Understanding Experimental Data: Curve fitting (Regression)

Fredo Durand, most slides by Eric Grimson, John Guttag and Ana Bell

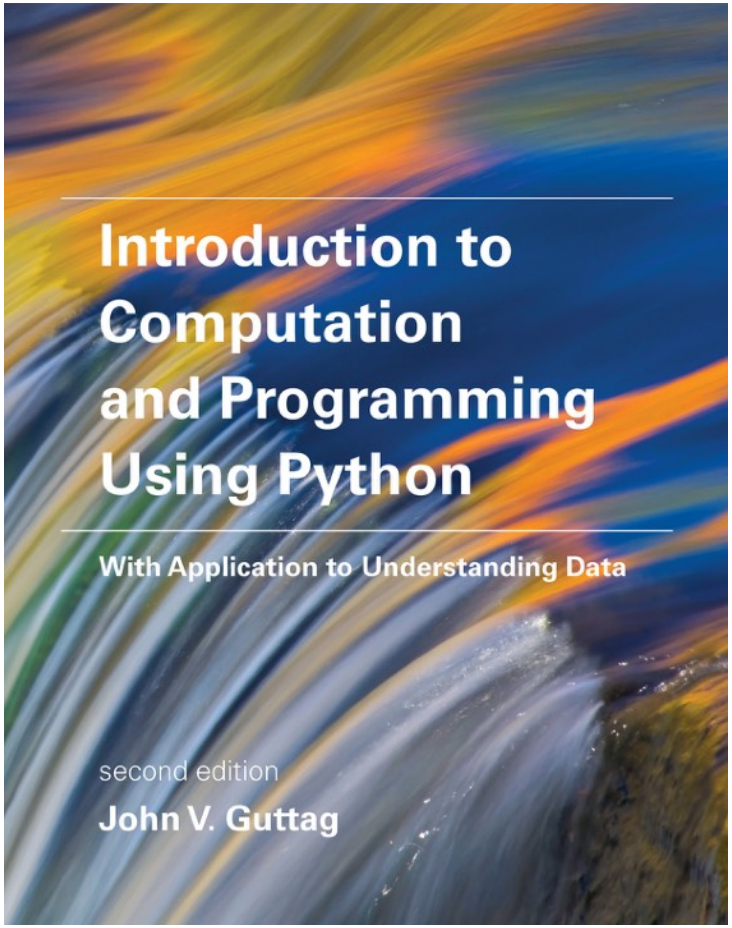
MIT Department of Electrical Engineering and Computer Science

Microquiz 3

- After this lecture

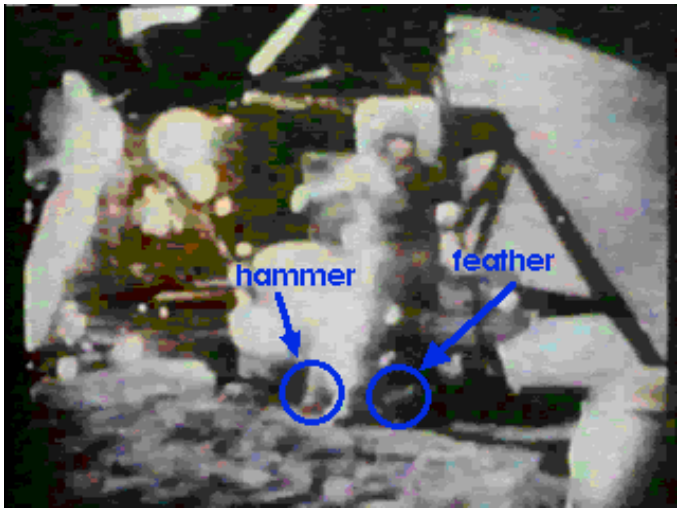
Assigned Reading

- Today:
  - Chapter 18
- Next lecture:
  - Chapter 22



Statistics Meets Experimental Science

- Conduct an experiment to gather (noisy) data
  - Physical (e.g., in a physics lab)
  - Social (e.g., questionnaires)
- Use theory to generate some questions about data
  - Physical (e.g., gravitational fields)
  - Social (e.g., people give inconsistent answers)
- Design a computation to help answer questions about data
- Can't afford a field trip to the moon, so consider, instead, a **spring**…

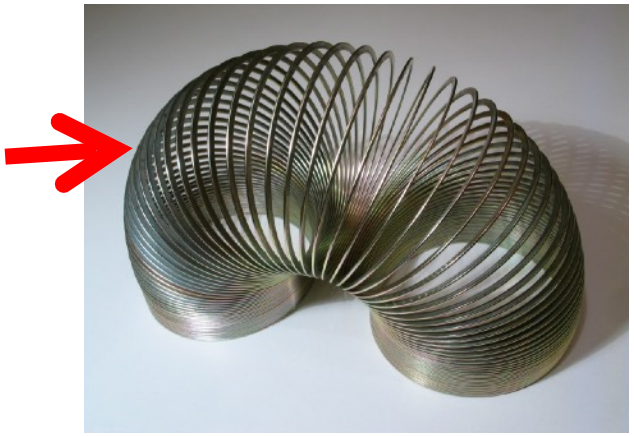


$$F_g = \frac{Gm_1m_2}{r^2}$$

Use equations and observations to determine global constants

This Kind of Spring

$$k \approx 35,000\, N/m$$



Hooke's Law ? Poll!

- Robert Hooke (1635-1703)
  - Discovered law of elasticity
    - Led to invention of balance spring, which led to first accurate watch
  - Huge believer in running experiments and then building models
    - *"The truth is, the Science of Nature has been already too long made only a work of the Brain and the Fancy: It is now high time that it should return to the plainness and soundness of Observations on material and obvious things."*

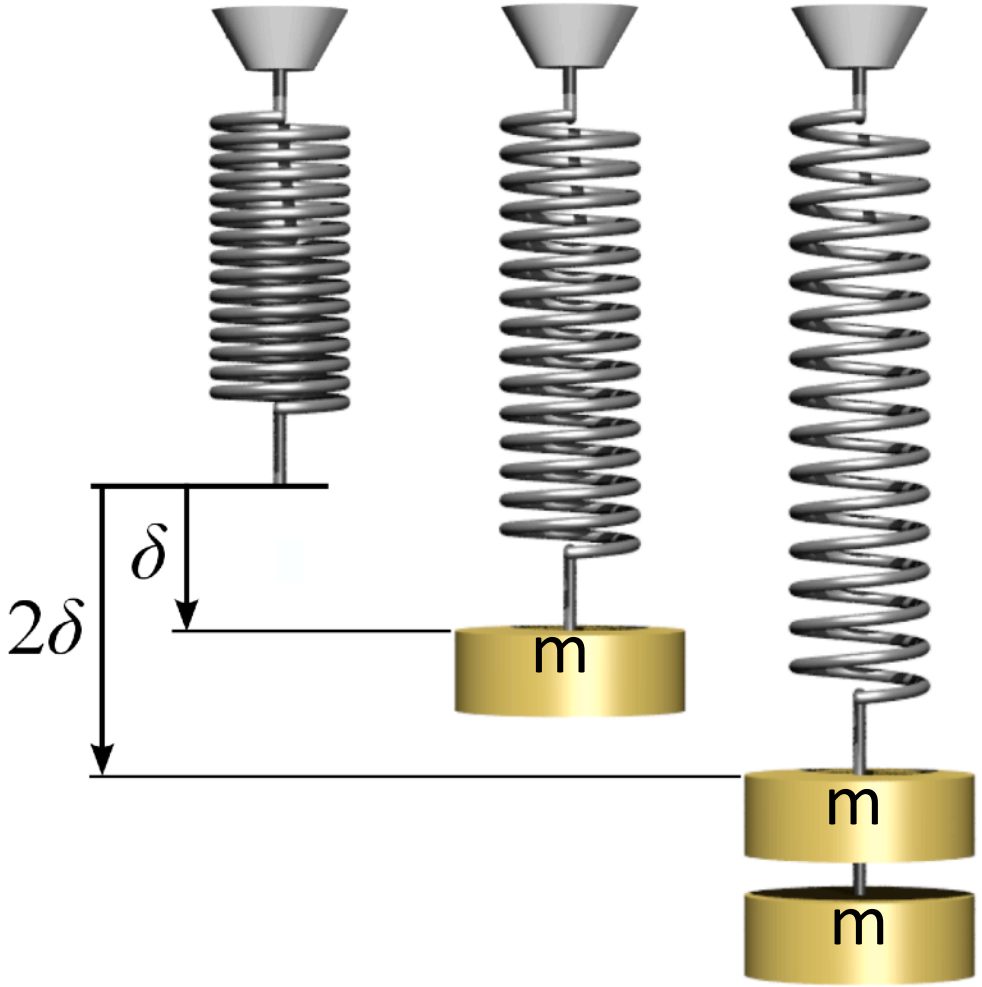
Hooke's Law

- F = -kd

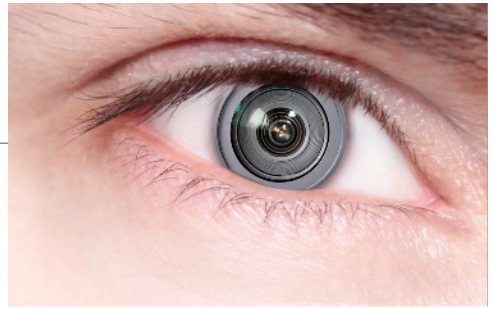
Why the '-'? Because deflection in opposite direction to force

Finding k: Some Data

Mass (kg)	Distance (m)
0.1	0.0865
0.15	0.1015
0.2	0.1106
0.25	0.1279
0.3	0.1892
0.35	0.2695
0.4	0.2888
0.45	0.2425
0.5	0.3465
0.55	0.3225
0.6	0.3764
0.65	0.4263
0.7	0.4562
0.75	0.4502
0.8	0.4499
0.85	0.4534
0.9	0.4416
0.95	0.4304
1.0	0.437



Taking a Look at the Data

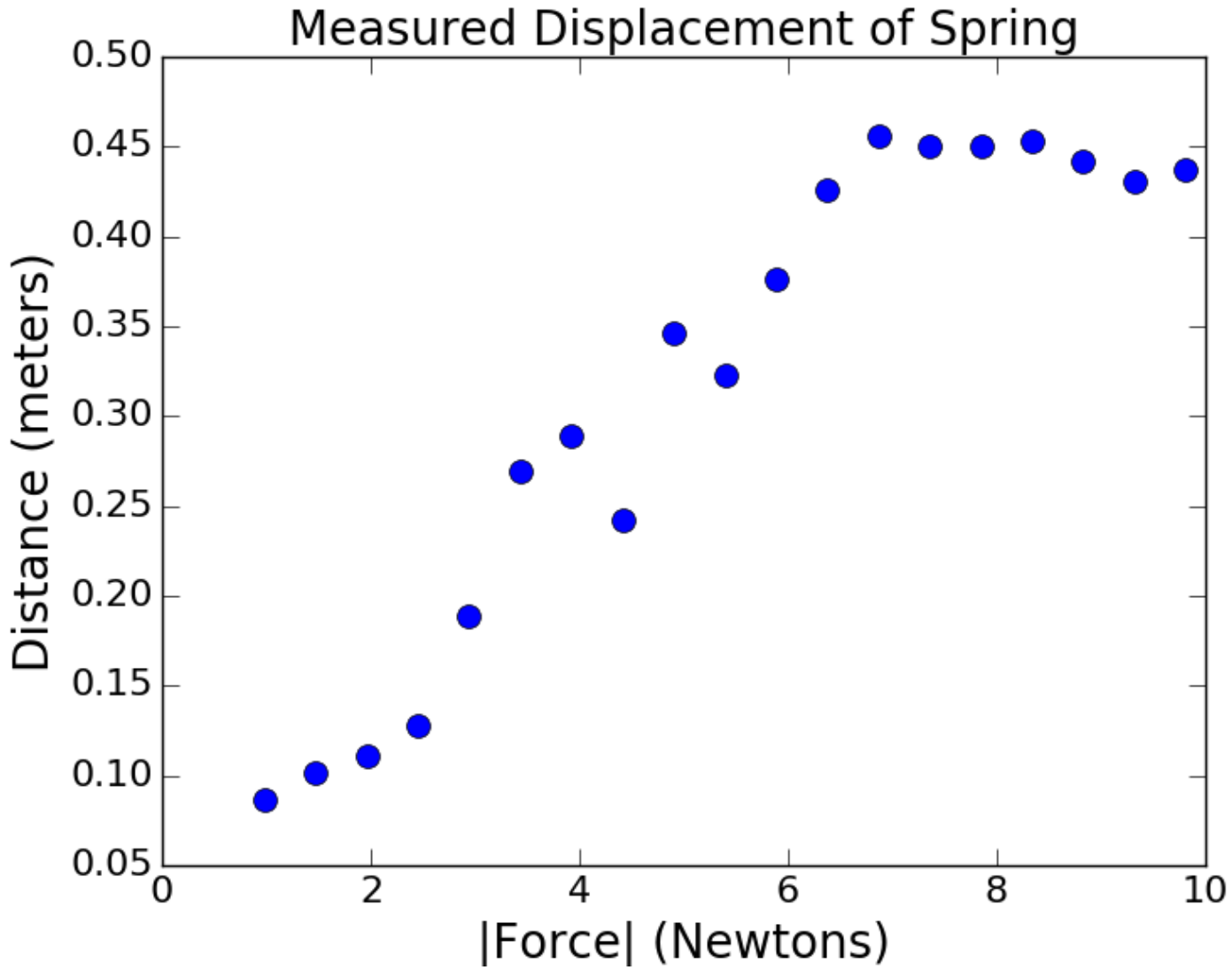


```
def plotData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals) #masses
    yVals = pylab.array(yVals) #distances/displacements
    xVals = xVals*9.81  #acc. due to gravity; forces
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured displacements')
    labelPlot()
```

A reminder/primer about python arrays:

- Converts a list into a linear data structure
- Can treat arrays algebraically; e.g., if a and b are arrays, then:
  - a*2 multiplies every element of a by 2
  - a + 3 adds 3 to every element of a
  - a – b subtracts each element of b from corresponding element of a
  - a*b multiplies each element of a by corresponding element of b

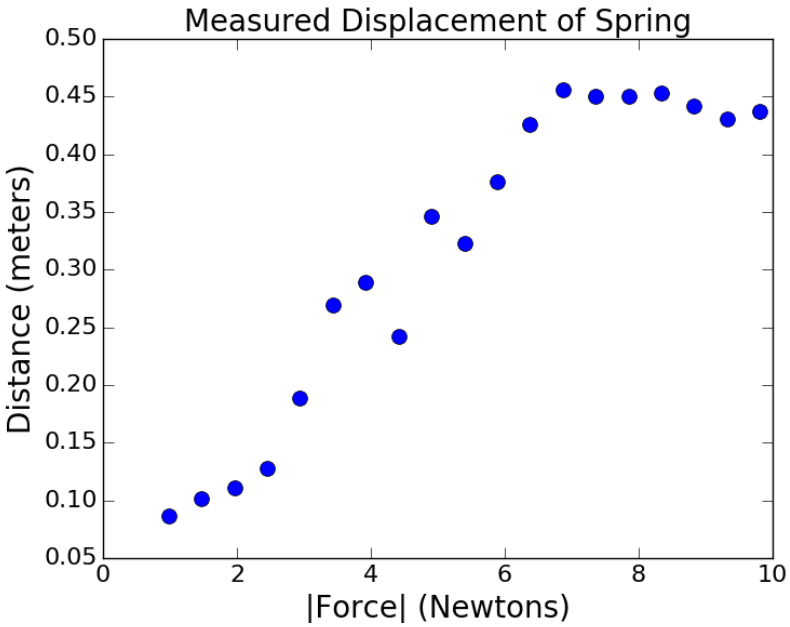
Taking a Look at the Data



Questions?

What Can We Do With This Data?

- We've run an experiment
- We can relate observations to measurements (distance d vs. force F)
- Our theory predicts a relationship between observations and measurements (F = -kd)
- Can we use these measurements to determine k and to validate model?

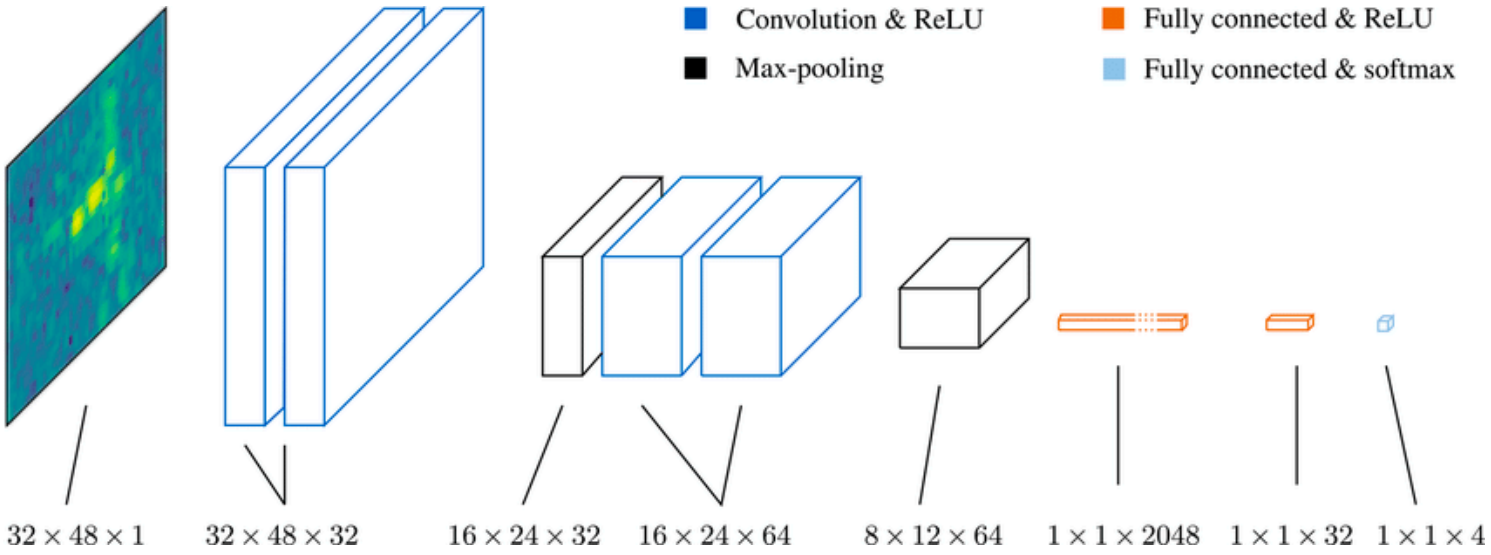


Fitting Curves to Data

- When we fit a curve (or ***model***) to a set of data, we are finding a fit that relates an *independent* variable (the mass or force) to an estimated value of a *dependent* variable (the distance)
- To decide how well a curve fits the data, we need a way to measure the goodness of the fit – called the **objective function (aka loss)**
- Once we define the objective function, we also need an algorithm to find the curve that minimizes it
  - Back to optimization!

Fitting Curves to Data

- Side note: also the main approach in AI/Machine learning these days



Least Squares Objective Function

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

- Why square? POLL!

Solving for Least Squares

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

- Simple example:
  - Use a degree-one polynomial, as model of data (best fitting line)
- Want to find

Solving for Least Squares

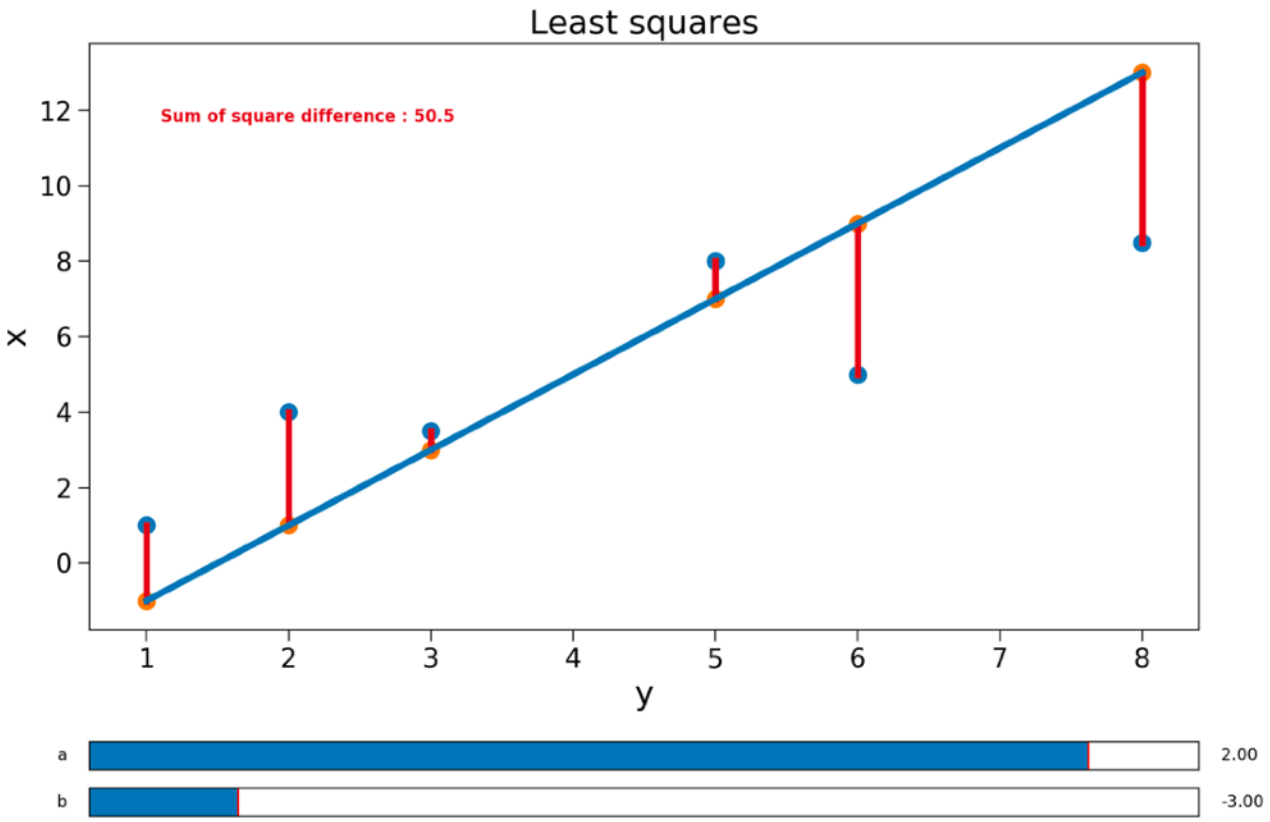
$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$

- Simple example:
  - Use a degree-one polynomial, $y = ax+b$, as model of data (best fitting line)
- Want to find values of a and b such that

$$\sum_{i=0}^{len(observed)-1} (observed[i] - a*x[i] - b)^2$$

is minimized, where *x[i]* is the i^{th} data point, and *observed[i]* is the corresponding measured value.

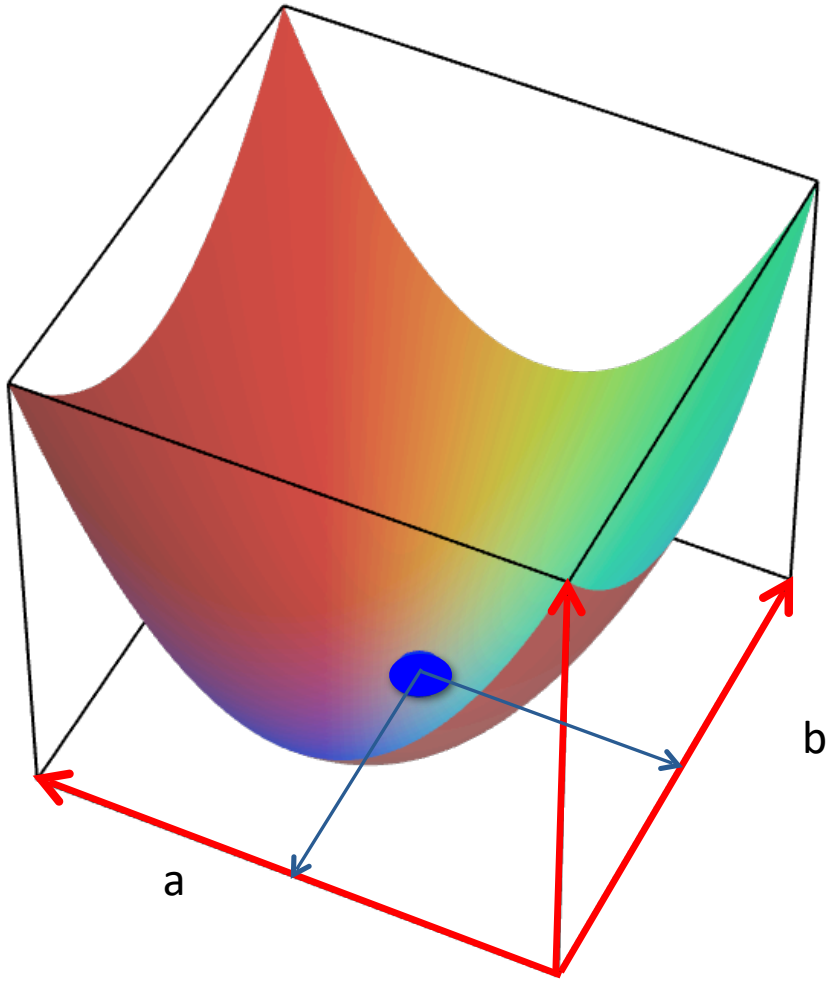
Demo: manual optimization



Questions?

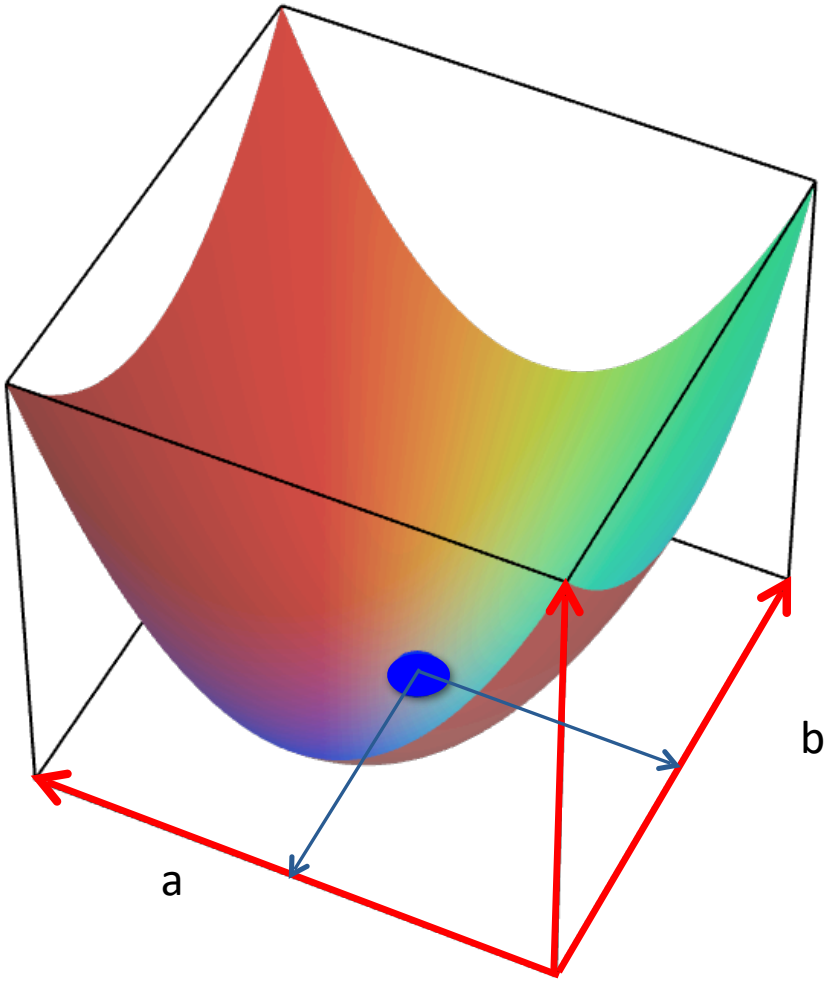
Finding the best curve (simplest case)

- Set of all possible lines can be represented by points in *a*-*b* space



Finding the best curve (simplest case)

- Set of all possible lines can be represented by points in *a*-*b* space
- Imagine a surface in this space, where height is value of the objective function
- Starting at any point on the surface, walk "downhill" **(step along gradient),** until you reach the "bottom"
- Corresponding point is **best** line to fit to data (for least squares optimization, space is convex)
- Can generalize to higher order models



Some nice properties of linear regression

- Objective surface using sum-of-squared-differences is differentiable
  - Means we can compute gradient direction analytically and use to efficiently compute next step in optimization
- Objective surface in this case has a global minimum
  - Means there is always a unique best fit

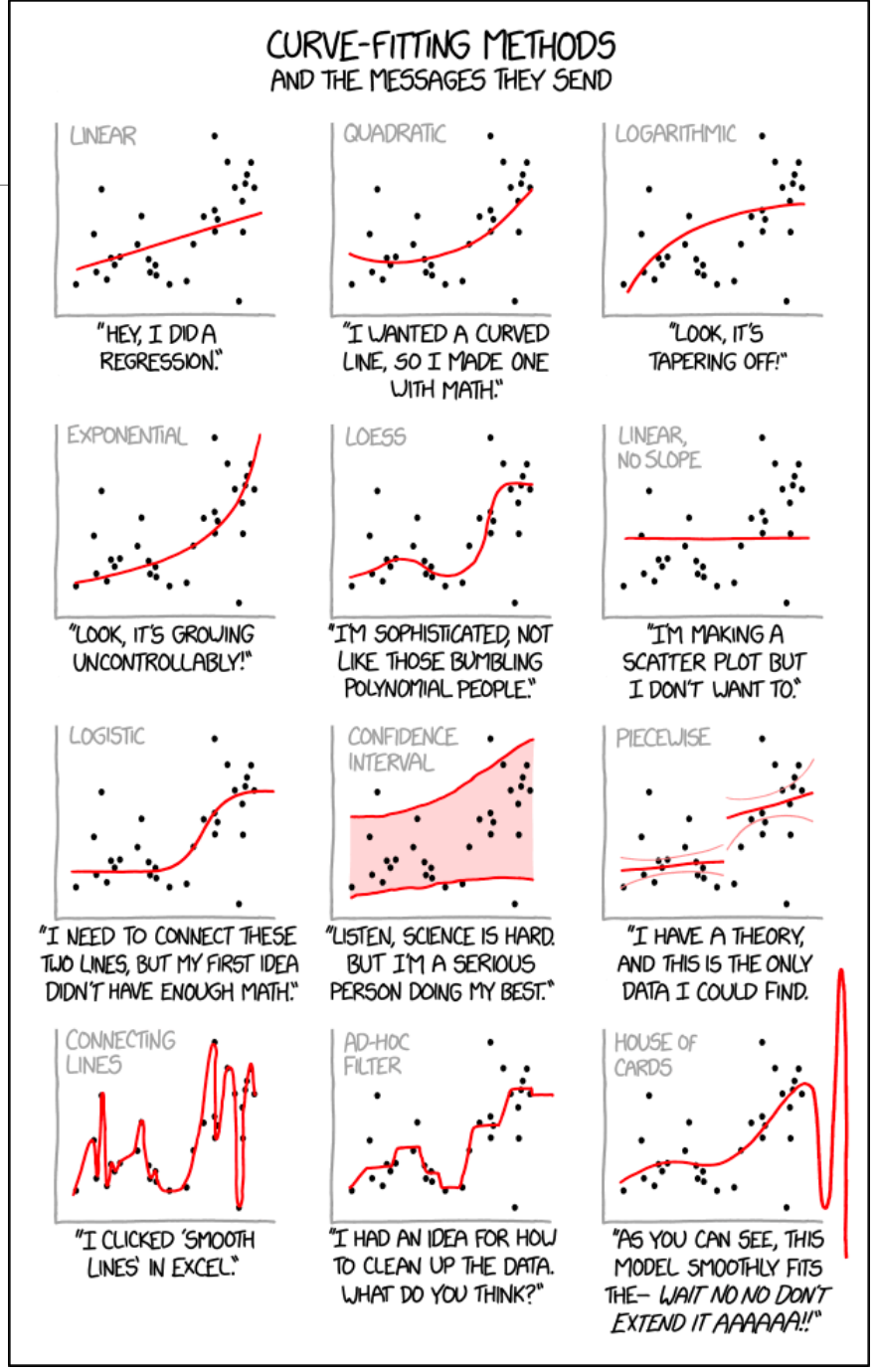
- Easy to solve for (linear system)
  - Minimum: We want the derivative to be 0
  - Derivative of a quadratic is linear

Derive it!

- Minimize $\sum_{i=0}^{len(observed)-1} (observed[i] - a * x[i] - b)^2$

Questions?

- https://xkcd.com/2048/



polyFit

- Good news is that pylab provides built in functions to find these polynomial fits
- pylab.polyfit(observedX, observedY, n)

finds coefficients of a polynomial, of degree n, that provides a best least squares fit for the observed data

- n = 1 – best line $y = ax + b$
- n = 2 – best parabola $y = ax^2 + bx + c$
- n = 3 – best cubic $y = ax^3 + bx^2 + cx + d$

Using polyfit

```
def fitData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    a,b = pylab.polyfit(xVals, yVals, 1)
    estYVals = a*pylab.array(xVals) + b
    print('a =', a, 'b =', b)
    pylab.plot(xVals, estYVals, 'r',
               label = 'Linear fit, k = '
               + str(round(1/a, 5)))
    pylab.legend(loc = 'best')
```

Using polyfit

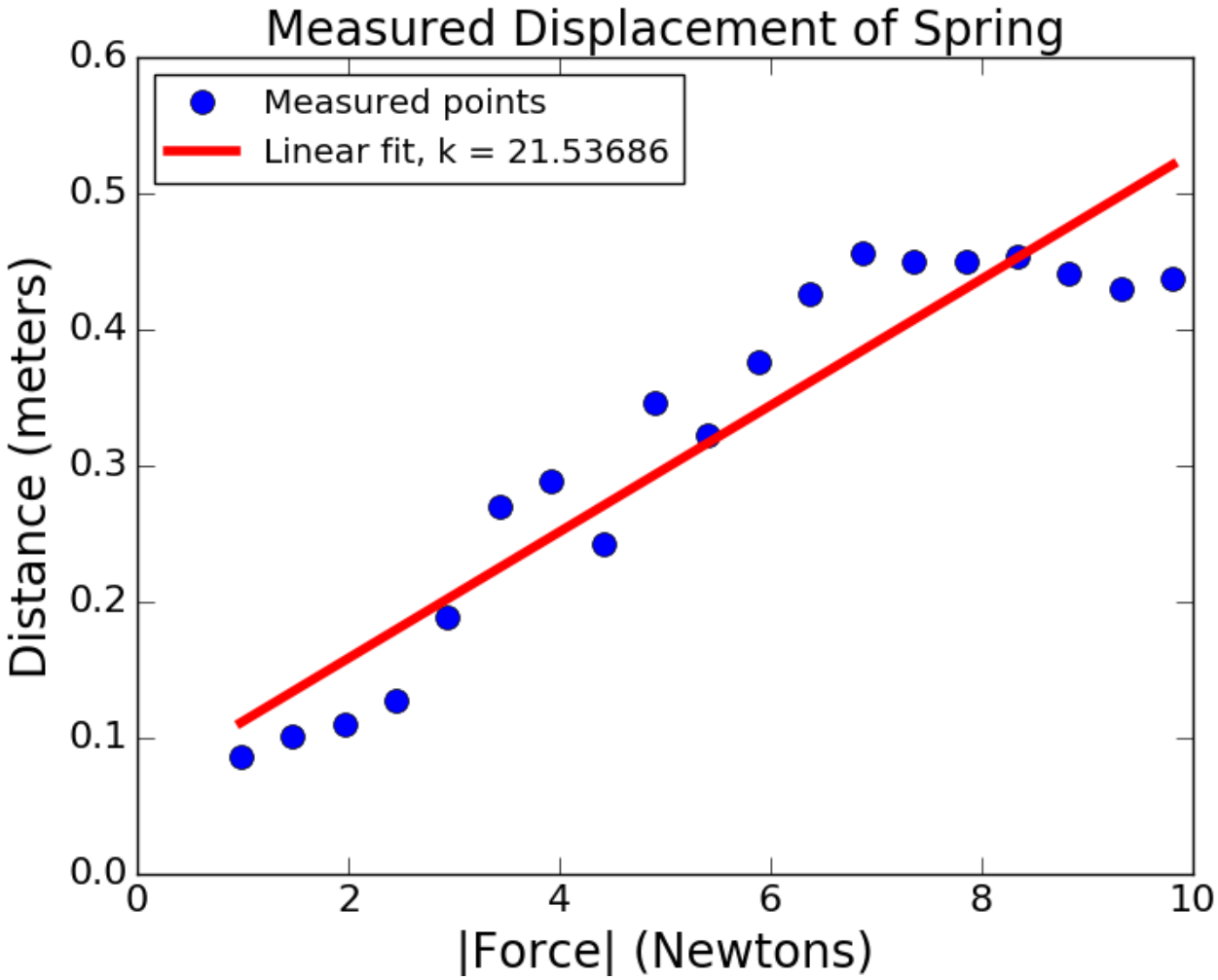
```
def fitData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    a,b = pylab.polyfit(xVals, yVals, 1)
    estYVals = a*pylab.array(xVals) + b
    print('a =', a, 'b =', b)
    pylab.plot(xVals, estYVals, 'r',
               label = 'Linear fit, k = '
               + str(round(1/a, 5)))
    pylab.legend(loc = 'best')
```

plotData

Note that conversion to array is redundant here

Remember Hooke:
F = kd
Here plotting d = aF
So k = 1/a

Visualizing the Fit



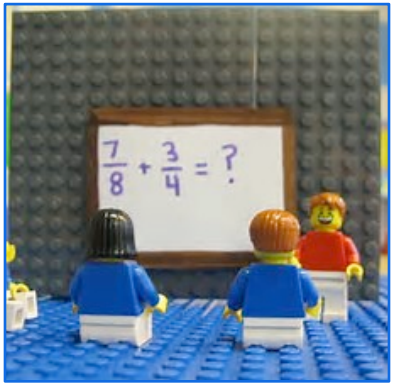
Version Using polyval

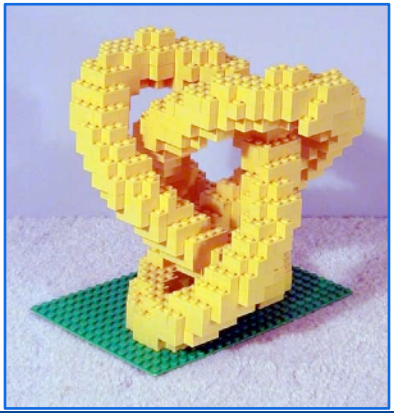
```
def fitData1(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    model = pylab.polyfit(xVals, yVals, 1)
    estYVals = pylab.polyval(model, xVals)
    pylab.plot(xVals, estYVals, 'r',
               label = 'Linear fit, k = '
               + str(round(1/model[0], 5)))
    pylab.legend(loc = 'best')
```

polyval will apply model to xVals for any order of model

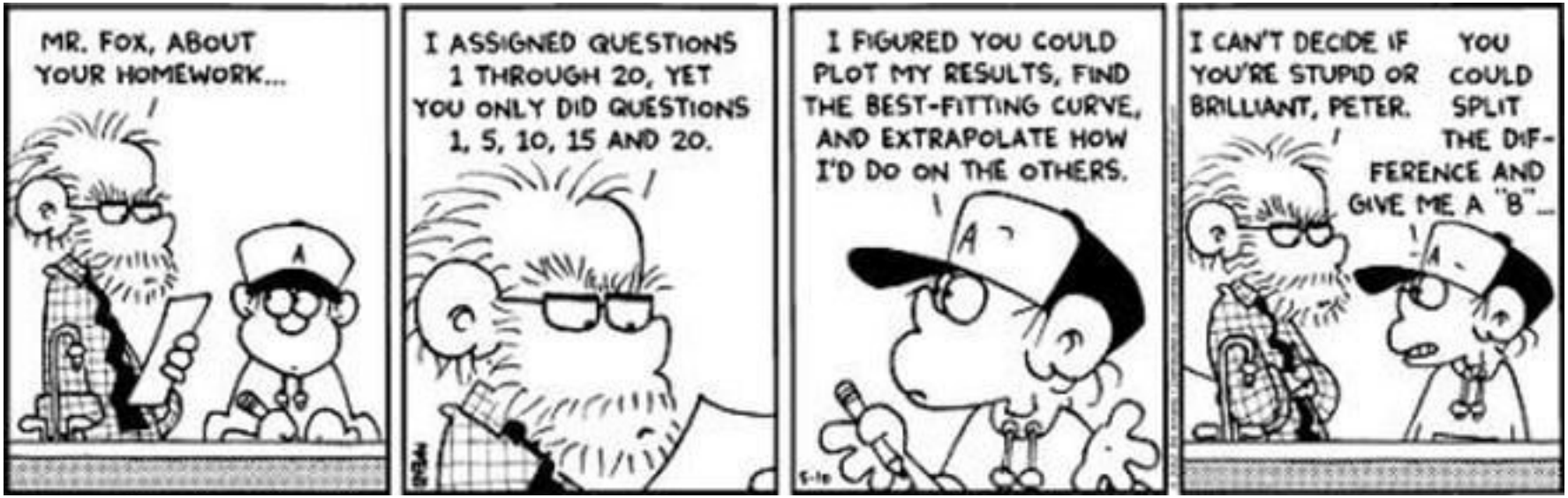
Quick Summary So Far

- Ran an experiment to gather data
- Theory predicts relationship between measurements (displacements) and experimental parameters (masses or forces)
- Linear regression lets us fit best model (line in our case) to observed data
  - Best here means minimize sum squared error between observed and predicted values
- So, let's apply this idea to other data…

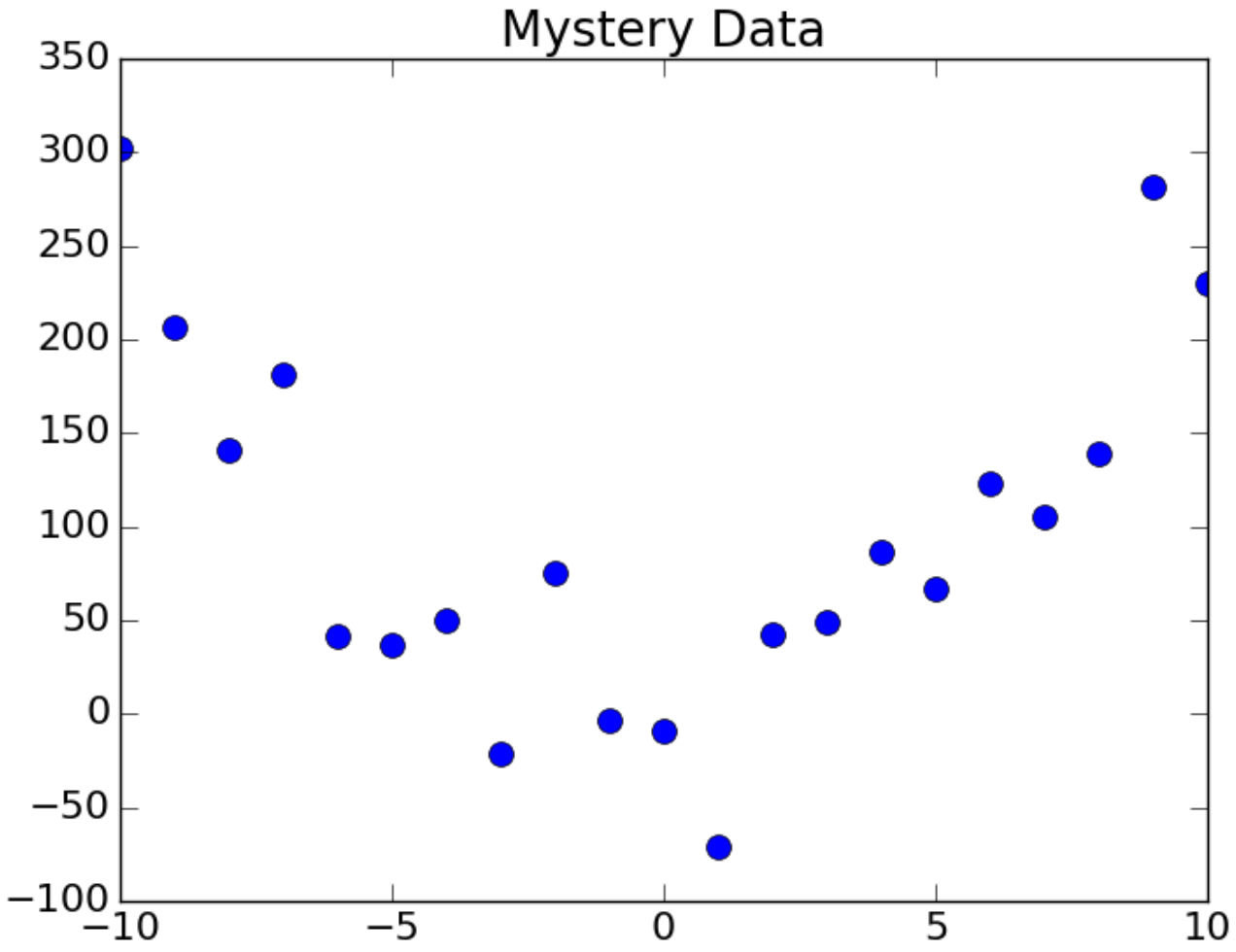




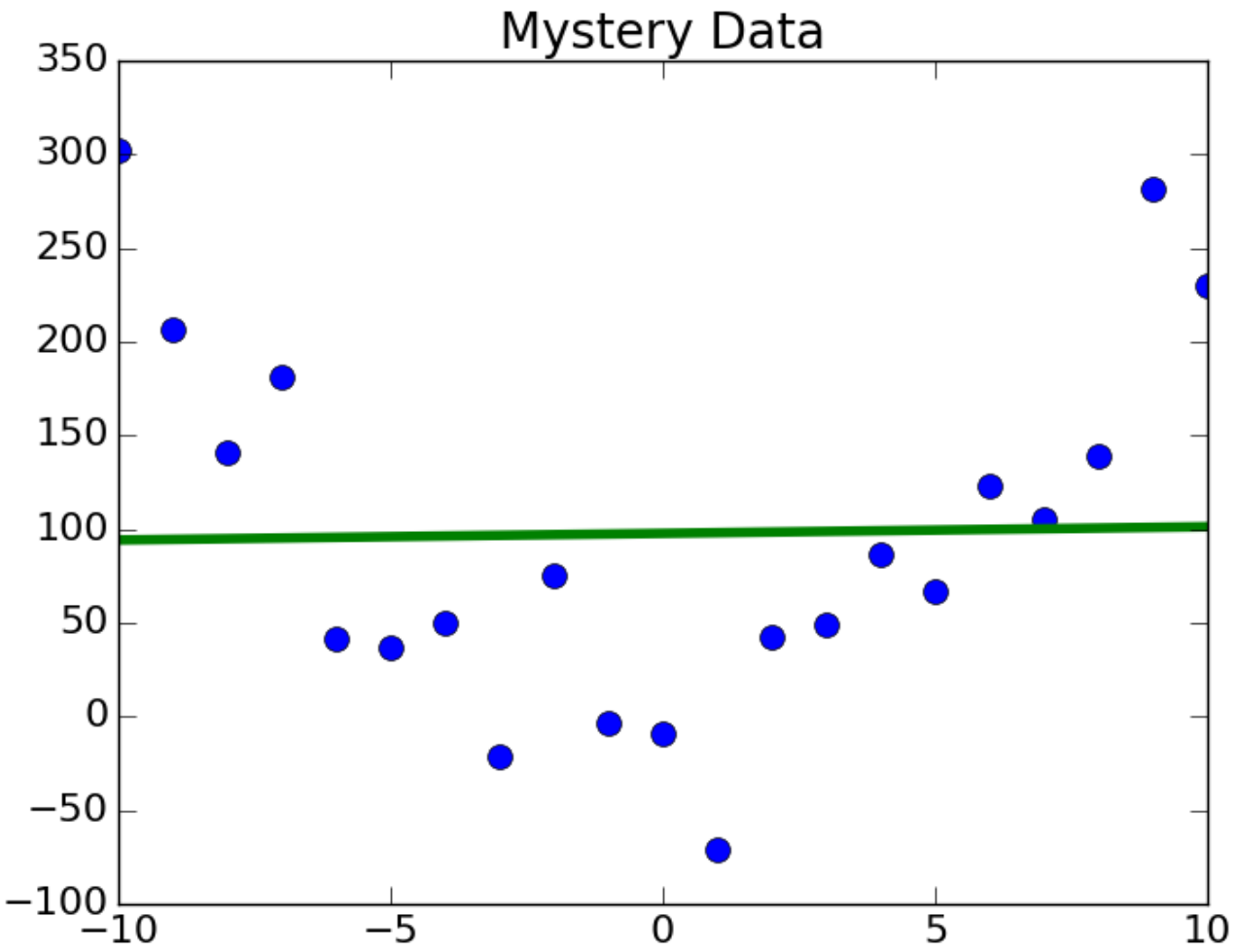
Questions?


MR. FOX, ABOUT YOUR HOMEWORK...
I ASSIGNED QUESTIONS 1 THROUGH 20, YET YOU ONLY DID QUESTIONS 1, 5, 10, 15 AND 20.
I FIGURED YOU COULD PLOT MY RESULTS, FIND THE BEST-FITTING CURVE, AND EXTRAPOLATE HOW I'D DO ON THE OTHERS.
I CAN'T DECIDE IF YOU'RE STUPID OR BRILLIANT, PETER.
YOU COULD SPLIT THE DIFFERENCE AND GIVE ME A "B"...

Another (Mystery) Experiment



Fit a Line



Let's Try a Higher-degree Model

```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Let's Try a Higher-degree Model

```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Note that this is **still** an example of linear regression, even though we are not fitting a line to the data (in this case we are finding the best parabola)

- Objective function depends linearly on unknowns, which are the coefficients of the polynomial

Let's Try a Higher-degree Model

```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Note that this is **still** an example of linear regression, even though we are not fitting a line to the data (in this case we are finding the best parabola)

- Objective function depends linearly on unknowns, which are the coefficients of the polynomial

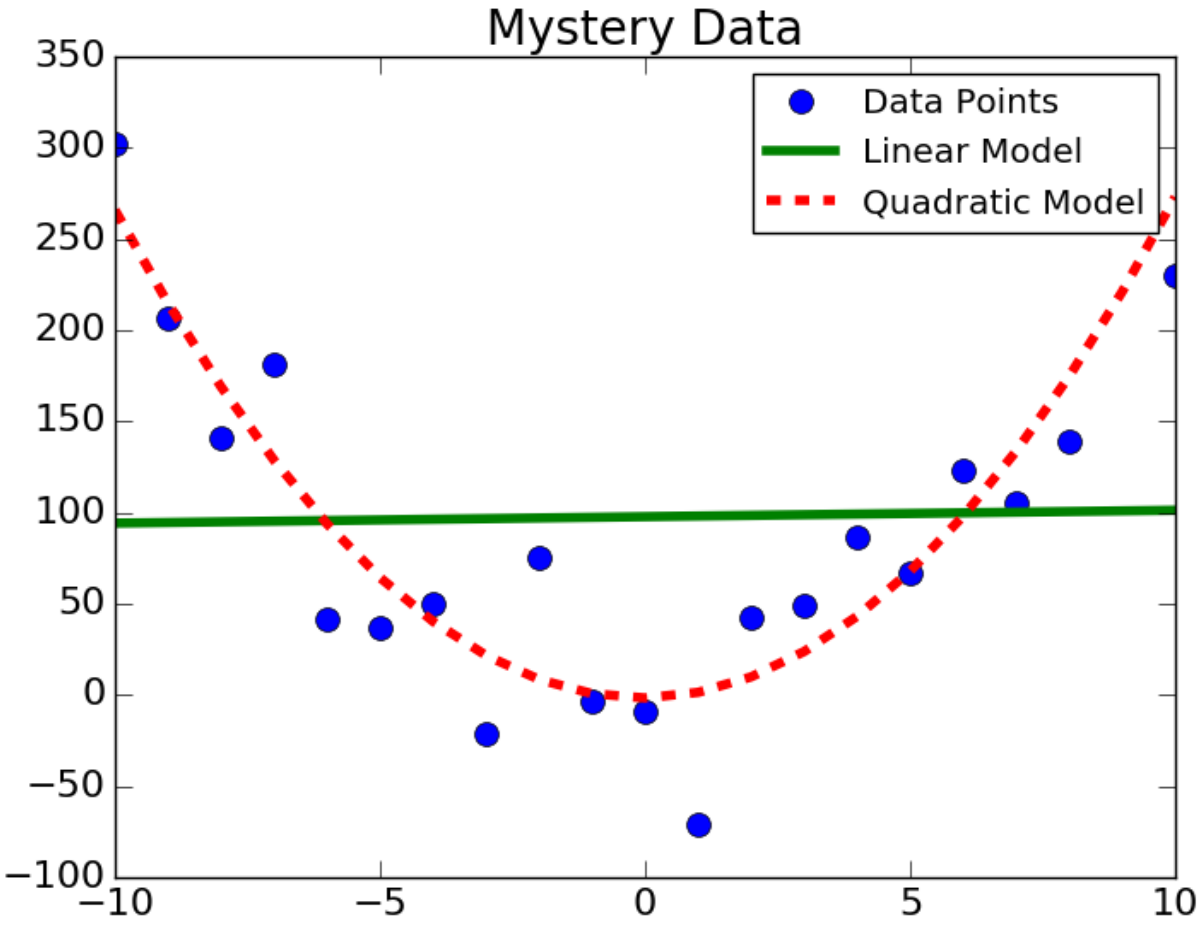
$$\sum_{i=0}^{len(observed)-1} (observed[i] - a * x[i]^2 - b * x[i] - c)$$

What is linear in linear least squares?

- Most importantly refers to model parameters, not input (independent variables)
  - The parameters are used without exponent or cross-multiplication
  - e.g. ax^3+bx^2+cx+d is still linear in a, b, c, d
  - ax + abx + a^2x is not linear in a and b
- Fitting a quadratic polynomial (or any degree) is still linear regression

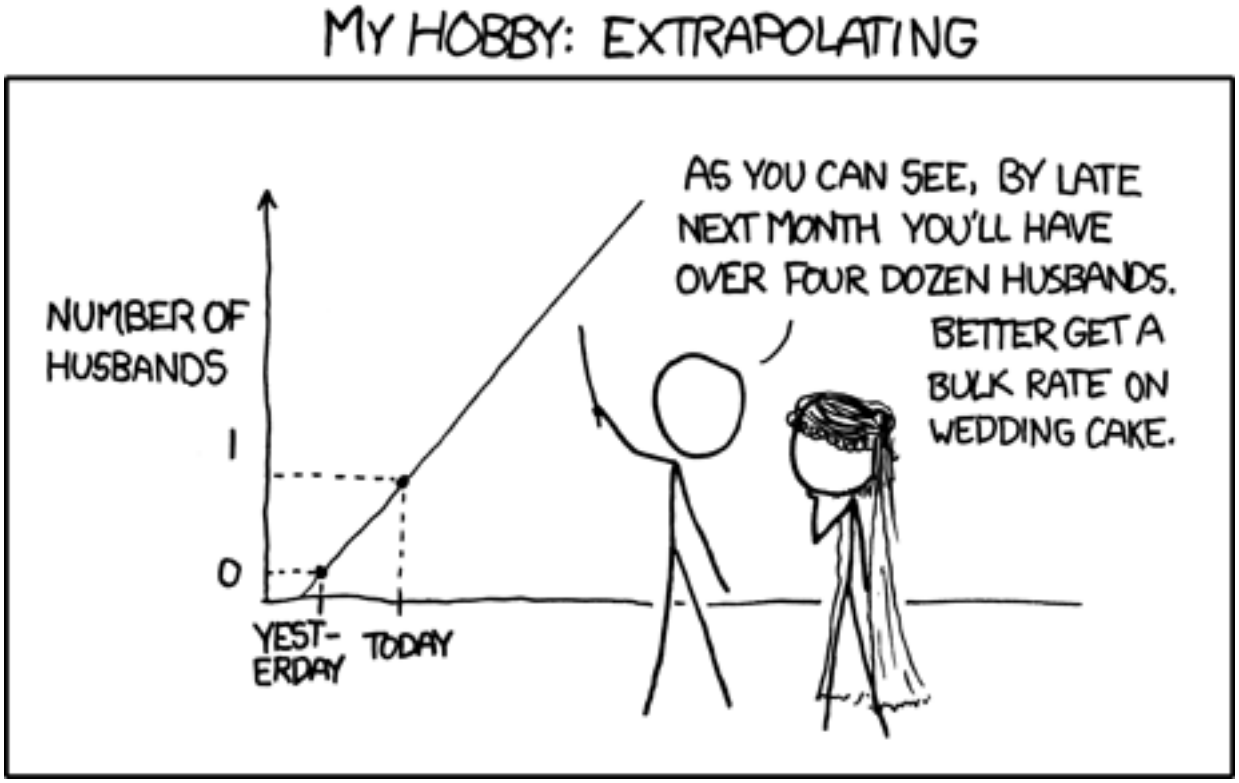
- (the term linear regression is often used for fitting lines, but sometimes to mean linear in parameters. It's messy)

Quadratic Appears to be a Better Fit

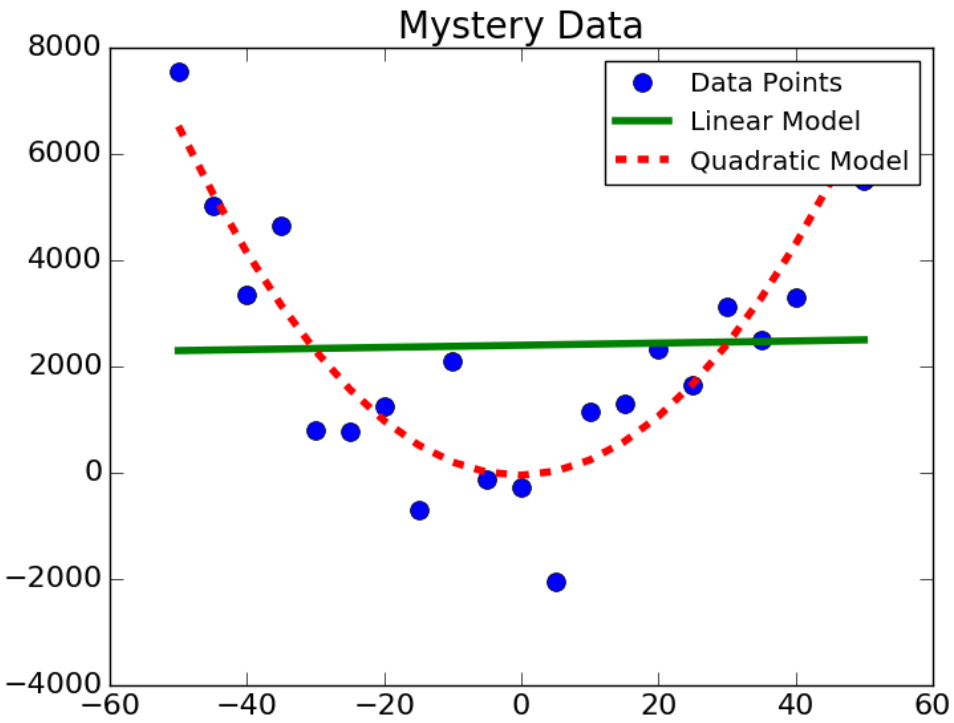


Questions?

- https://xkcd.com/605/



How Good Are These Fits?



- How good are they relative to each other?
- How good are they in an absolute sense?

Comparing Mean Squared Error

```
def aveMeanSquareError(data, predicted):
    error = 0.0
    for i in range(len(data)):
        error += (data[i] - predicted[i])**2
    return error/len(data)

estYVals = pylab.polyval(model1, xVals)
print('Ave. mean square error for linear model =',
      aveMeanSquareError(yVals, estYVals))
estYVals = pylab.polyval(model2, xVals)
print('Ave. mean square error for quadratic model =',
      aveMeanSquareError(yVals, estYVals))
```

Comparing Mean Squared Error

```
def aveMeanSquareError(data, predicted):
    error = 0.0
    for i in range(len(data)):
        error += (data[i] - predicted[i])**2
    return error/len(data)

estYVals = pylab.polyval(model1, xVals)
print('Ave. mean square error for linear model =',
      aveMeanSquareError(yVals, estYVals))
estYVals = pylab.polyval(model2, xVals)
print('Ave. mean square error for quadratic model =',
      aveMeanSquareError(yVals, estYVals))
```

Ave. mean square error for linear model = 9372.73078965
Ave. mean square error for quadratic model = 1524.02044718

Given this improvement in mean squared error from linear to quadratic model, is there something even better?

In an Absolute Sense

- Mean square error useful for comparing two different models for the same data
- Is it also useful for getting a sense of absolute goodness of fit?
  - Is 1524 good?
  - Poll

In an Absolute Sense

- Mean square error useful for comparing two different models for the same data
- Is it also useful for getting a sense of absolute goodness of fit?
  - Is 1524 good?
- Hard to know – no bound on values; not scale independent
  - For example, if we double the masses, get double the error

R^2: coefficient of determination

- Instead we use coefficient of determination, R^2,

y_i are measured values

p_i are predicted values

μ is mean of measured values

R^2: coefficient of determination

- Instead we use coefficient of determination, R^2,

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

← Error in estimates

← Variability in measured data

y_i are measured values
p_i are predicted values
μ is mean of measured values

R^2

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

- By comparing the estimation errors (the numerator) with the variability of the original values (the denominator),
  R^2 captures the proportion of variability in a data set that is accounted for by the model
  - Said differently: compare model to a constant model
  - (the mean is the best constant estimate under least squares)

R^2

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

- Between 0 and 1 when fit generated by a linear regression* and tested on training data
  - If $R^2 = 1$,
  - If $R^2 = 0$,
  - If $R^2 = 0.5$,

* assuming the model has a constant term

R^2

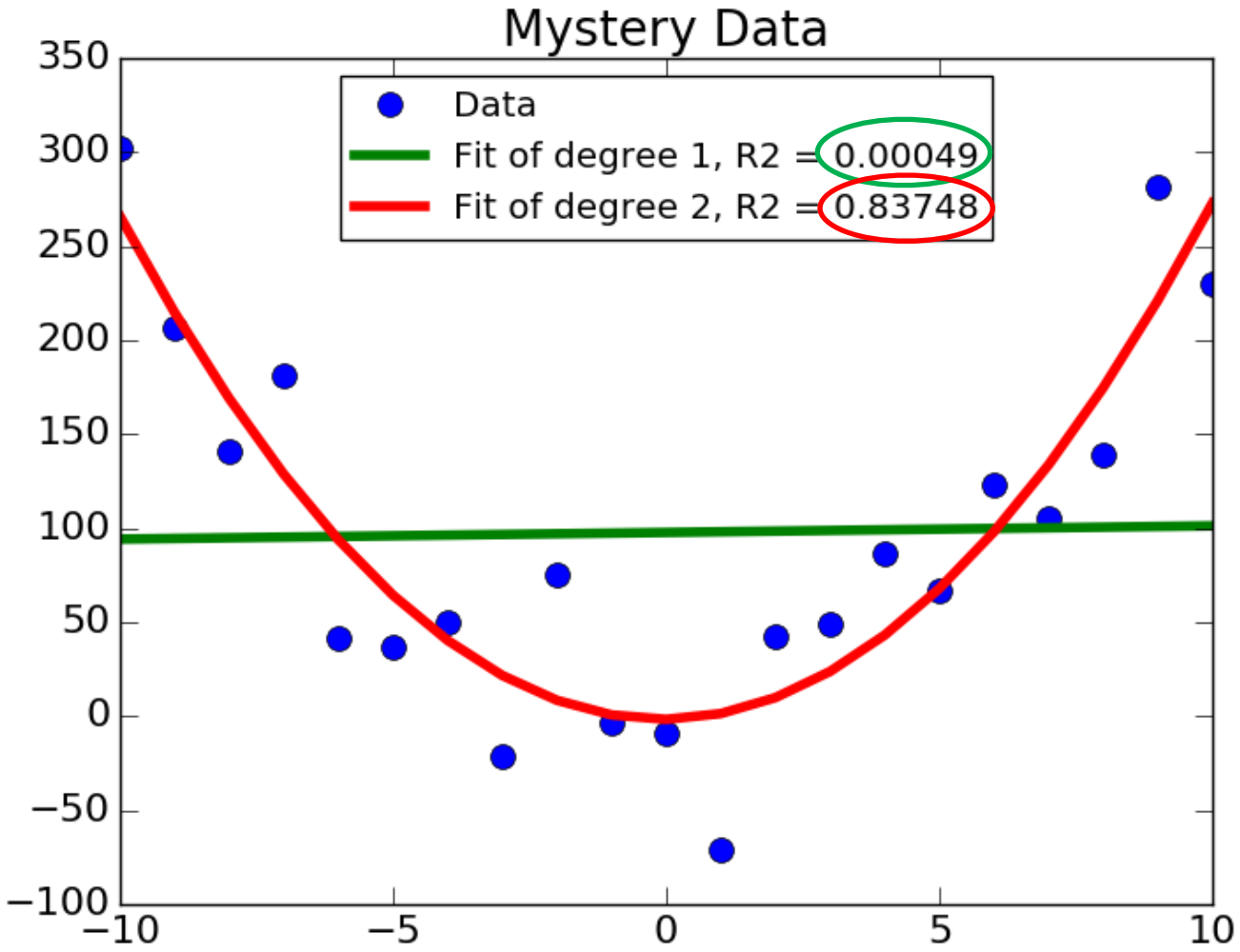
- Between 0 and 1 when fit generated by a linear regression* and tested on training data
  - If $R^2 = 1$, the model explains all of the variability in the data.
  - If $R^2 = 0$,
    there is no relationship between the values predicted by the model and the actual data.
    (no better than constant prediction)
  - If $R^2 = 0.5$,
    the model explains half the variability in the data.

* assuming the model has a constant term

Testing Goodness of Fits

```
def genFits(xVals, yVals, degrees):
    models = []
    for d in degrees:
        model = pylab.polyfit(xVals, yVals, d)
        models.append(model)
    return models

def testFits(models, degrees, xVals, yVals, title):
    pylab.plot(xVals, yVals, 'o', label = 'Data')
    for i in range(len(models)):
        estYVals = pylab.polyval(models[i], xVals)
        error = rSquared(yVals, estYVals)
        pylab.plot(xVals, estYVals,
                   label = 'Fit of degree '\
                   + str(degrees[i])\
                   + ', R2 = ' + str(round(error, 5)))
    pylab.legend(loc = 'best')
    pylab.title(title)
```

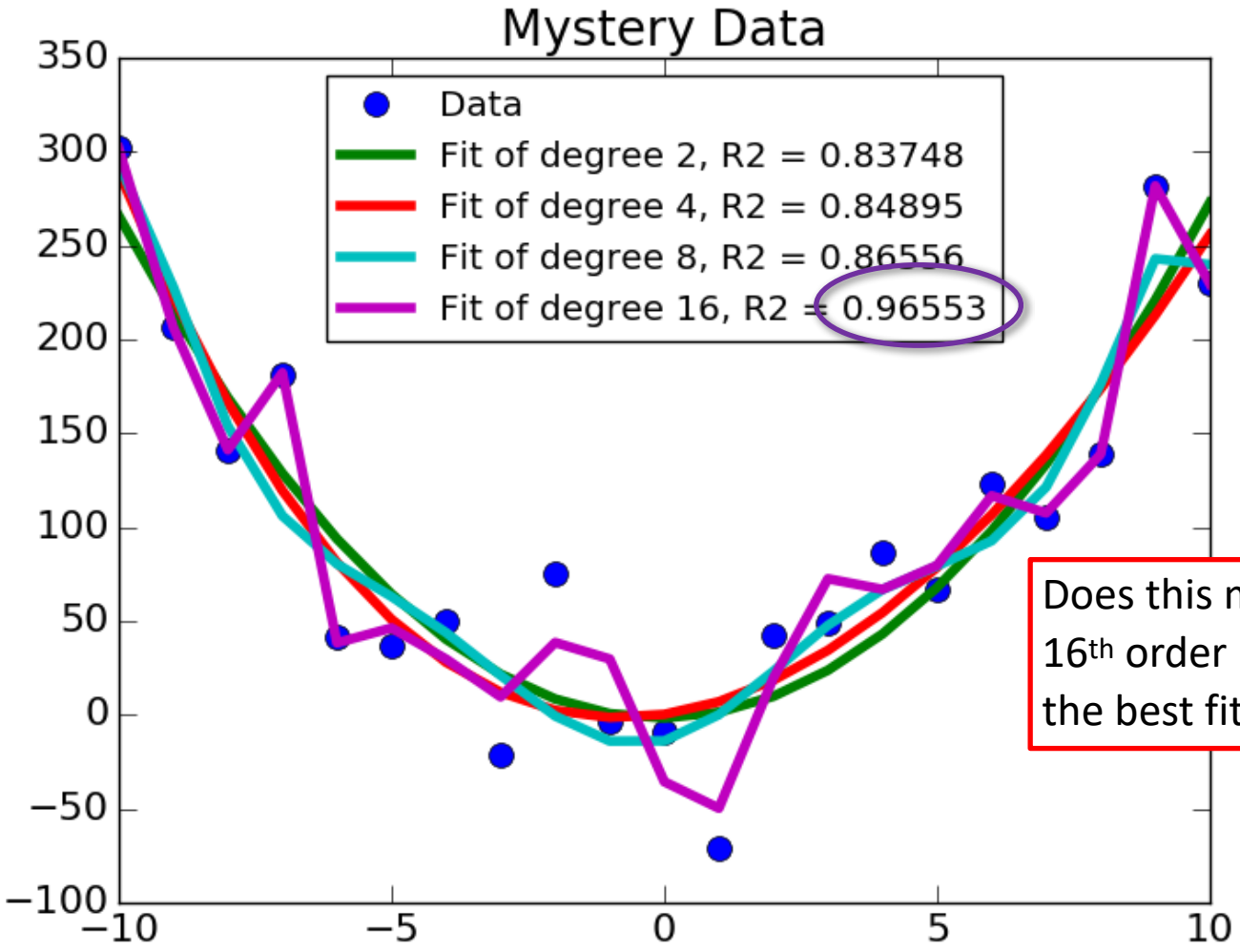
How Well Do Fits Explain Variance?



Can We Do Better?

- Saw that linear fit was poor – both visually and through R^2 measure
- Saw that quadratic fit was better – again both visually and through R^2 measure
- What about fitting higher order polynomials to data?
  - Degree 4?
  - Degree 8?
  - Degree 16?

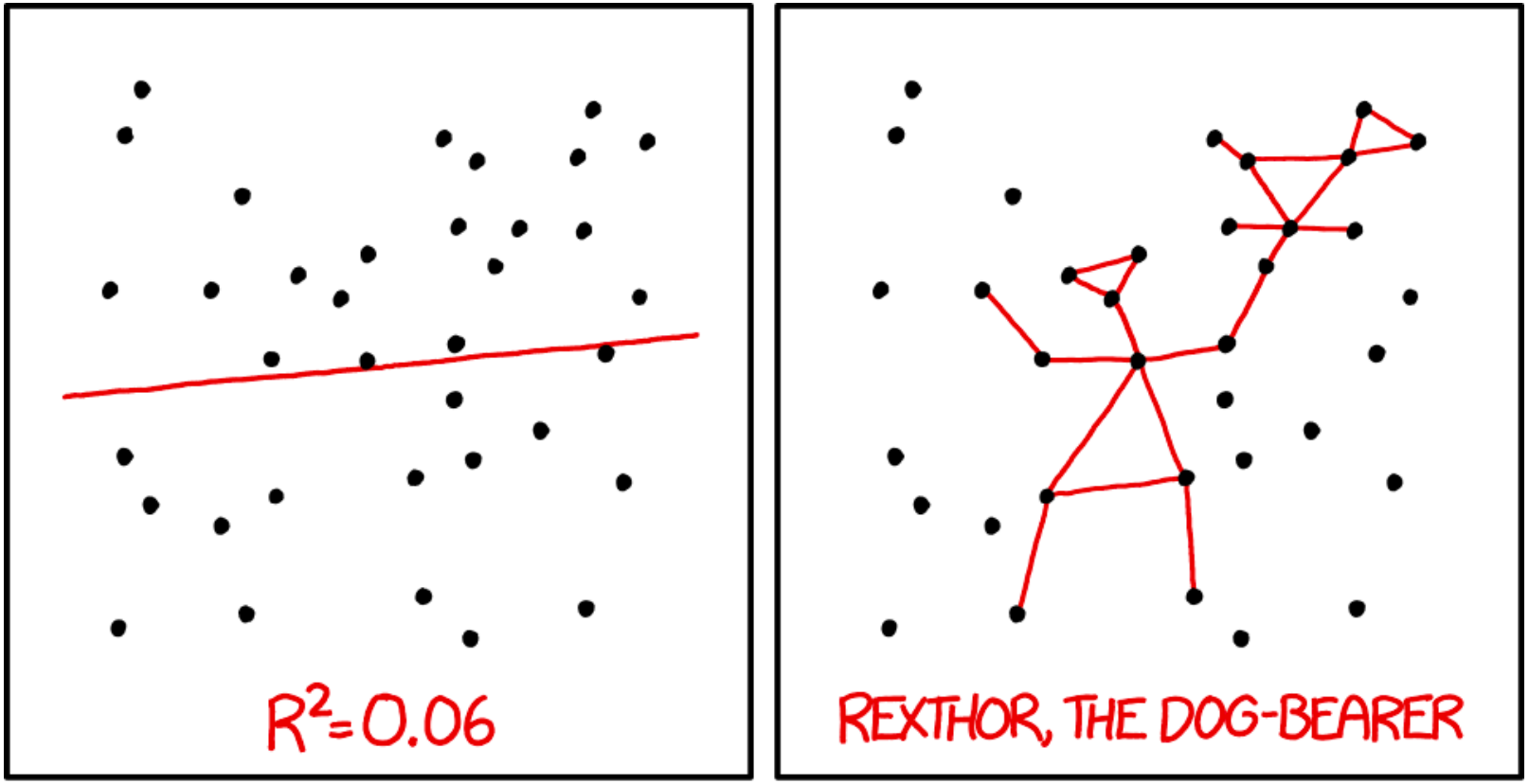
Can We Get a Tighter Fit?



Does this mean that a 16th order polynomial is the best fit for the data?

Questions?

- https://xkcd.com/1725/



Does Tightest = Best?

- Looks like an order 16 fit is really good – so should we just use this as our model?
  - To answer, need to ask – **why build models in first place?**
- 1) Help us understand process that generated the data
  - E.g., the properties of a particular linear spring
- 2) Help us make predictions about out-of-sample data
  - E.g., predict the displacement of a spring when a force is applied to it
  - E.g., predict the effect of treatment on a patient
  - E.g., predict the outcome of an election
- A good model helps us do both of these things

How Mystery Data Was Generated

```
def genNoisyParabolicData(a, b, c, xVals, fName):
    yVals = []
    for x in xVals:
        theoreticalVal = a*x**2 + b*x + c
        yVals.append(theoreticalVal + random.gauss(0, 35))
    f = open(fName,'w')
    f.write('x       y\n')
    for i in range(len(yVals)):
        f.write(str(yVals[i]) + ' ' + str(xVals[i]) + '\n')
    f.close()

#parameters for generating data
xVals = range(-10, 11, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals,  'Mystery Data.txt')
```

Zero mean, Gaussian noise

If data was generated by quadratic, why was 16th order polynomial the "best" fit?

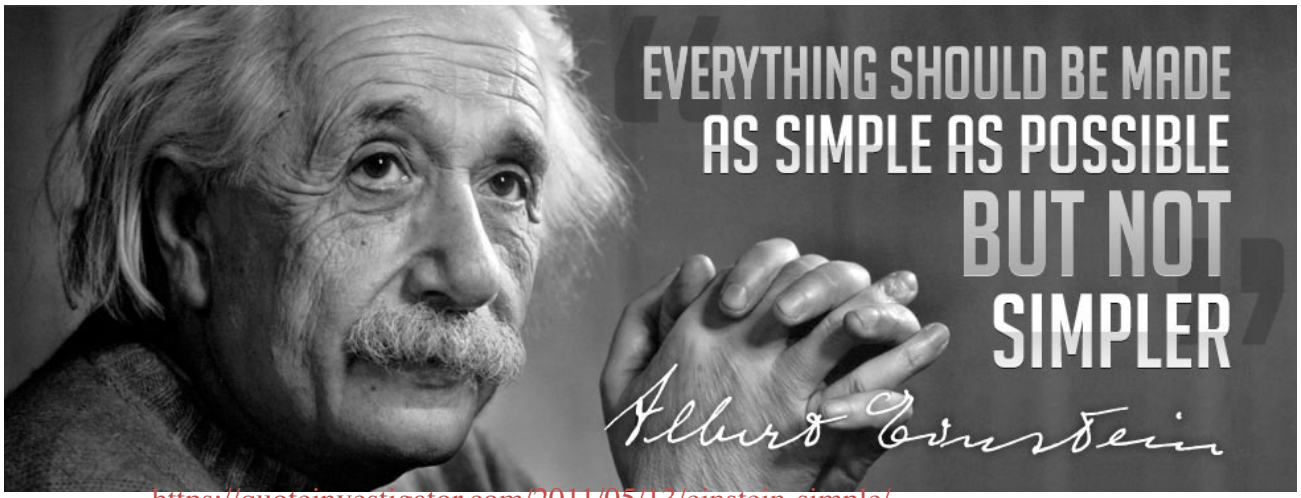
Because it fit the noise.

Increasing the Complexity

- Is it just luck that we got a "better" fit on training data with higher order model?
- What happens when we increase order of polynomial during training?
  - Can we get a worse fit to training data?
- If extra term is useless, coefficient will merely be zero
- But if data is noisy, can fit the noise rather than the underlying pattern in the data
  - May lead to a "better" R^2 value, but not really a "better" fit

The Take Home Message

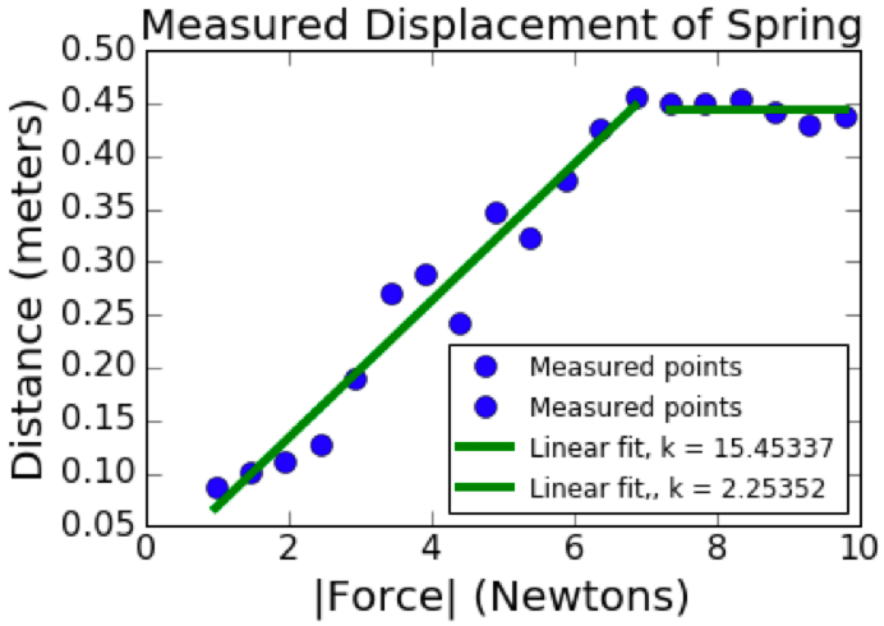
- Choosing an overly-complex model leads to overfitting to the training data
- Increases the risk of a model that works poorly on data not included in the training set
- On the other hand choosing an insufficiently complex model has other problems
  - As we saw when we fit a line to data that was basically parabolic



https://quoteinvestigator.com/2011/05/13/einstein-simple/

One Last Thought

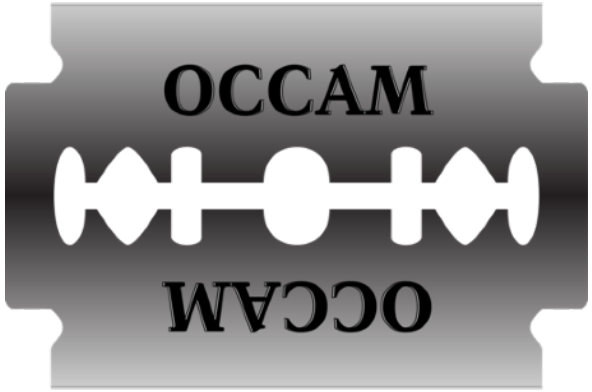
- Alternatively, search for point at which to break data into two sets, and fit model to first set of data but fit constant line to second set; look for break that minimizes sum of residual error in both parts



- R^2 value for lower part now .9539; without break, have R^2 of .8815

Wrapping Up Curve Fitting

- We can use linear regression to fit a curve to data
  - Mapping from independent values to dependent values
- That curve is a model of the data that can be used to predict the value associated with independent values we haven't seen (out-of-sample data)
- R-squared used to evaluate model
  - Higher not always "better" because of risk of over fitting
- Choose complexity of model based on
  - Theory about structure of data
  - Cross validation
  - Simplicity



Occam's Razor

- *"Frustra fit per plura quod potest fieri per pauciora"*
  - *"It is futile to do with more things that which can be done with fewer"*
- Among competing hypotheses, the one with the fewest assumptions should be selected

William of Occam
1287-1347

Quiz?

- If a miracle occurs, we still have time for cross correlation
-

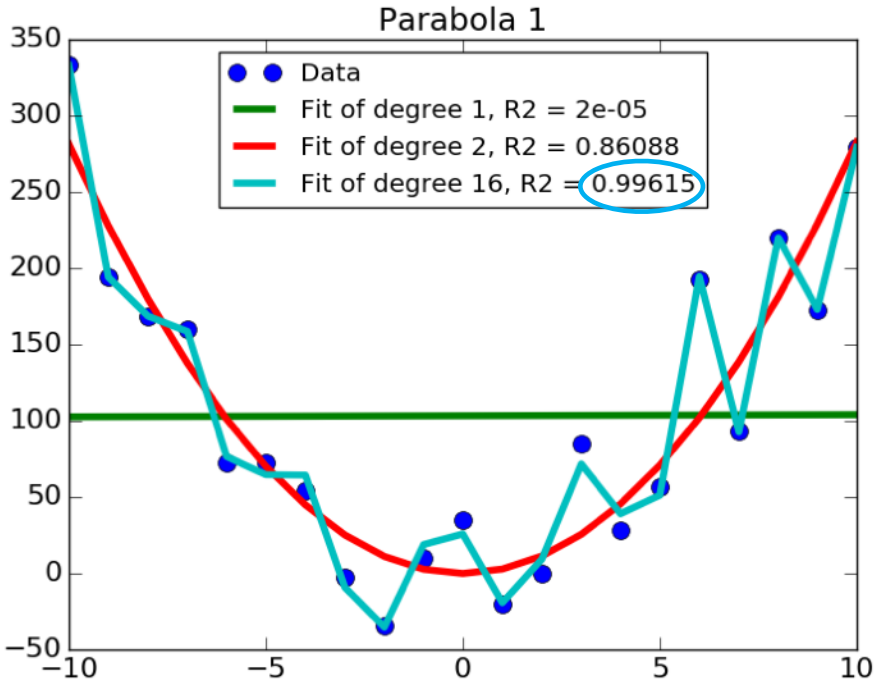
Training versus Testing

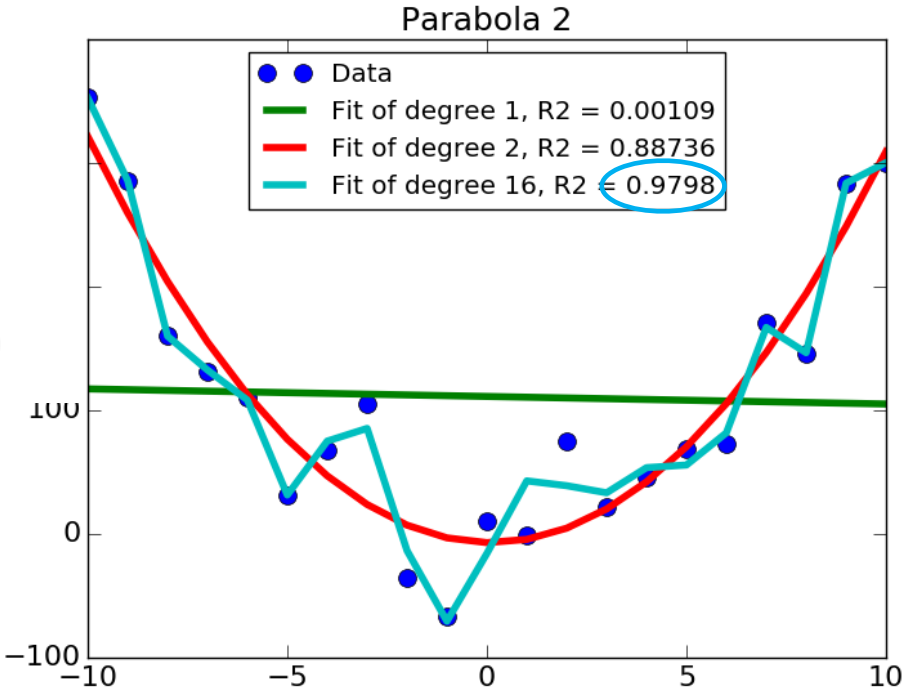
- One way to separate out impact of noise is to take advantage of fact that noise will typically be different each time we sample a system
- So can cross validate:
  - Generate a set of data as a "training" set, and use to fit a model
  - Generate a second set of data as a "test" set, and see how well the model from the training set accounts for the test set

Generate 2 Data Sets from Same Distribution

```
xVals = range(-10, 11, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals, 'parabola1.txt')
genNoisyParabolicData(a, b, c, xVals, 'parabola2.txt')

degrees = (1, 2, 16)
xVals1, yVals1 = getData('parabola1.txt')
models1 = genFits(xVals1, yVals1, degrees)
testFits(models1, degrees, xVals1, yVals1, 'Parabola 1')

pylab.figure()
xVals2, yVals2 = getData('parabola2.txt')
models2 = genFits(xVals2, yVals2, degrees)
testFits(models2, degrees, xVals2, yVals2, 'Parabola 2')
```

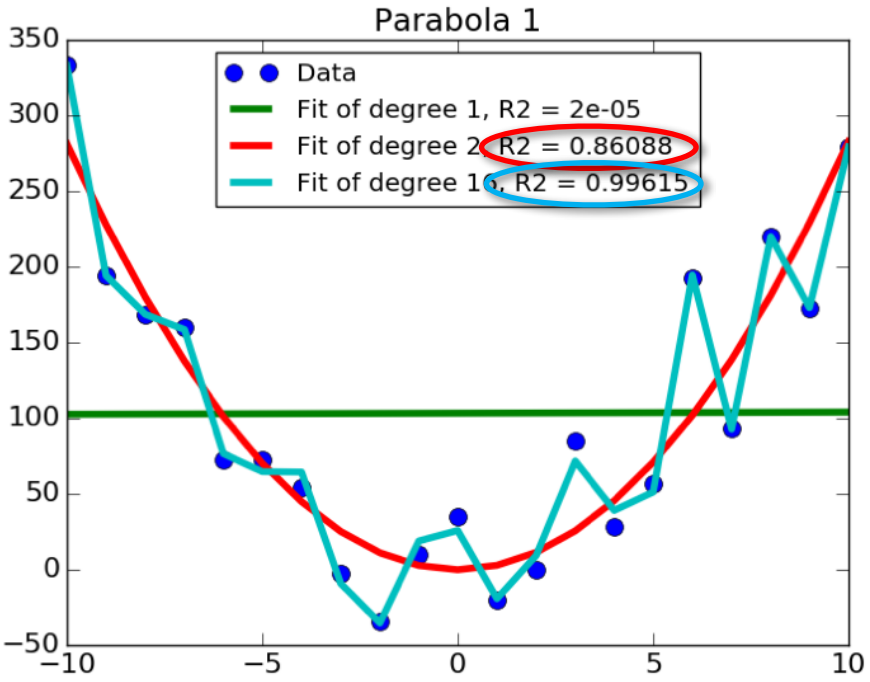
Look at Fits

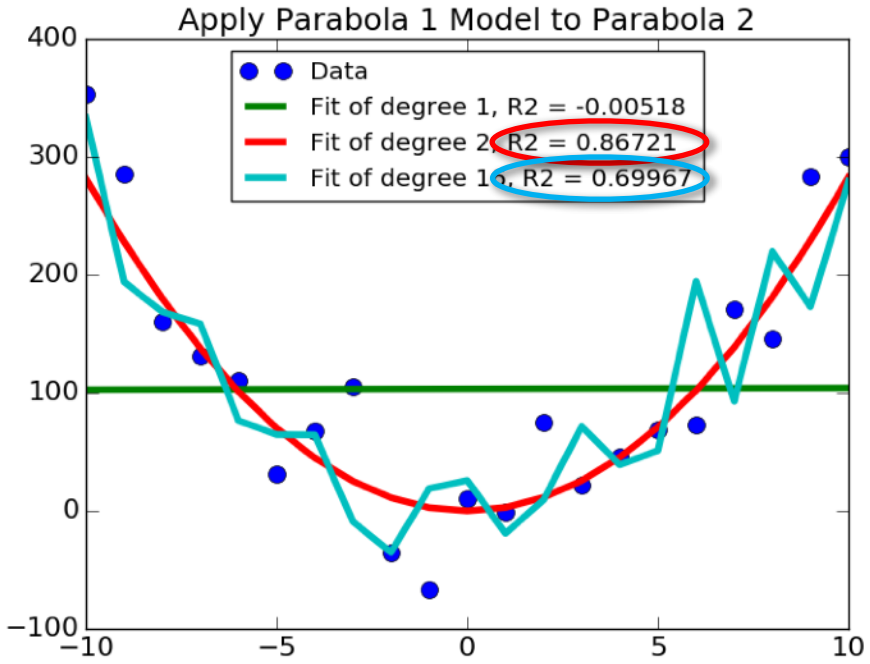

Parabola 1
Data
Fit of degree 1, R2 = 2e-05
Fit of degree 2, R2 = 0.86088
Fit of degree 16, R2 = 0.99615
350
300
250
200
150
100
50
0
−50
−10
−5
0
5
10


Parabola 2
Data
Fit of degree 1, R2 = 0.00109
Fit of degree 2, R2 = 0.88736
Fit of degree 16, R2 = 0.9798
100
0
−100
−10
−5
0
5
10

Training and Testing Errors

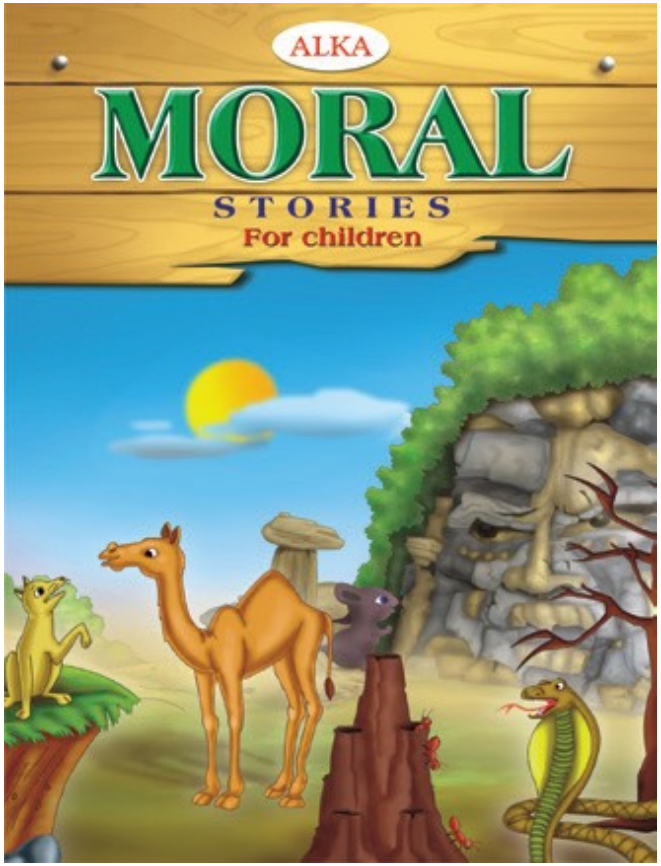
```
testFits(models1, degrees, xVals1, yVals1, 'Parabola 1')

testFits(models1, degrees, xVals2, yVals2,
         'Apply Parabola 1 Model to Parabola 2')
```





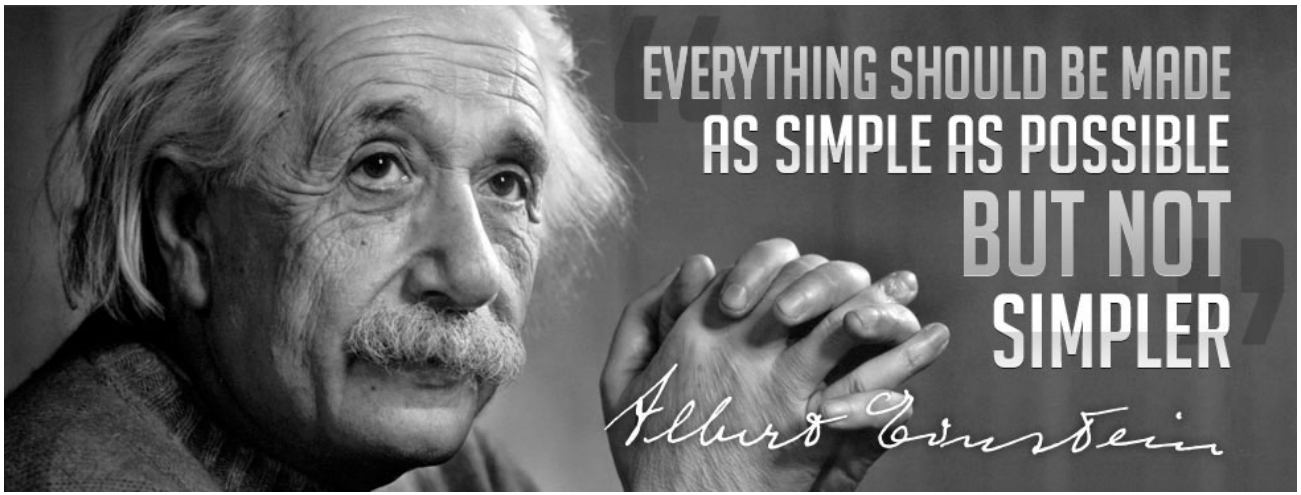
The Moral of the Story

- 16-degree polynomial is an example of overfitting to the data
- If we only look at how well model fits training data, we may not detect that model is too complex
- Need to cross validate: Train on one data set, then test on a different one



The Take Home Message

- Choosing an overly-complex model leads to overfitting to the training data
- Increases the risk of a model that works poorly on data not included in the training set
- On the other hand choosing an insufficiently complex model has other problems
  - As we saw when we fit a line to data that was basically parabolic




QUIZ