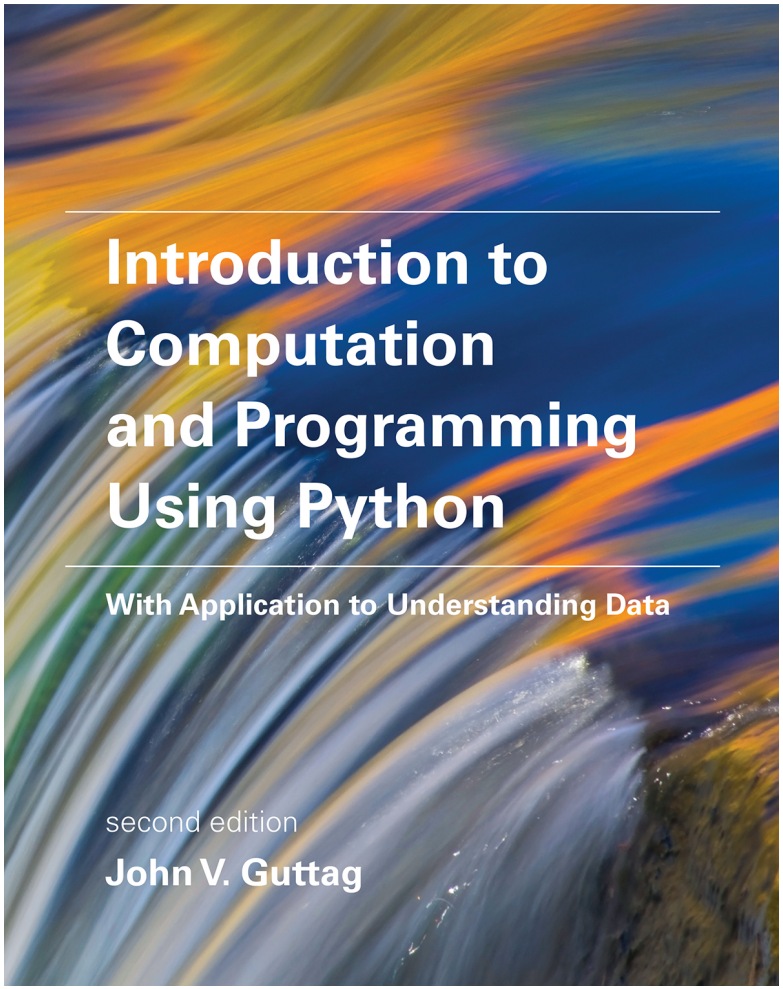
Understanding Experimental Data

Eric Grimson

MIT Department of Electrical Engineering and Computer Science

Assigned Reading

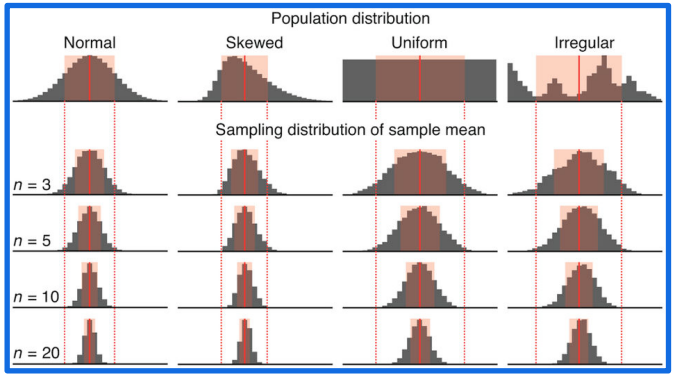
- Today:
  - Chapter 18
- Next lecture:
  - Chapter 22



Where Have We Been?

- Monte Carlo simulation to build models
  - The world is mostly stochastic, so draw multiple samples, then infer explicit model parameters from statistics
  - Useful tool even when randomness not present
  - Can estimate reliability of simulation results

- Understanding populations
  - Cannot examine all members
  - Rely on sampling to infer information
  - Can estimate confidence in inferences
  - Central Limit Theorem lets us use a single sample, and still assert inferences with specific confidence levels



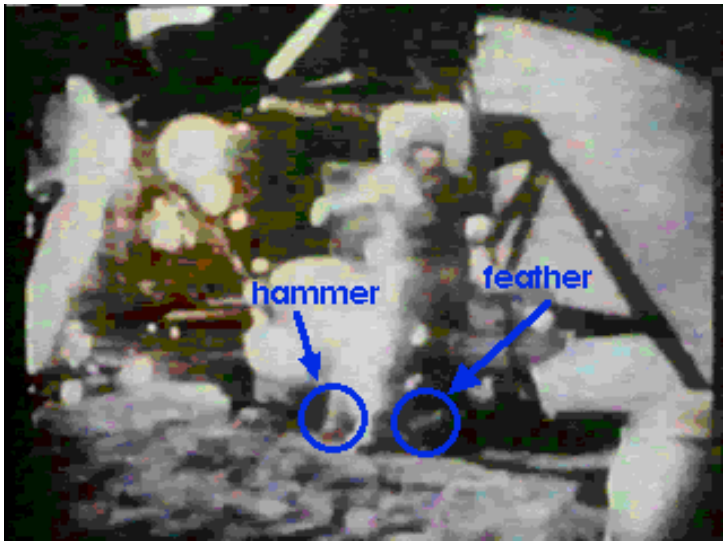
- Have focused on estimating explicit parameters of distribution of samples
  - E.g, mean and deviation

Where Have We Been?

- Thus, we have applied methods where:
  - we know the underlying model, and want to estimate the value of a critical parameter through direct simulation
    - e.g., roulette spins, die rolls, random walk of particle or cell
  - we want to estimate explicit parameters of a population without examining the whole population
- What happens when the situation we want to model only has an indirect connection to things we can observe?
  - Combine actual observations and theory to set up computational experiment

Statistics Meets Experimental Science

- Conduct an experiment to gather (noisy) data
  - Physical (e.g., in a mechanics 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
  - E.g., what is gravitational constant?
- We'd like to explore this idea, but can't afford a field trip to the moon, so consider, instead, a spring…



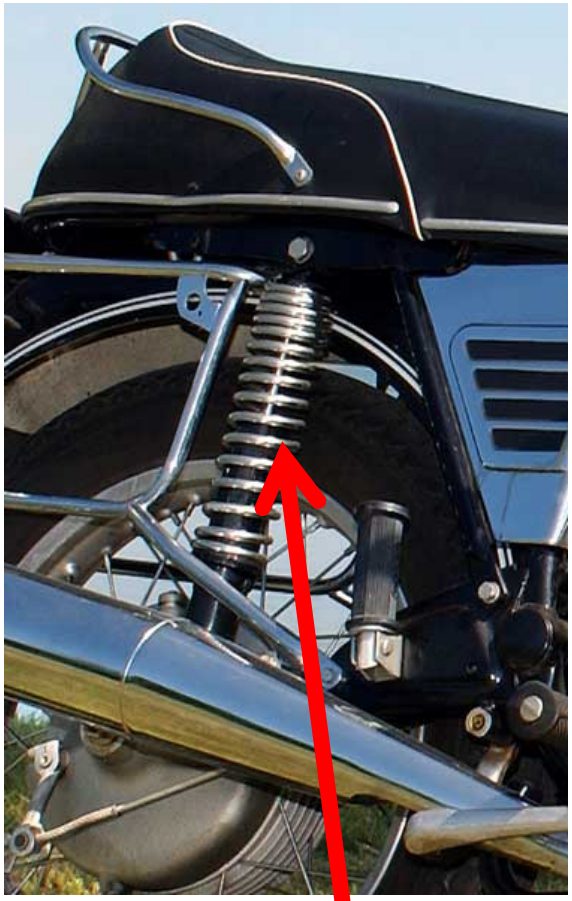
$$F_g = \frac{Gm_1m_2}{r^2}$$

Use equations and experimental observations to find global constants

One Kind of Spring

Another Kind of Spring

This Kind of Spring



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

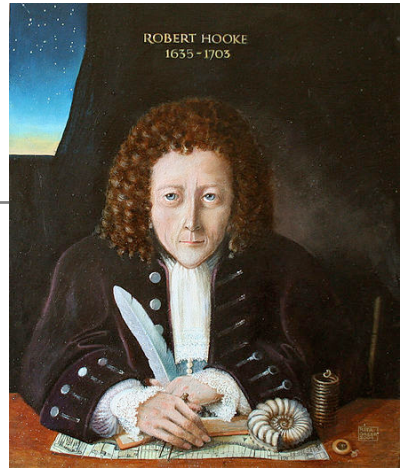
$k \approx 1N/m$

Linear spring: amount of force needed to stretch or compress spring is linear in the distance the spring is stretched or compressed, up to some maximum force

Each spring has a spring constant, k, that determines how much force is needed

Newton = force to accelerate 1 kg mass 1 meter per second per second

Model comes from Hooke's Law



- Robert Hooke
  - 1635-1703
  - Curator of experiments of the Royal Society of London
  - Surveyor for City of London after the Great Fire
    - Close friend of Christopher Wren (St. Paul's Cathedral architect)
  - Discovered law of elasticity
    - Led to invention of balance spring, which led to first accurate watch
  - Observed that gravity was an inverse effect, but didn't know it was inverse square (Newton gets credit for that)
  - Huge believer in running experiments and then building models
    - *"It is commonly believed that anyone who tabulates numbers is a statistician. This is like believing that anyone who owns a scalpel is a surgeon."*
    - *"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

- $F = -k\delta$

Won't worry about direction of force, so can ignore the '-' sign

- How much does a rider have to weigh to compress spring 1cm?

$$F = 0.01m * 35{,}000N/m$$

$$F = 350N$$

$$mass * 9.81m/s^2 = 350N$$

$$mass = \frac{350N}{9.81m/s^2}$$

$$mass = \frac{350k}{9.81}$$

This *k* refers to kilograms, not the spring constant!

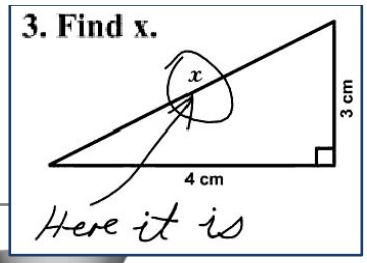
$$mass \approx 35.68k$$

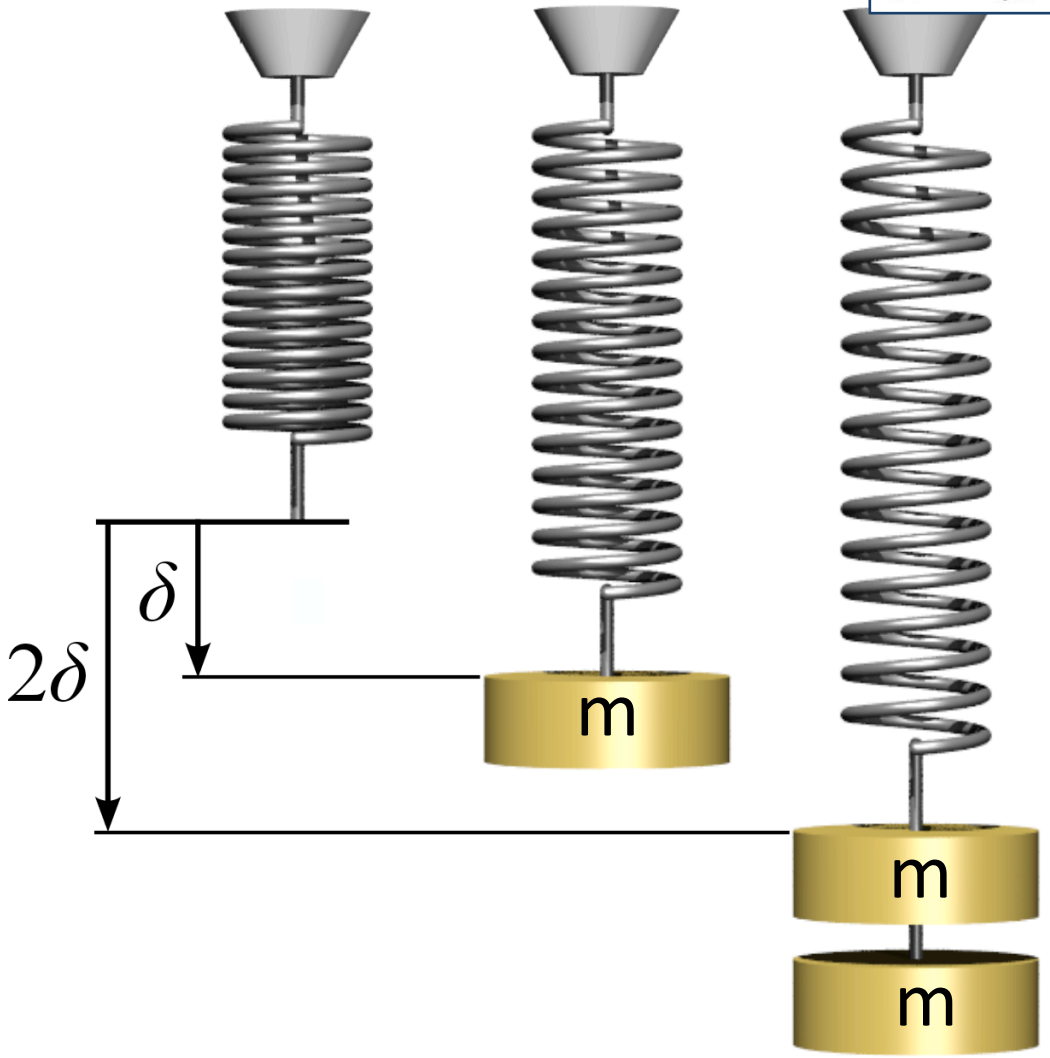
$$F = mass * acc$$

$$F = mass * 9.81m/s^2$$

Finding k

- $F = k\delta$
- $k = F/\delta$
- $k = 9.81*m/\delta$

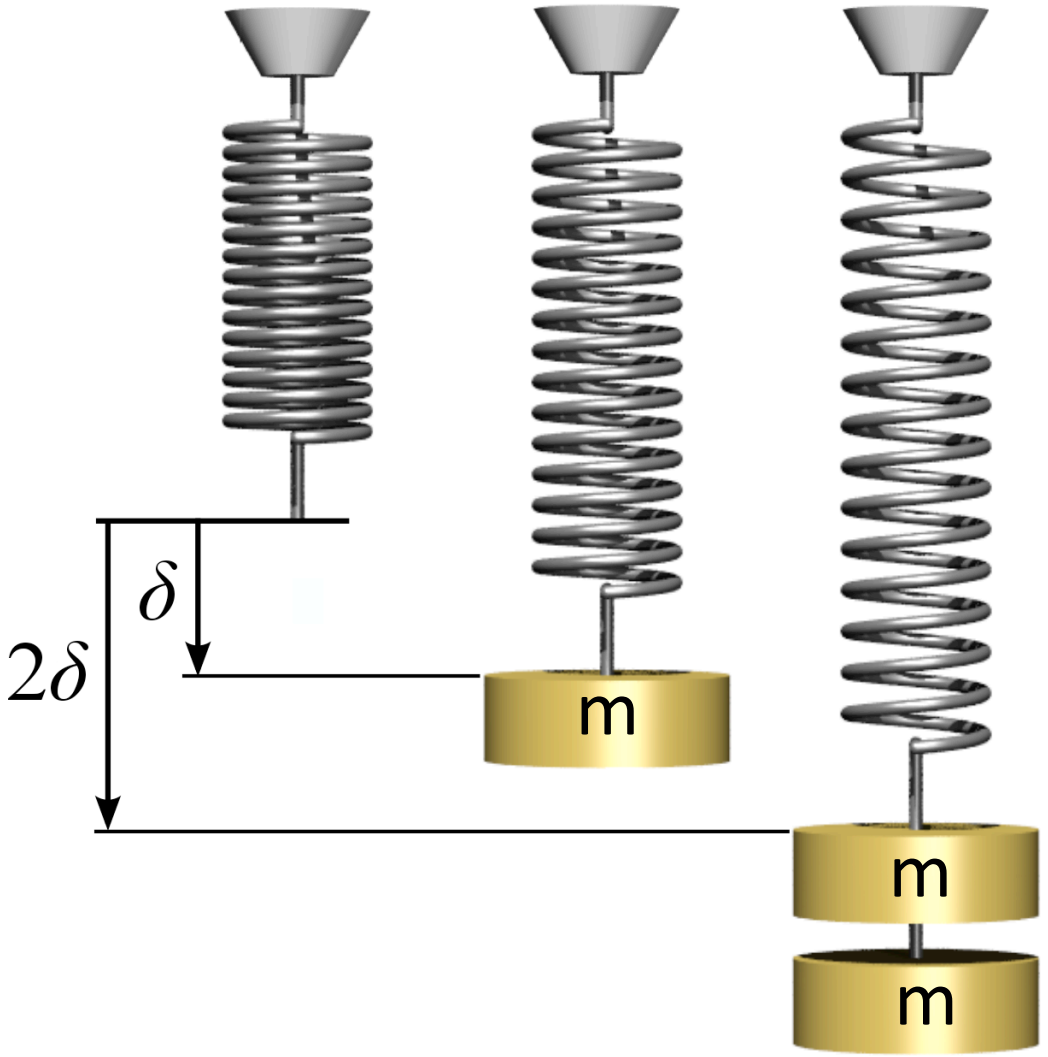




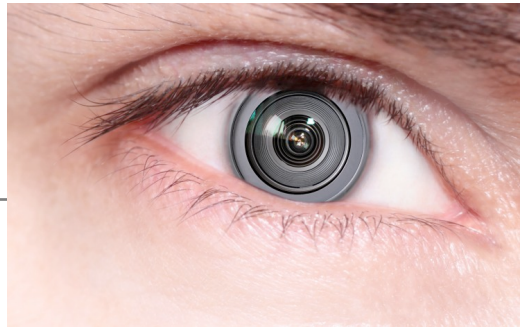
By Yapparina (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Some Data for Finding k

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

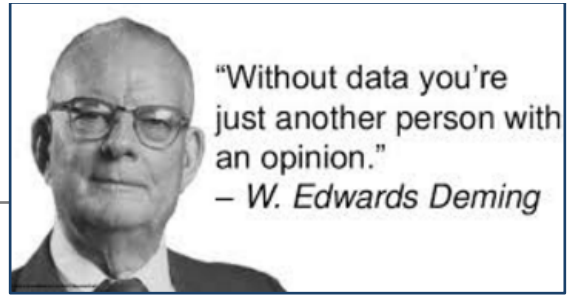


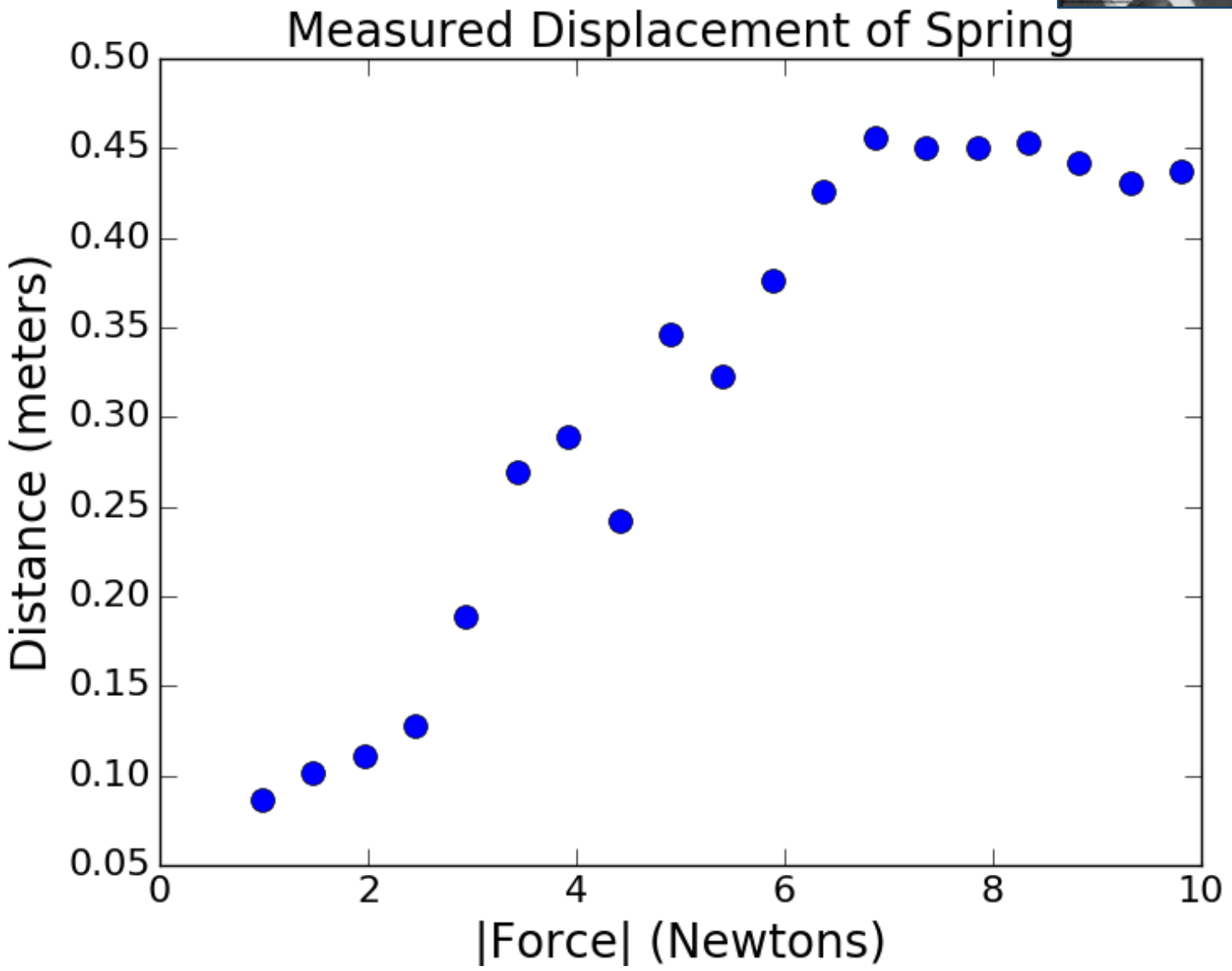
```
def plotData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals) #masses
    yVals = pylab.array(yVals) #distances/displacements
    xVals = xVals*9.81  #acc. due to gravity; forces
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured displacements')
   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

A reminder about python arrays:

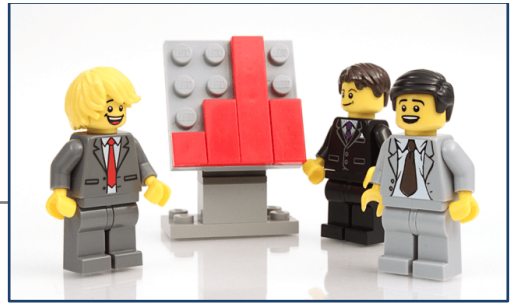
- Converts a list into a linear data structure
- Can treat arrays algebraically; e.g., if a and b are arrays, then:
  - a*2 multiplies every element of a by 2
  - a + 3 adds 3 to every element of a
  - a – b subtracts each element of b from corresponding element of a
  - a*b multiplies each element of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

Taking a Look at the Data

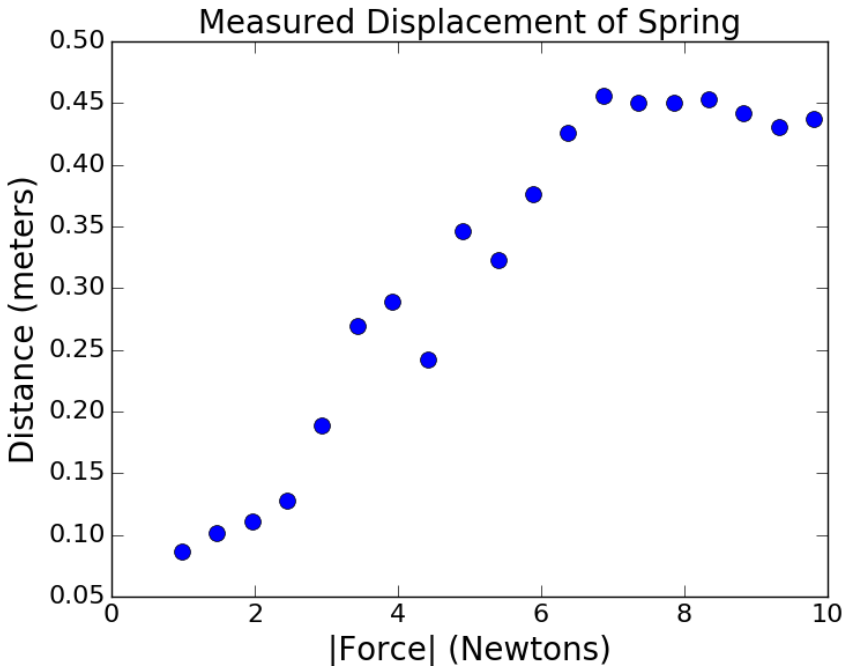




What Can We Do With This Data?



- 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?

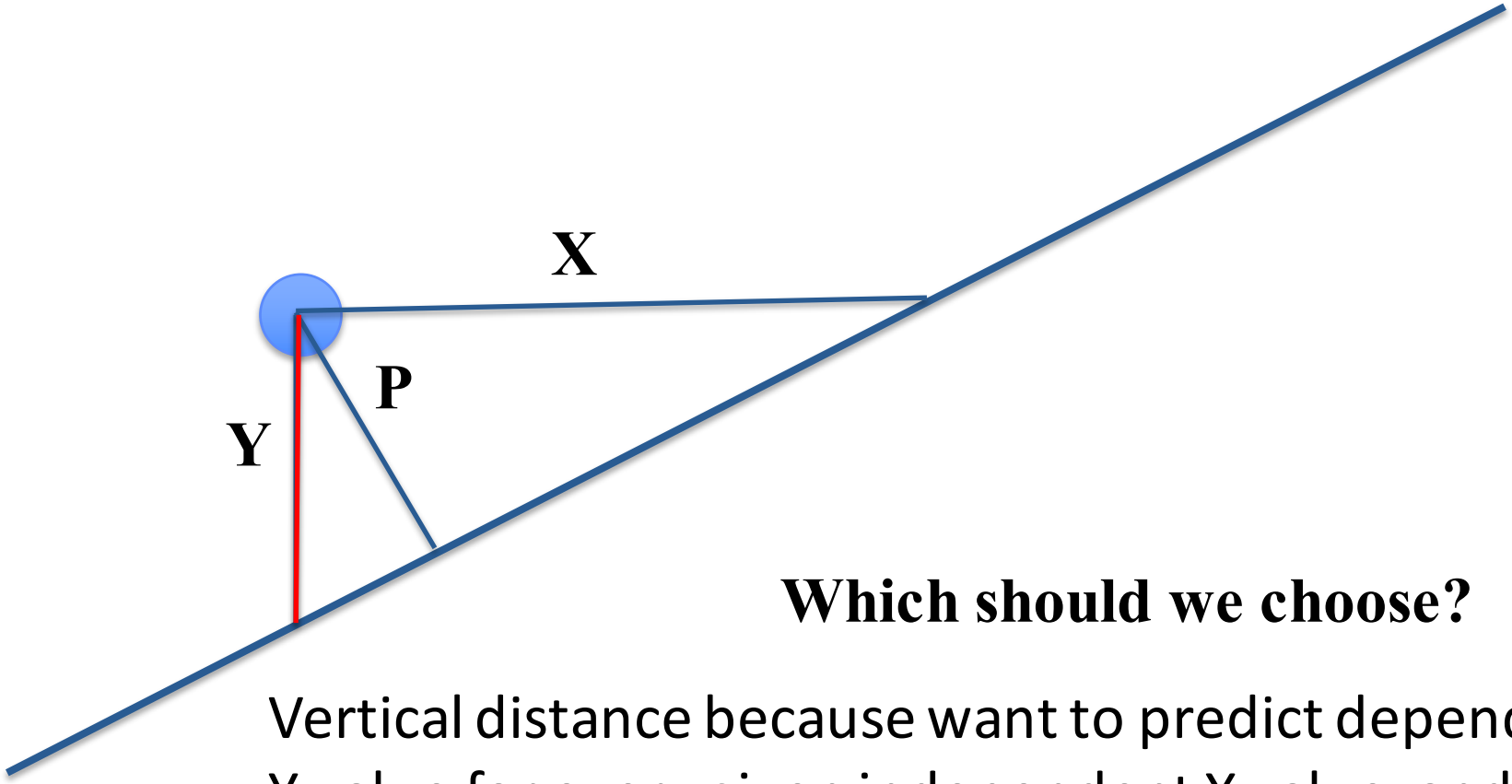


Could just estimate k for each sample, then average those values, but this isn't guaranteed to minimize variance of estimates

Fitting Curves to Data

- Would like to find a curve (from set of possible models) that best accounts for (or fits to) the data
  - E.g., find line from set of all possible lines that best matches data
  - When we fit a curve to a set of data, we are finding a model that relates an independent variable (e.g., mass or force) to an estimated value of a dependent variable (e.g., displacement)
- To decide how well a given curve fits the data, we need to measure the goodness of the fit – called the **objective function**
- Once we define the objective function, we also need an algorithm to find the curve that minimizes it
- In this case, we want to find a **line** such that some function of the distances from the **line** to the measured points is minimized
- If we can do this, then we can predict results on unseen data points

Measuring Distance



Which should we choose?

Vertical distance because want to predict dependent Y value for every given independent X value, and vertical distance measures error in that prediction

Least Squares Objective Function

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

- Look familiar?
  - This is variance times number of observations
  - So minimizing this will also minimize the variance

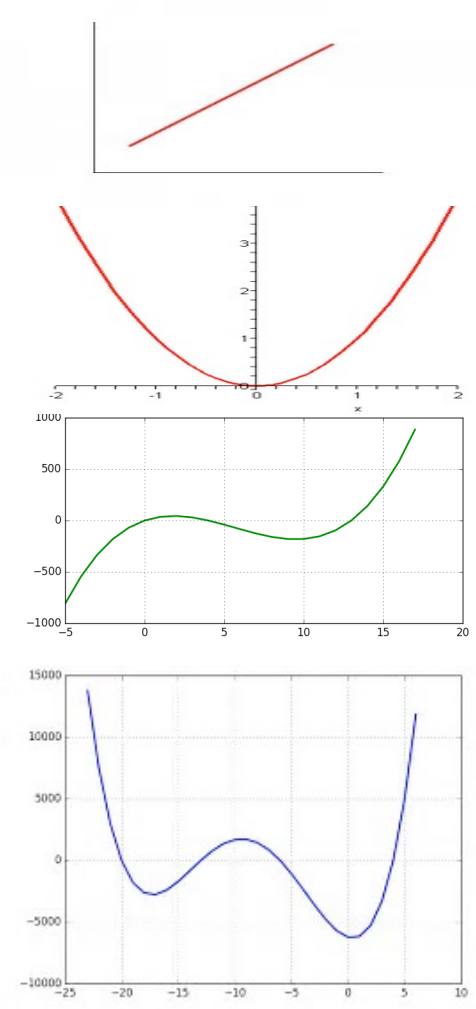
Solving for Least Squares

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

- To minimize this objective function, want to find a curve for the **predicted** observations that leads to minimum value of sum of squared differences between observed and predicted values
- Need to make a choice on kinds of curves – we will use **polynomials of one variable**
- Also need to find the best curve – use **linear regression** to find a polynomial representation for the predicted model that minimizes the objective function

Aside: Polynomials with One Variable (x)

- Definition: 0 or sum of finite number of non-zero terms
- Each term of the form cx^p
  - c, the coefficient, a real number
  - p, the degree (or power) of the term, a non-negative integer
- The degree of the polynomial is the largest degree of any non-zero term
- Examples
  - Line: $ax + b$
  - Parabola: $ax^2 + bx + c$
  - Cubic: $ax^3 + bx^2 + cx + d$
  - Quartic: $ax^4 + bx^3 + cx^2 + dx + e$



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 ith data point (e.g., mass*acceleration or force), and *observed[i]* is the corresponding measured value (e.g., displacement).

Solving for Least Squares

- For linear case

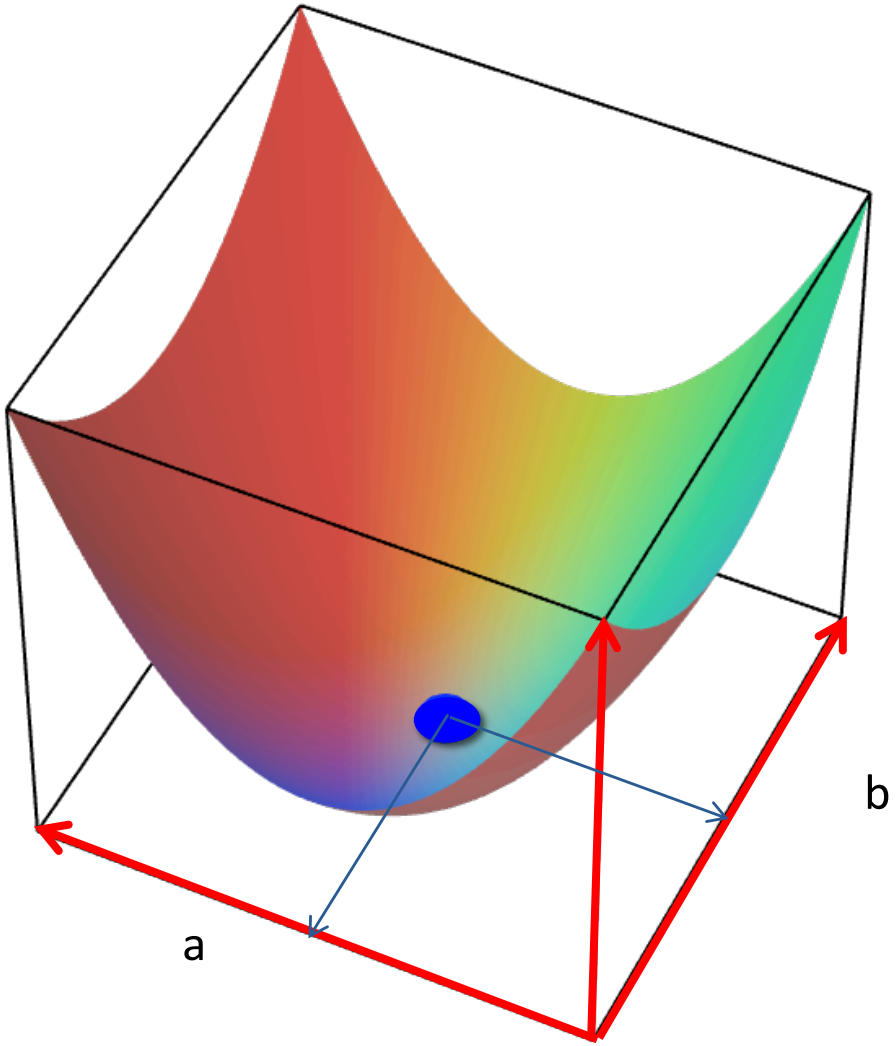
$$\sum_{i=0}^{len(observed)-1} (observed[i] - a * x[i] - b)^2$$

polynomial $ax + b$ is predicting *y* values for all the *x* values in our experiment, such that sum squared difference of **predicted** values and corresponding **observed** values is minimized

- A **linear regression** problem
- Many algorithms for doing this, including one similar to Newton's method (which you saw in 6.0001)
  - You could write your own …

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
  - Means we don't need to compute the entire surface explicitly, but just start at a point, compute the value of the objective function there, determine the downhill direction, and move in that direction
- Objective surface in this case has a global minimum
  - Means there is always a **unique** best fit

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

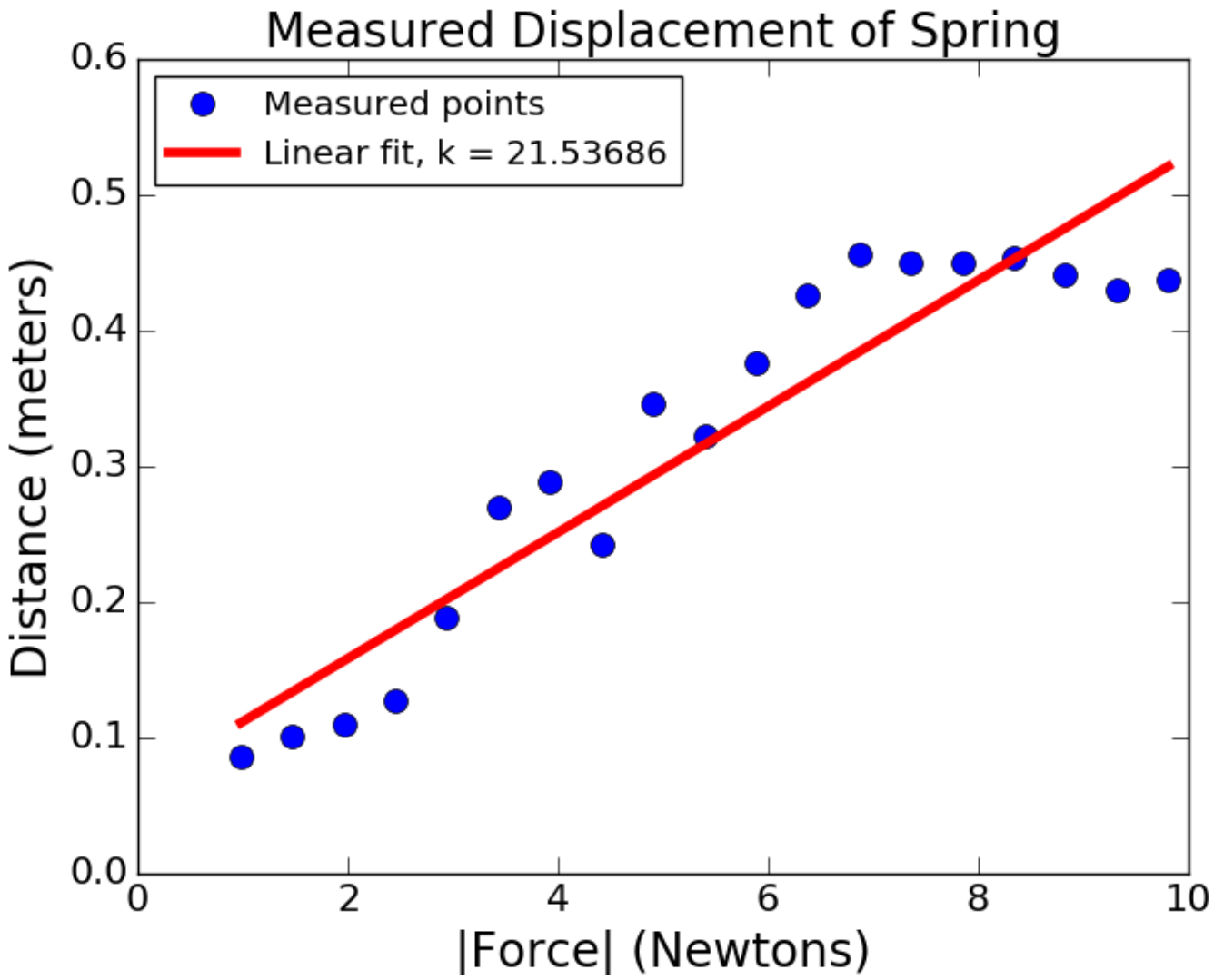
```
def fitData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    a,b = pylab.polyfit(xVals, yVals, 1)
    estYVals = 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')
```

plotData

Note that conversion to array is redundant here

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

Visualizing the Fit



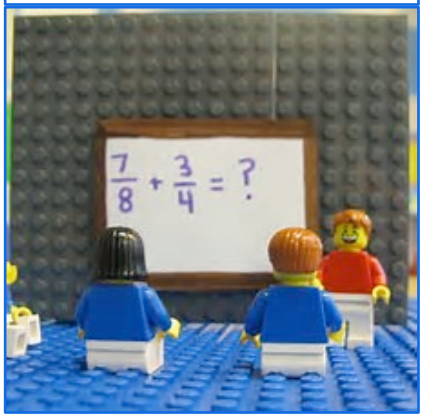
Version Using polyval

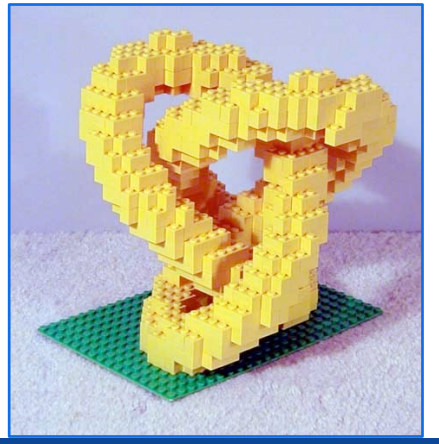
```
def fitData1(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    model = pylab.polyfit(xVals, yVals, 1)
    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 fit model to xVals for any order of model

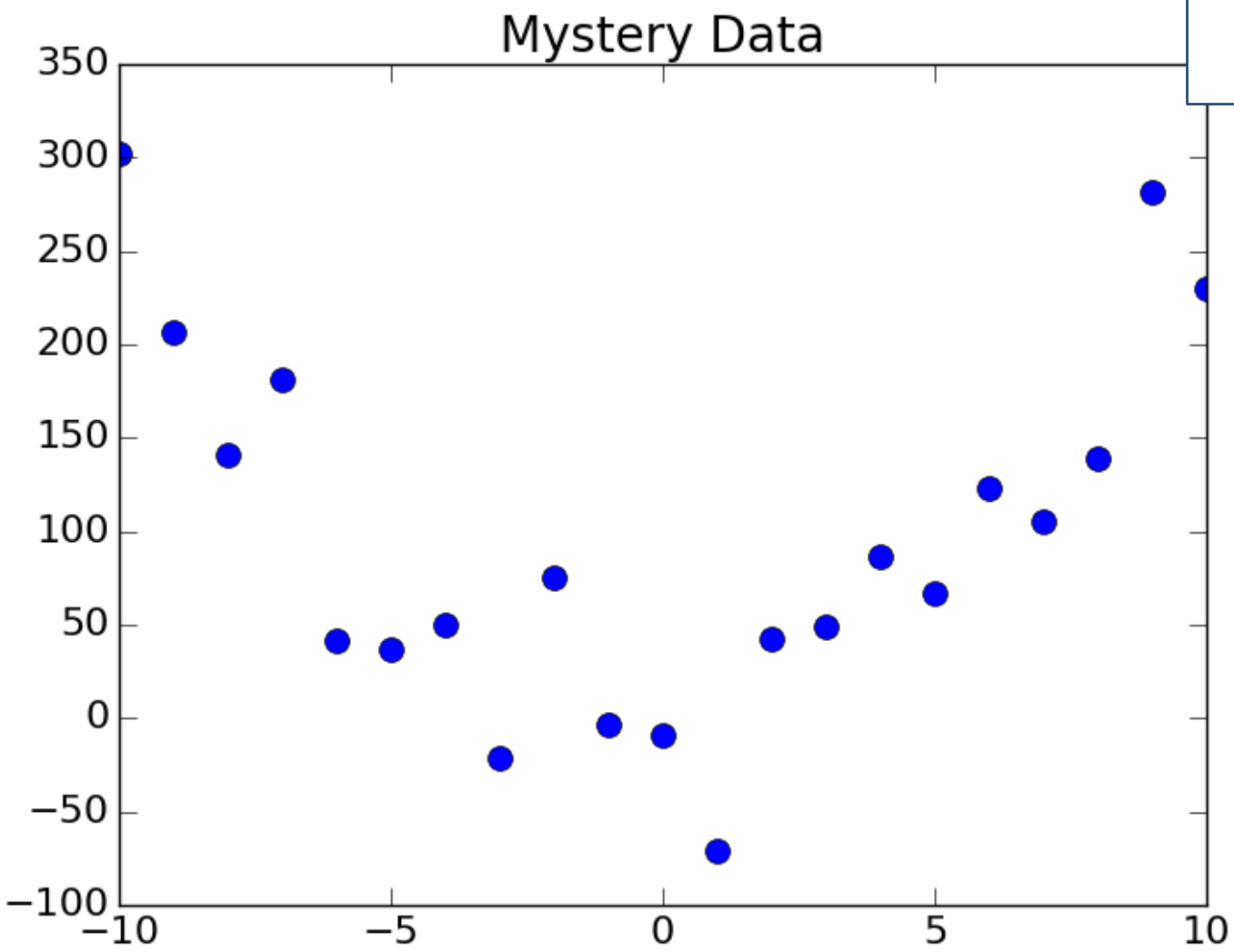
Quick Summary So Far

- 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 minimizes sum squared error between observed and predicted values
- So, let's apply this idea to other data…

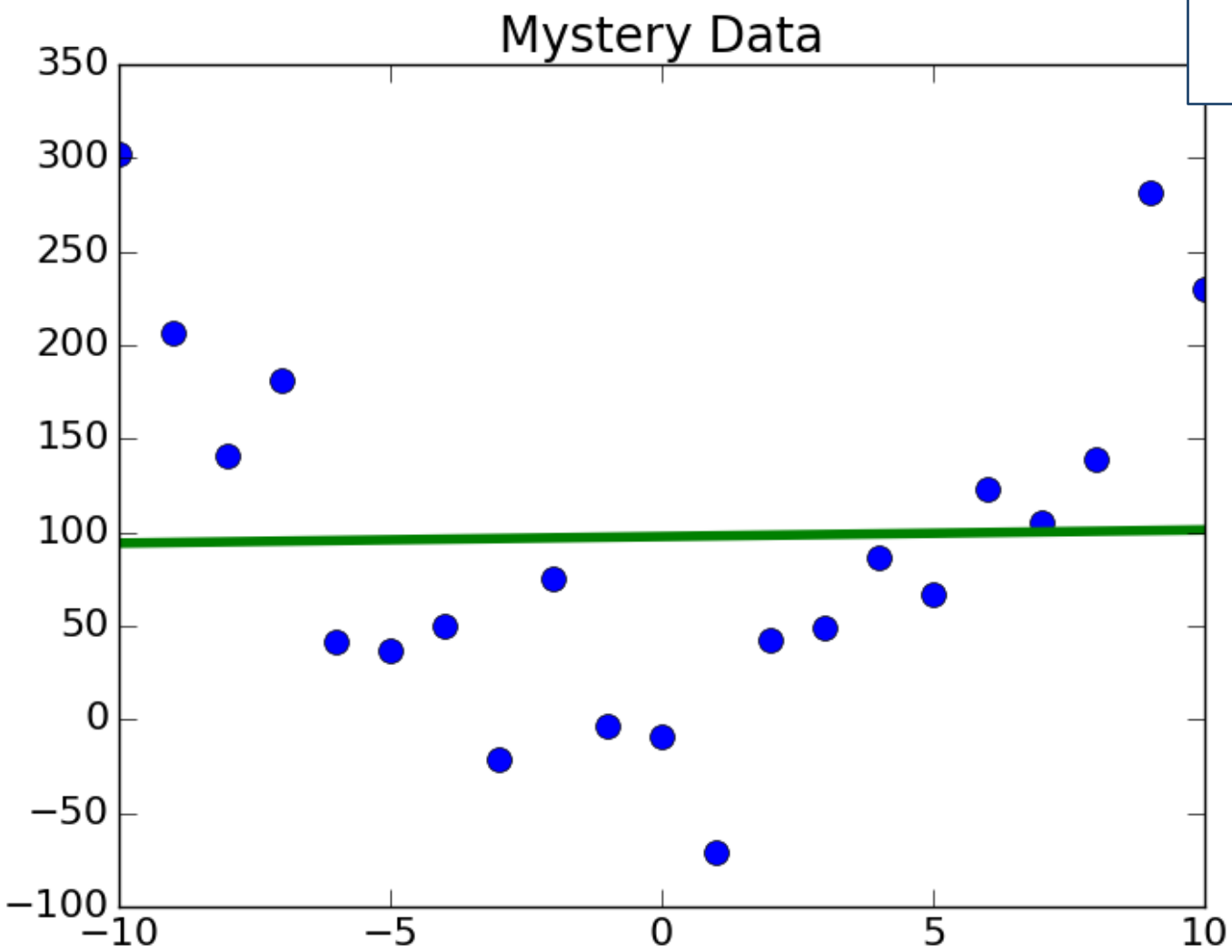




Another Experiment



Fit a Line



Let's Try a Higher-degree Model

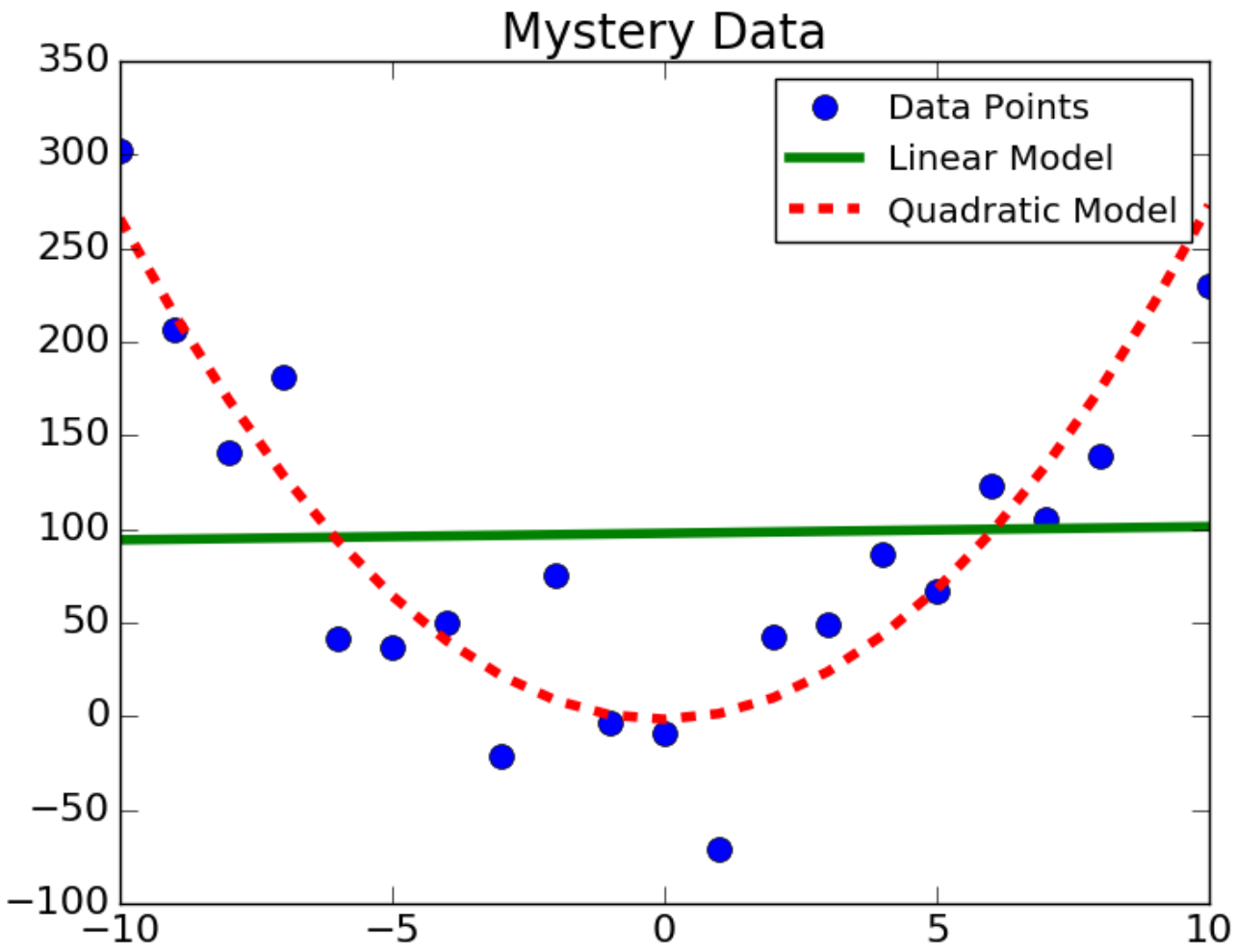
```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

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)

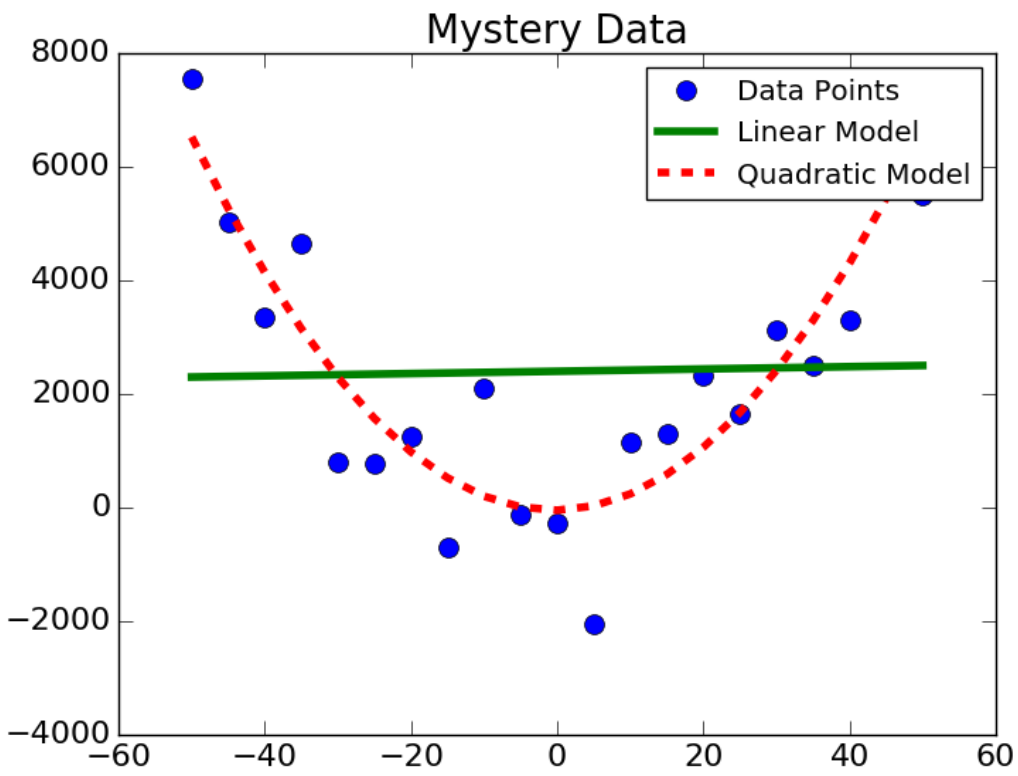
- Objective function depends linearly on unknowns, which are the coefficients of the polynomial

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

Quadratic Appears to be a Better Fit



How Good Are These Fits?



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

Relative to Each Other

- Fit is a function from the independent variable to the dependent variable
- Given an independent value, provides an estimate of the dependent value
- Which fit provides better estimates?
- Since we found fit by minimizing mean square error, could just evaluate goodness of fit by looking at that error
  - Know this is also minimizing variance between predicted and measured values

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?

Giovanni Verrazzano discovers New York Bay

- Hard to know – no bound on values; not scale independent
  - For example, if we double the masses, get double the error
- 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

If You Prefer Code

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

```
def rSquared(observed, predicted):
    error = ((predicted - observed)**2).sum()
    meanError = error/len(observed)
    return 1 - (meanError/numpy.var(observed))
```

I am playing a "clever trick" here:

- Numerator is sum of squared errors
- Dividing by number of samples gives average sum-squared-error
- Denominator is variance times number of samples
- So mean SSE/variance is same as R^2 ratio

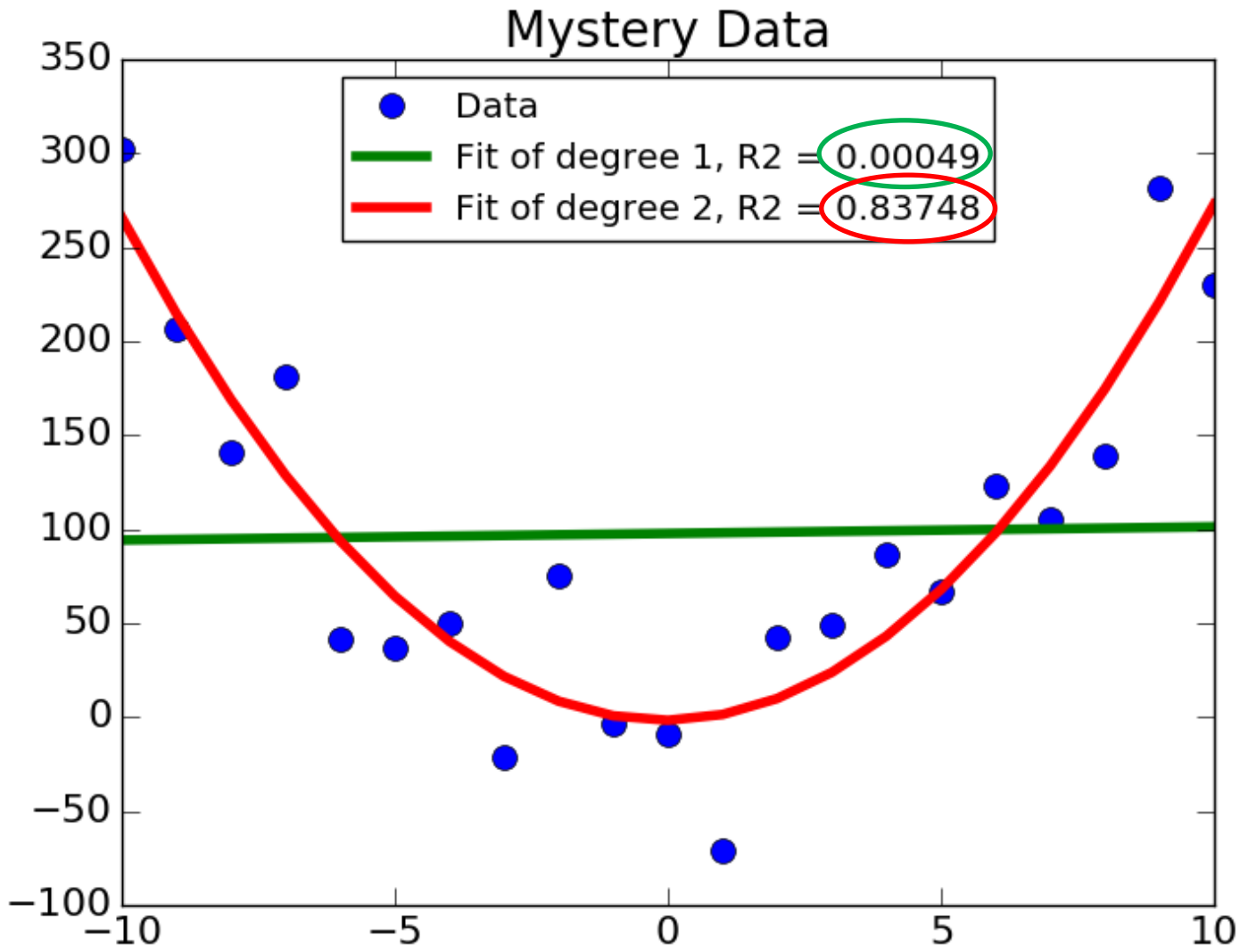
R^2

- By comparing the estimation errors (the numerator) with the variability of the original values (the denominator), R^2 is intended to capture the proportion of variability in a data set that is accounted for by the statistical model provided by the fit
- Always between 0 and 1 when fit is generated by a linear regression and is 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.
  - If $R^2 = 0.5$, the model explains half the variability in the data.

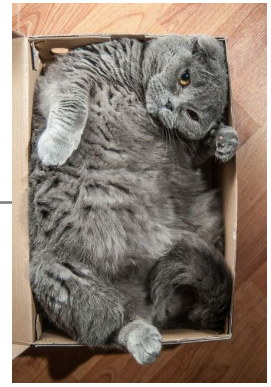
Testing Goodness of Fits

```
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?

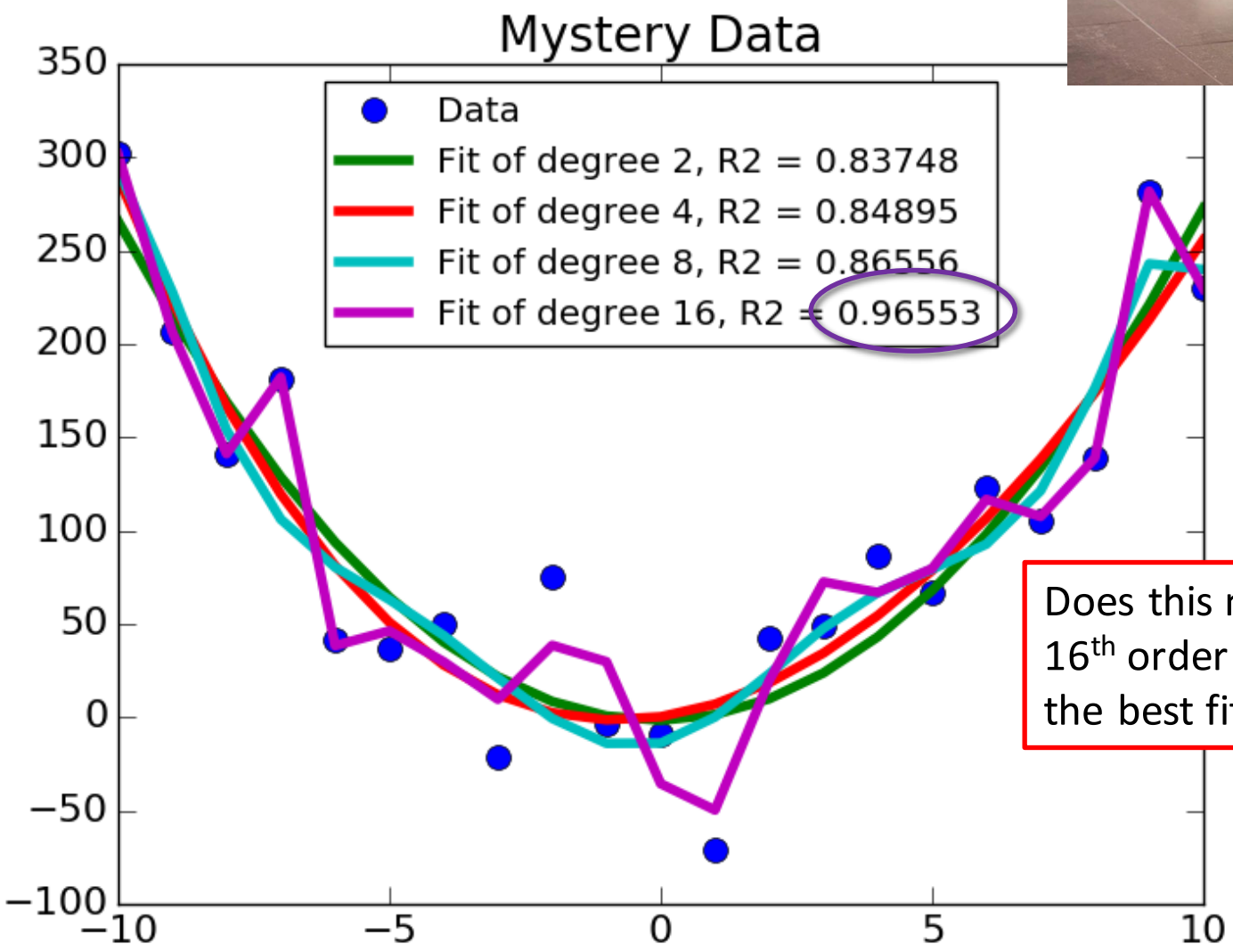


Can We Get a Tighter Fit?



- 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?

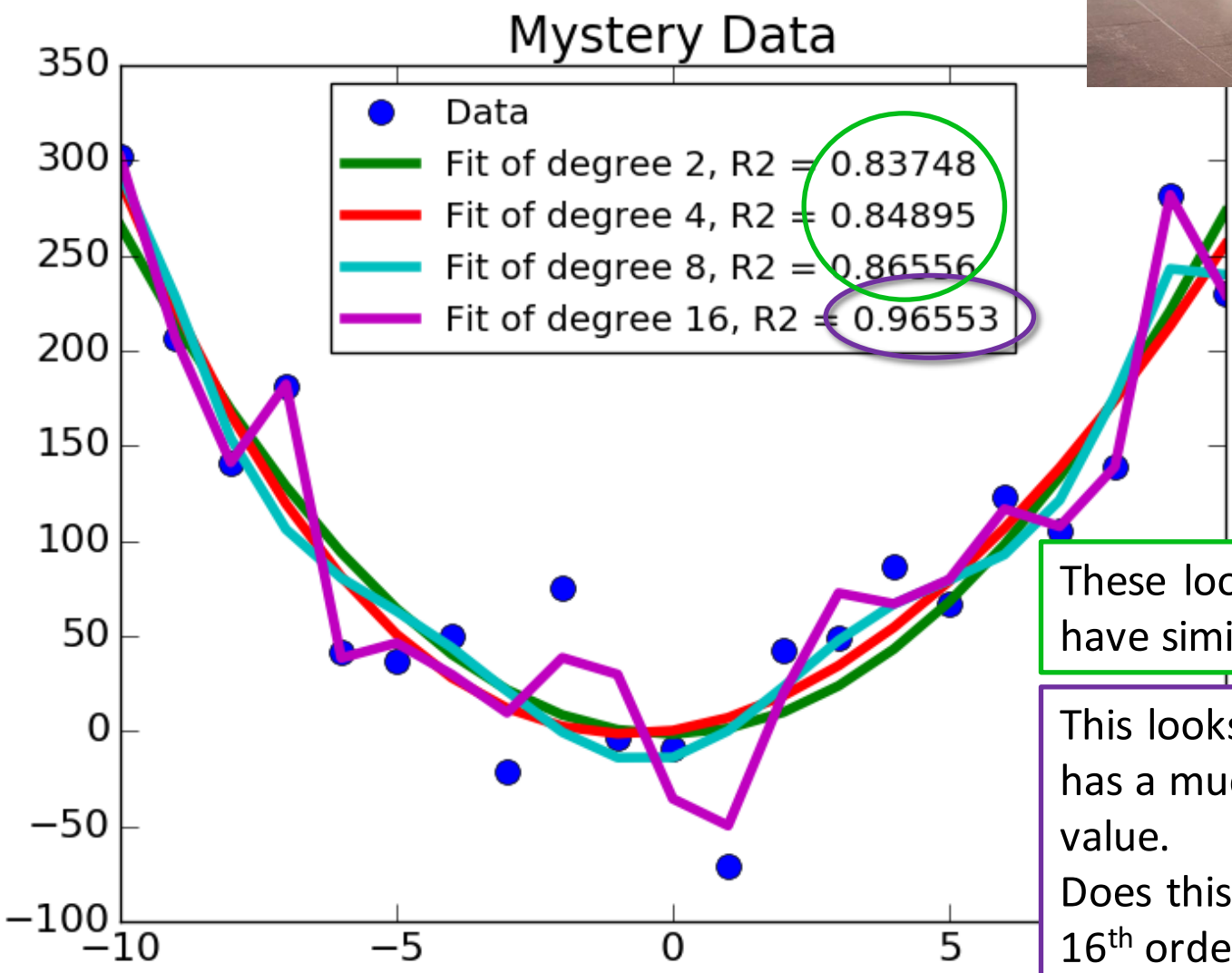
Can We Get a Tighter Fit?



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

Five Minute Break

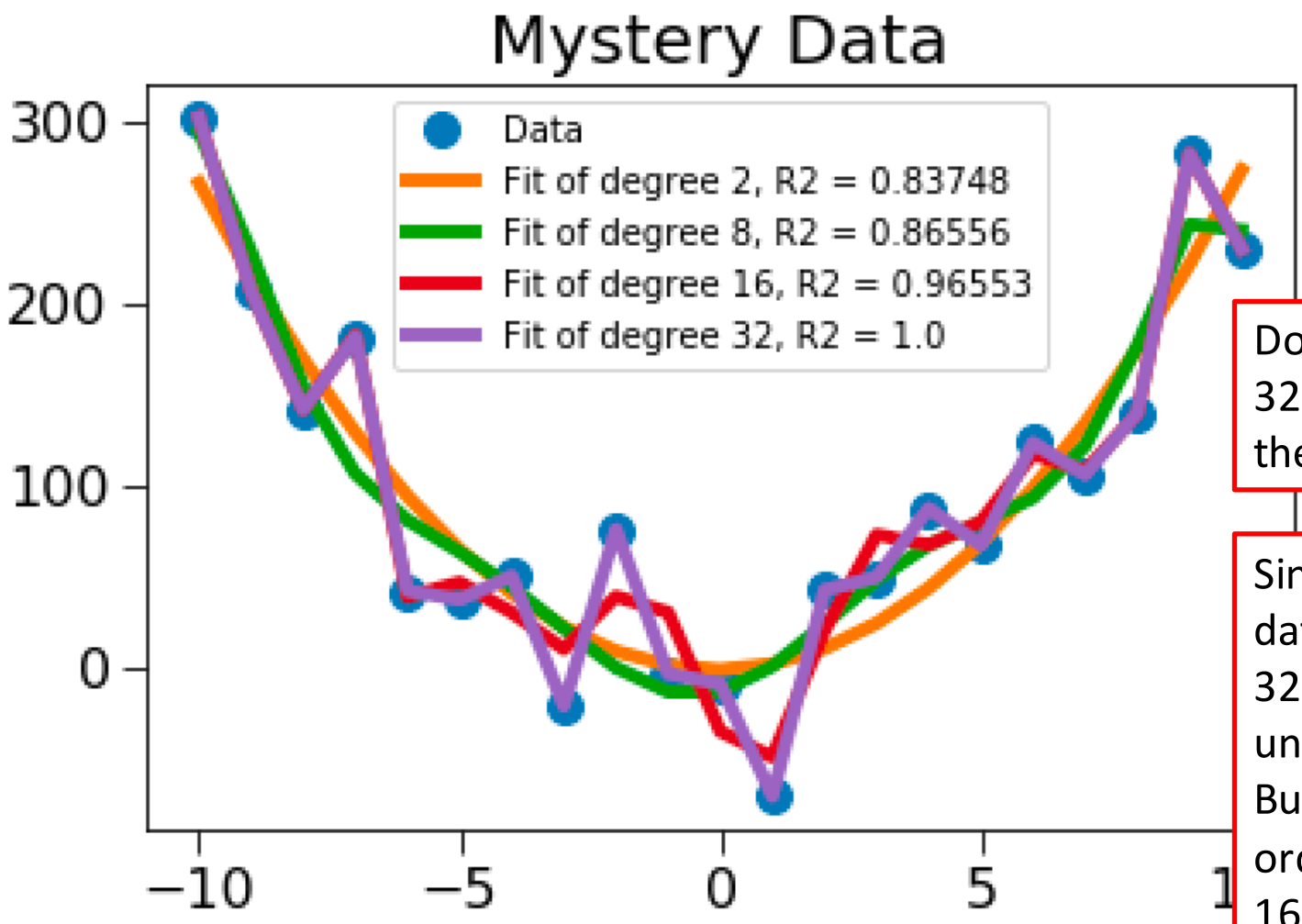
Can We Get a Tighter Fit?



These look similar, and have similar R2 values

This looks different and has a much better R2 value.
Does this mean that a 16th order polynomial is the best fit for the data?

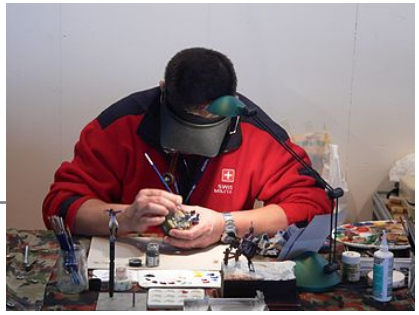
Is There a Tighter Fit at 32?



Does this mean that a 32nd order polynomial is the best fit for the data?

Since there are only 21 data points, fitting a 32nd order polynomial is undoubtedly **overfitting.** But what is the right order to use? 2? 4? 8? 16?

Why We Build Models



- 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?**
- Help us understand process that generated the data
  - E.g., the properties of a particular linear spring
  - E.g., springs follow the theoretically predicted linear behavior
- 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

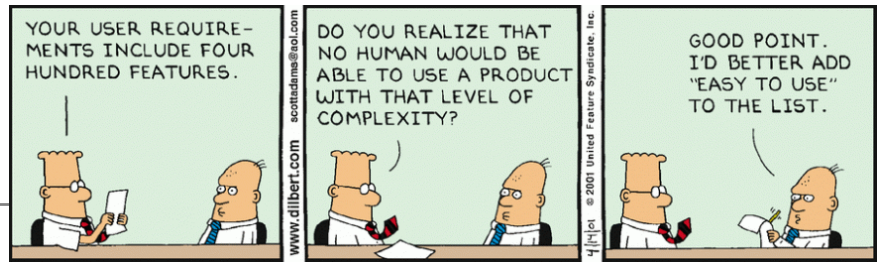

```
def genNoisyParabolicData(a, b, c, xVals, fName):
    yVals = []
    for x in xVals:
        theoreticalVal = a*x**2 + b*x + c
        yVals.append(theoreticalVal + random.gauss(0, 35))
    f = open(fName,'w')
    f.write('x          y\n')
    for i in range(len(yVals)):
        f.write(str(yVals[i])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

Increase 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 ever get a worse fit to training data when doing this?
- If extra term is useless, coefficient will simply 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

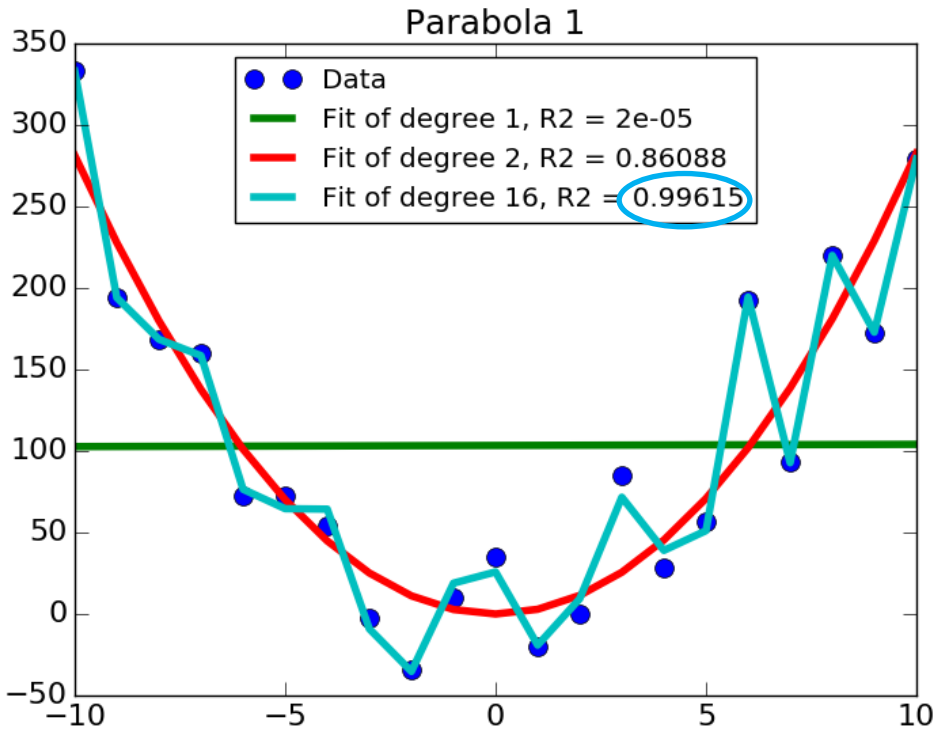
Training versus Testing

- 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 that data 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 data

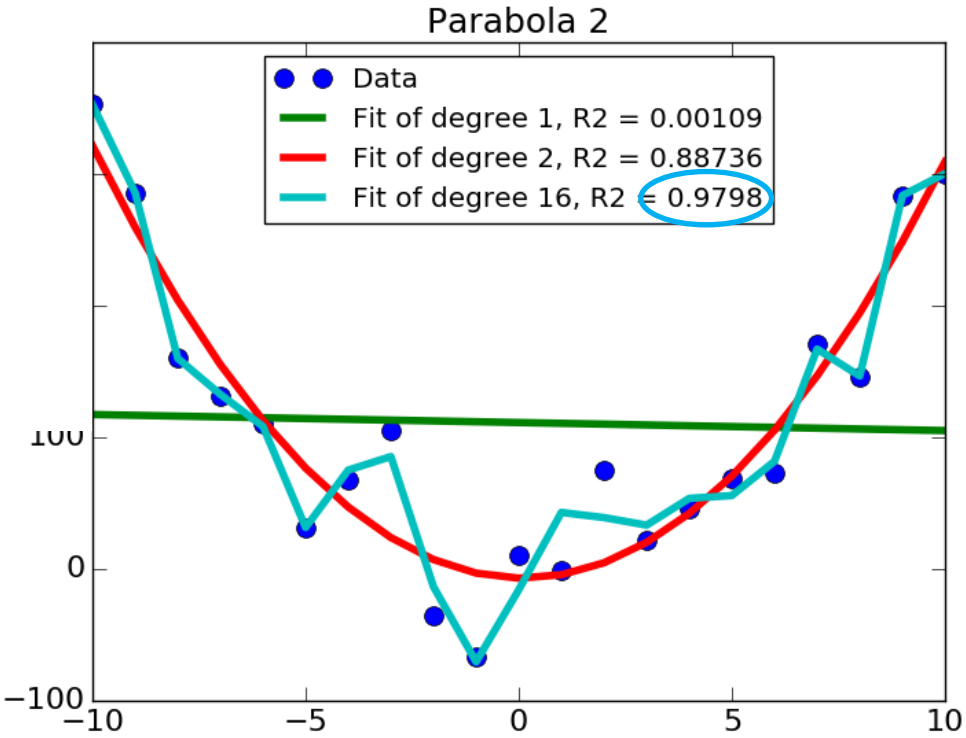
Generate 2 Data Sets from Same Distribution

```
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

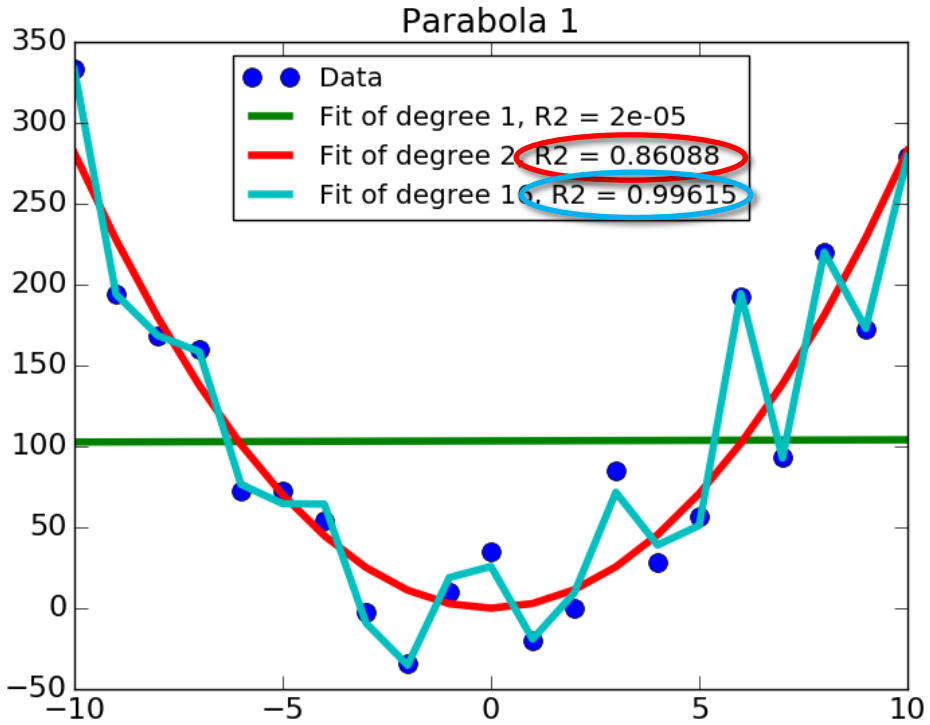


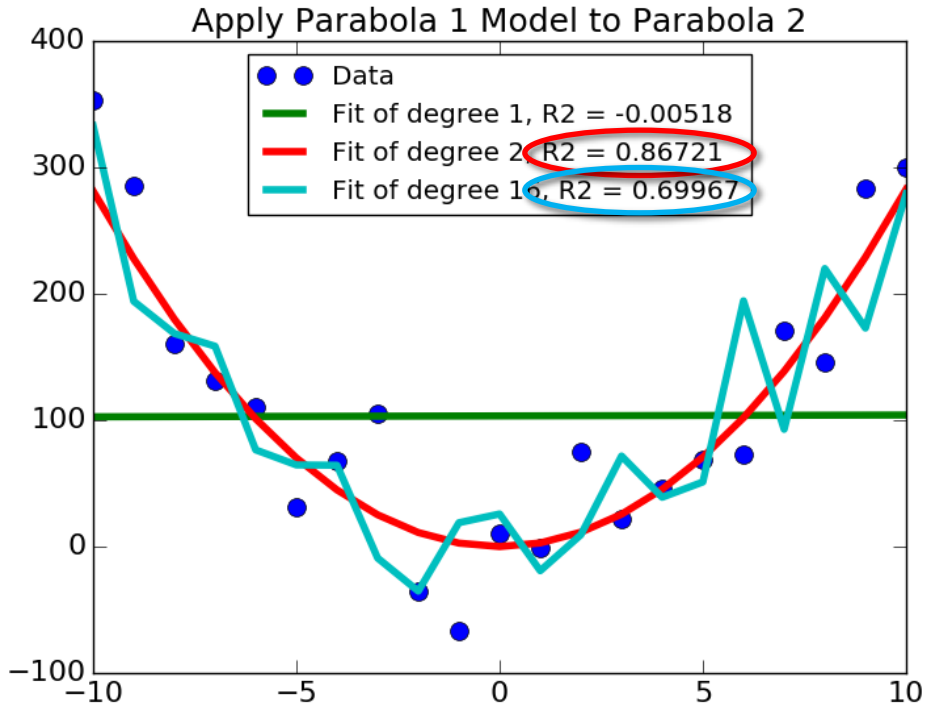
Each model is a good fit to its training data



Training and Testing Errors

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

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

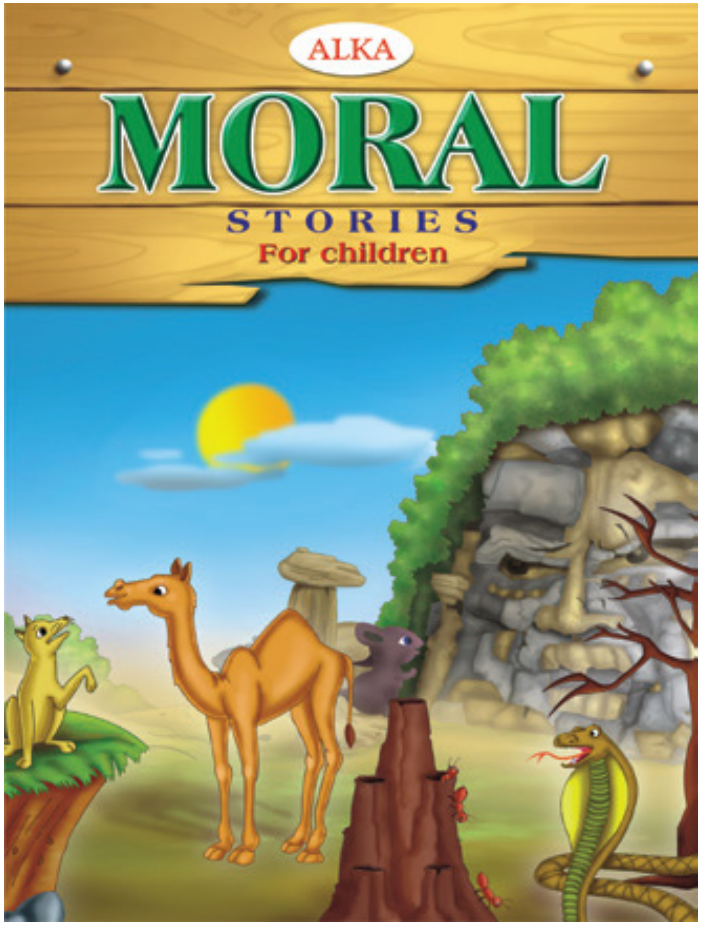




Order 2 model accounts for test data well;
but order 16 model does not account for test data

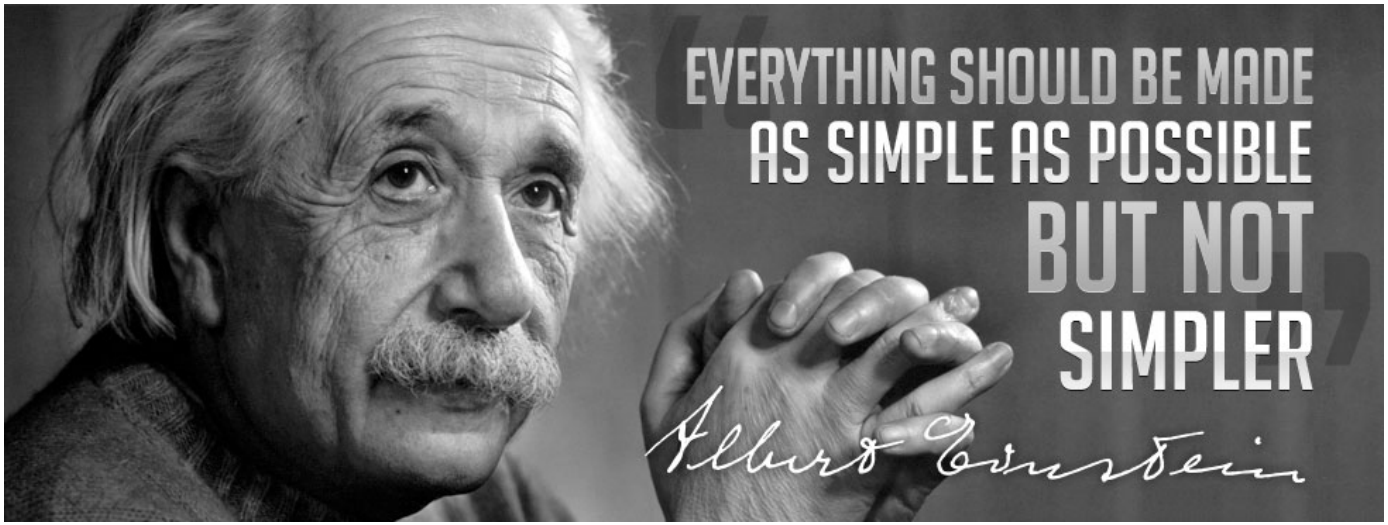
The Moral of the Story

- 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



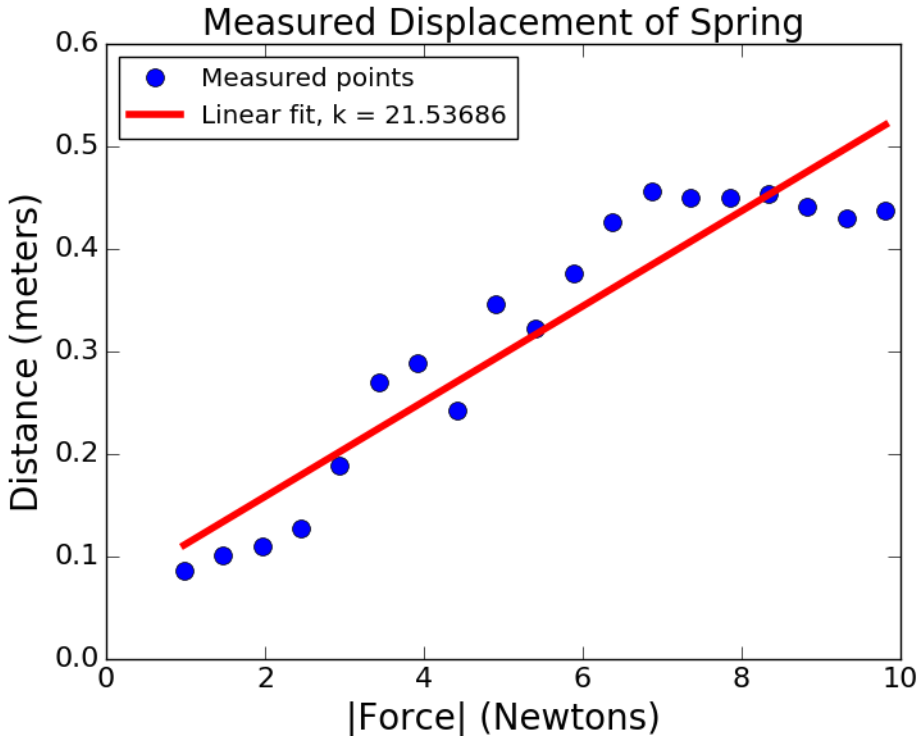
The Take Home Message

- 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



One Final Thought

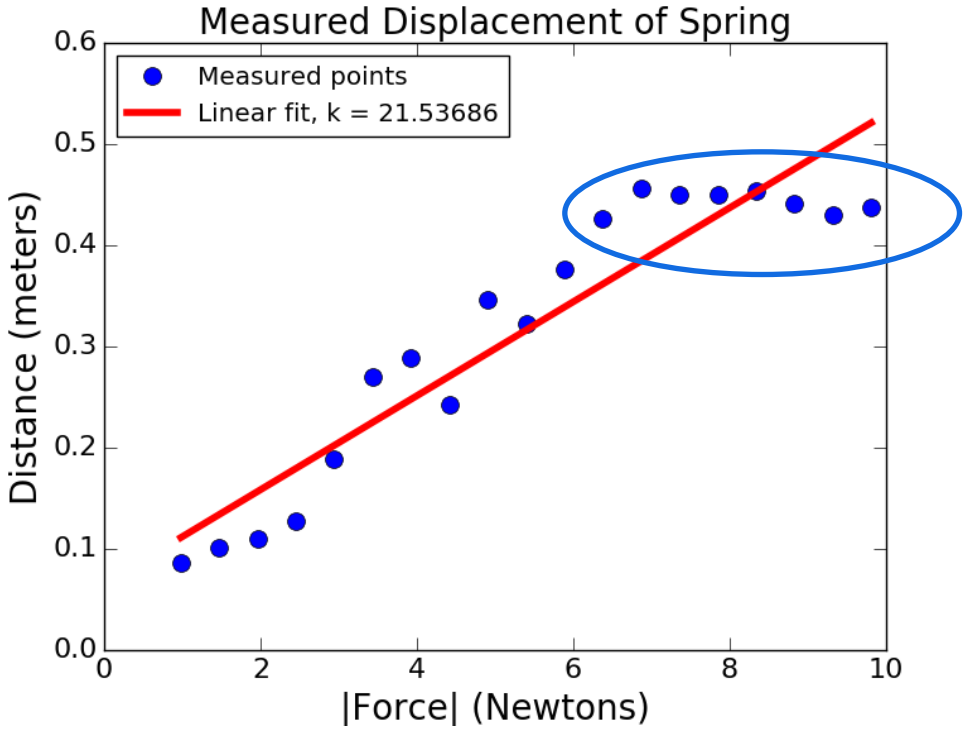
- Combining model information with goodness of fit can provide additional insight
- Consider the fit to the original spring data



- R^2 value is .8815 – which is decent

One Final Thought

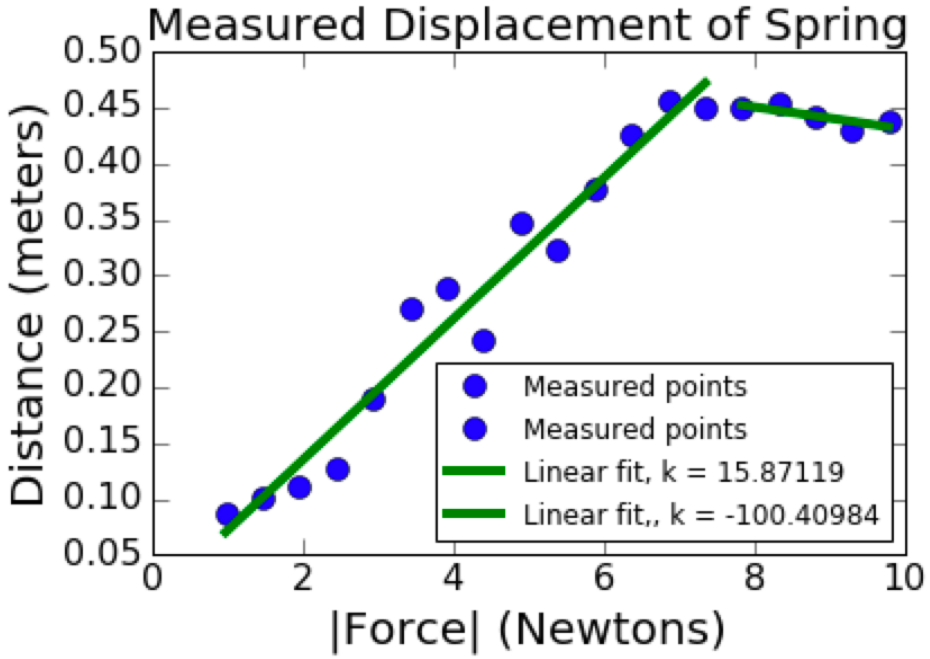
- But visual inspection suggests that something is happening for large forces?



- Remember model said Hooke's law applied up to some maximum force

One Final Thought

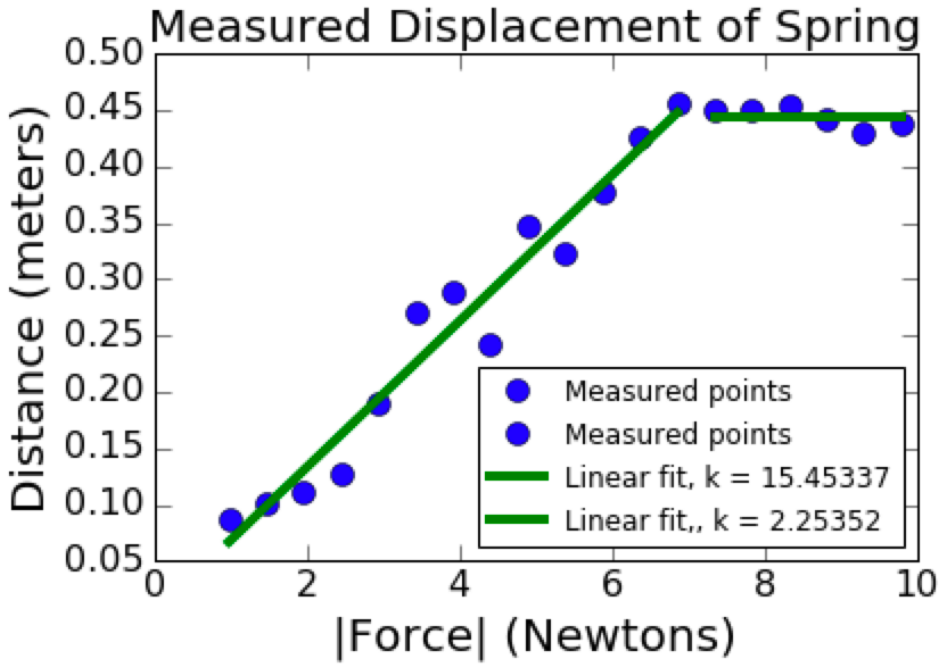
- Perhaps better to search for point at which to break data into two sets, and fit models to both sets of data separately; look for break that minimizes sum of residual error in both parts?



- R^2 value for first part now .9581; for second part .6784; without break, have R^2 of .8815

One Final Thought

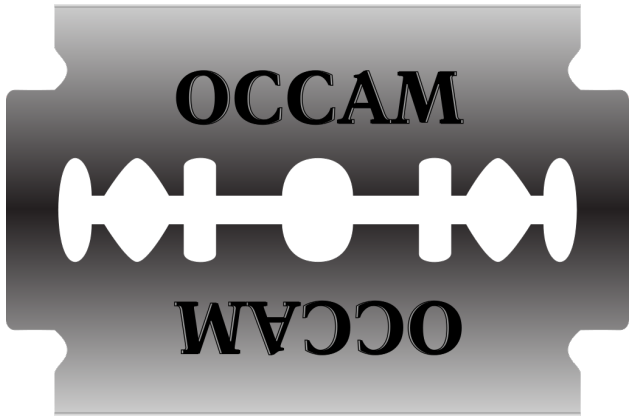
- Alternatively, search for point at which to break data into two sets, and fit model to first set of data but fit constant line to second set; look for break that minimizes sum of residual error in both parts



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

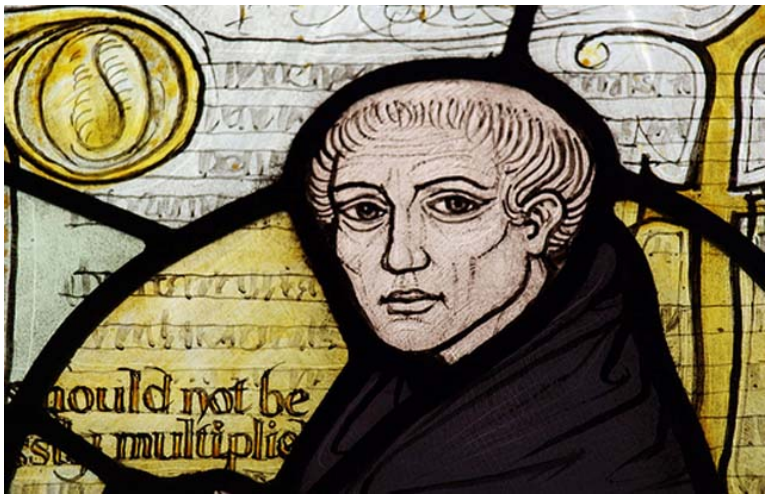
Wrapping Up Curve Fitting

- 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