in Numerical Recipes for more details and limitations
  - Putting these ideas into the $\chi^2$ expression suggests that we consider
    \[
    \frac{\chi^2}{N - M}
    \]
    - If this is $< 1$, then the fit is good
    - But watch out, $\ll 1$ may also mean our errors were too large to begin with, we used too many parameters, ...
Generating Our Experimental Data

- We perturb a desired functional form with random number
  - The random numbers sample a Gaussian-normalized distribution
    - `numpy.random.randn()` in python

\[ y(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-x^2/(2\sigma^2)} \]

Gaussian-normalized distribution matches our expectation of the behavior of experimental error
Started with $y(x) = 10 + 3x$
This has a $\chi^2/(N-2) = 0.85$
Ex: Linear Fit

Started with $y(x) = 2 + 1.5x - 0.02x^2$
This has a $\chi^2/(N-2) = 3.7$

Let's look at the code and see how the $\chi^2$ varies as we play with the $\sigma$s

code: linear-regression.py
Extending Utility of Linear Fitting

- Sometimes a simple transform can make the data look linear
  - E.g. for fitting to $Z(t) = \alpha t^\beta$, take
  - $Y = \ln Z$, $x = \ln t$, $a_1 = \ln \alpha$, $a_2 = \beta$
  - See NR and Garcia for more examples
General Linear Least Squares

- The general linear least squares problem does not have a general analytic solution
  - But our linear algebra techniques come into play to save the day
  - Again, Garcia and Numerical Recipes provide a good discussion here
- We want to fit to

\[ Y(x; \{a_j\}) = \sum_{j=1}^{M} a_j Y_j(x) \]

- Note that the \( Y \)s may be nonlinear but we are still linear in the \( a \)s
- Here, \( Y_j \) are our basis set—they can be \( x^i \) in which case we fit to a general polynomial
Again, we minimize our $\chi^2$

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

- Bringing the derivative inside the sums and simplifying, we have:

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

- Note that the only index not summed is $j$
- This is $M$ equations to solve
General Linear Least Squares

- We introduce the design matrix \((N \times M)\):

\[
A_{ij} = \frac{Y_j(x_i)}{\sigma_i}
\]

- Our system then becomes (see NR or Garcia):

\[
\sum_{i=1}^{N} \sum_{k=1}^{M} A_{ij} A_{ik} a_k = \sum_{i=1}^{N} A_{ij} \frac{y_i}{\sigma_i}
\]

- Looking at which indices contract, we have:

\[
(A^T A)a = A^T b
\]

- This is a linear system, consisting of an \(M \times M\) matrix
- We can solve for the fitting parameters using Gaussian elimination
General Linear Least Squares

- $M=3$ (quadratic) fit to data
  - Data generated from $y(x) = 2 + 1.5x - 0.02x^2$ with Gaussian normal errors
  - $\chi^2/(N-M) = 0.81$
  - Coefficients:

  \[
  a = \\
  \begin{bmatrix}
  3.0835124 \\
  1.50175118 \\
  -0.02026005
  \end{bmatrix}
  \]

  code: general-linear.py
General Linear Least Squares

- M=10 (quadratic) fit to data
  - Same data
  - $\chi^2/(N-M)=0.91$
  - Coefficients:

```python
a = [ 2.27488631e+00
     8.29616711e-01
     2.89014125e-01
    -3.65205170e-02
     1.97413575e-03
    -5.80360431e-05
     9.88242216e-07
    -9.74442949e-09
     5.16759888e-11
    -1.14121212e-13]
```

Look how small some of the terms are!

code: general-linear.py
Other Basis Functions

- Instead of using $1, x, x^2, x^3, ...$
  - Use Legendre Polynomials
  - M-degree fit should be identical to what we already did, but coefficients will differ
  - Coefficients:

```
[  2.37164216e+00
  8.07646029e-01
  1.93810011e-01
 -1.46343131e-02
  4.51547675e-04
 -7.37178812e-06
  6.84575548e-08
 -3.63443852e-10
  1.02791589e-12
 -1.20179031e-15]
```

Same polynomial, but what did that get us?
Condition Number

- The matrix $A^T A$ is notoriously ill-conditioned
  - For our examples above
    - M=3 fit: $\text{cond}(A^T A) = 1.70 \times 10^8$
    - M=10 fit: $\text{cond}(A^T A) = 1.93 \times 10^{33}$
    - M=10 fit w/ Legendre polynomials: $\text{cond}(A^T A) = 9.29 \times 10^{37}$
- These are large condition numbers—in fact Gaussian elimination would have trouble with these
  - `numpy.linalg.solve()` uses `singular-value decomposition`
- Legendre polynomials made things worse!
  - But recall, the special thing about Legendre polynomials is that they are orthogonal in $[-1, 1]$
Condition Number

- On [-1,1], using the simple $x^j$ and Legendre polynomials will again give the same resulting polynomial, but:
  - M=10, simple polynomials: $\text{cond}(A^T A) = 1.45 \times 10^6$
  - M=10, Legendre polynomials: $\text{cond}(A^T A) = 17.8$

- Generally speaking: using orthogonal basis functions in your interval makes the problem better posed (condition number is much smaller)
  - You can create polynomial basis function on any interval by doing the inner products in your code (see Yakowitz & Szidarovszky, for example)

code: general-linear.py
Errors in Both x and y

- Depending on the experiment, you may have errors in the dependent variable
  - For linear regression, our function to minimize becomes:
    \[
    \chi^2(a_1, a_2) = \sum_{i=1}^{N} \frac{(a_1 + a_2 x_i - y_i)^2}{\sigma_{y,i}^2 + a_2^2 \sigma_{x,i}^2}
    \]
    - Denominator is the total variance of the linear combination we are minimizing:
      \[
      \text{Var}(a_1 + a_2 x_i - y_i) = \text{Var}(a_2 x_i - y_i) = a_2^2 \text{Var}(x_i) + \text{Var}(y_i) = a_2^2 \sigma_{x,i}^2 + \sigma_{y,i}^2
      \]
      (think about propagation of errors)

- We cannot solve analytically for the parameters, but we can use our root finding techniques on this.
  - See NR and references therein for more details
Estimating Errors in the Fit Parameters

- We can use propagation of errors to estimate the uncertainty in our fit parameters

\[ \sigma^2_{a_j} = \sum_{i=1}^{N} \left( \frac{\partial a_j}{\partial y_i} \right)^2 \sigma_i^2 \]

- For linear regression, this gives:

\[ \sigma^2_{a_1} = \frac{\xi_2}{S\xi_2 - \xi_1^2} \quad \sigma^2_{a_2} = \frac{S}{S\xi_2 - \xi_1^2} \]

(blackboard derivation...)

- For the general linear least squares problem, we find:

\[ \sigma_{a_j} = \sqrt{C_{jj}} \quad C = (A^T A)^{-1} \]

(see Numerical Recipes for a good derivation)
Estimating Errors in the Fit Parameters

- Linear fit with associate parameter errors:

  \[ \text{reduced chisq} = 1.05378308895 \]
  \[ a_1 = 25.161505 \pm 7.759730 \]
  \[ a_2 = 2.768434 \pm 0.133549 \]
General Non-linear Fitting
(Yakowitz and Szidarovszky)

- Consider fitting directly to a function where the parameters enter non-linearly:

\[ f(a_0, a_1) = a_0 e^{a_1 x} \]

- We want to minimize

\[ Q = \sum_{i=1}^{N} (y_i - a_0 e^{a_1 x_i})^2 \]

- Set the derivatives to zero:

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

\[ f_1 = \frac{\partial Q}{\partial a_1} = \sum_{i=1}^{N} x_i e^{a_1 x_i} (a_0 e^{a_1 x_i} - y_i) = 0 \]
General Non-linear Fitting
(Yakowitz and Szidarovszky)

- This is a nonlinear system—we can use the multivariate root-finding techniques we learned earlier
  - Compute the Jacobian
  - Take an initial guess: $a^{(0)}$
  - Use Newton-Raphson techniques to compute the correction:
    $$\delta a = -J^{-1}f$$
  - Iterate
- Note: this can be very sensitive to your initial guess.
General Non-linear Fitting
(Yakowitz and Szidarovszky)

- Data from $f(a_0, a_1) = a_0 e^{a_1 x}$
  - With $a_0 = 2.5, a_1 = 2/3$ with a Gaussian-sampled error
  - Initial guess is very sensitive—sometimes it diverges
Sometimes parameters can be redundant, leading to a singular matrix

- NR example: \( y(x) = ae^{-bx} + d \)
- Here there is functionally no difference between \( a \) and \( d \)
- The resulting matrix will be singular
Fitting is a very sensitive procedure—especially for nonlinear cases

Lots of minimization packages exist that offer robust fitting procedures—use them!

- **MINUIT**: the standard package in high-energy physics (and yes, there is a python version: PyMinuit)
- **MINPACK**: Fortran library for solving least squares problems—this is what is used under the hood for the built in SciPy least squares routine
- These packages often allow you to impose constraints on parameters, bounds, etc...

**SciPy optimize example...**