

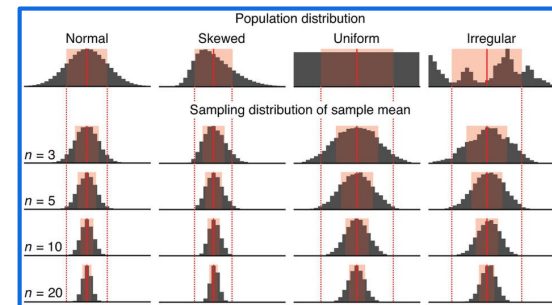
Understanding Experimental Data

Ana Bell

MIT Department of Electrical Engineering and
Computer Science

Where Have We Been?

- Using Monte Carlo simulation to build models
 - The world is mostly stochastic, so draw multiple samples
 - Useful tool even when randomness not present
 - Estimate reliability of simulation results
- Understanding populations
 - Cannot examine all members
 - Rely on sampling to infer information
 - 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 variables and properties (e.g., mean) of distributions



Statistics Meets Experimental Science

- Conduct an experiment to gather 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
- All the time remembering that the data will be noisy!
- Let's look at a spring

One Kind of Spring



Another Kind of Spring



This Kind of Spring



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

$$k \approx 1 \text{ N} / \text{m}$$



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

Hooke's Law

- 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

- $F = kd$
- How much does a rider have to weigh to compress spring 1cm?

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

$$F = 350N$$

$$mass * 9.8m/s^2 = 350N$$

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

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

$$mass \approx 35.68kg$$

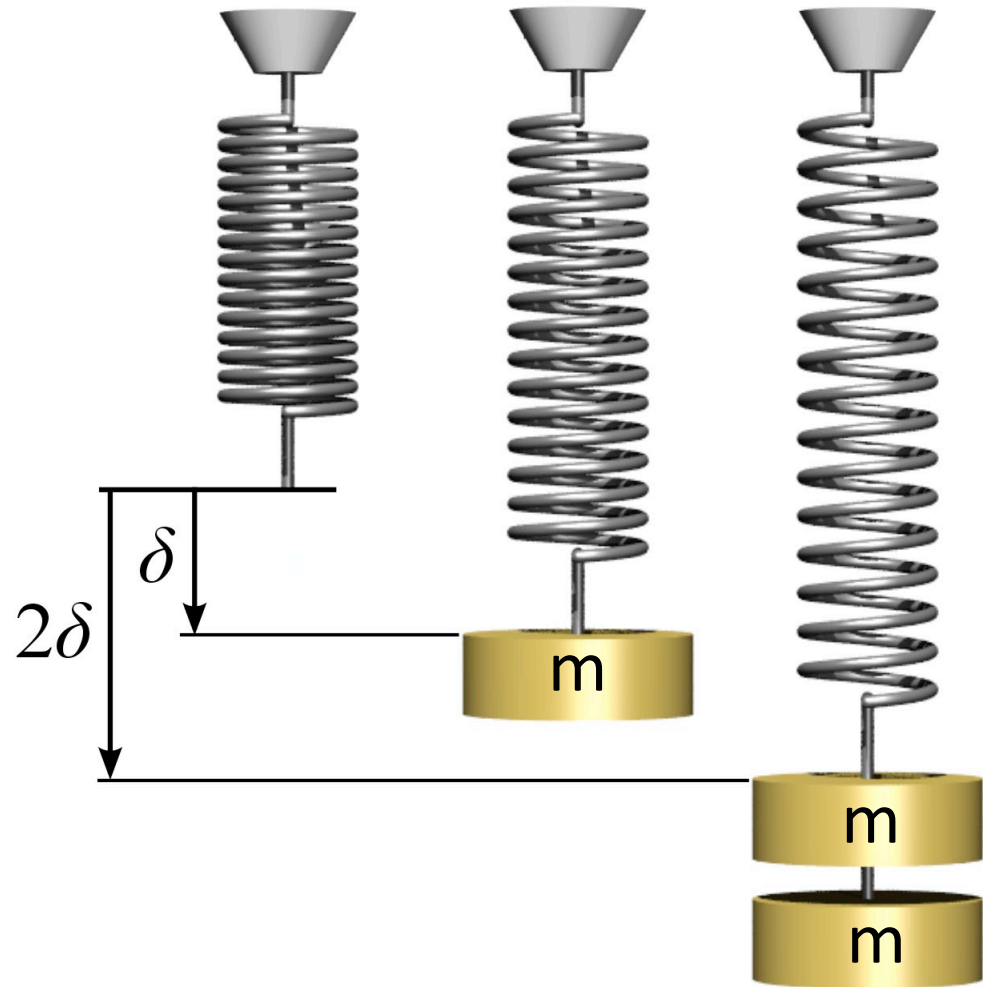
$$F = mass * acc$$

$$F = mass * 9.8m/s^2$$



Finding k

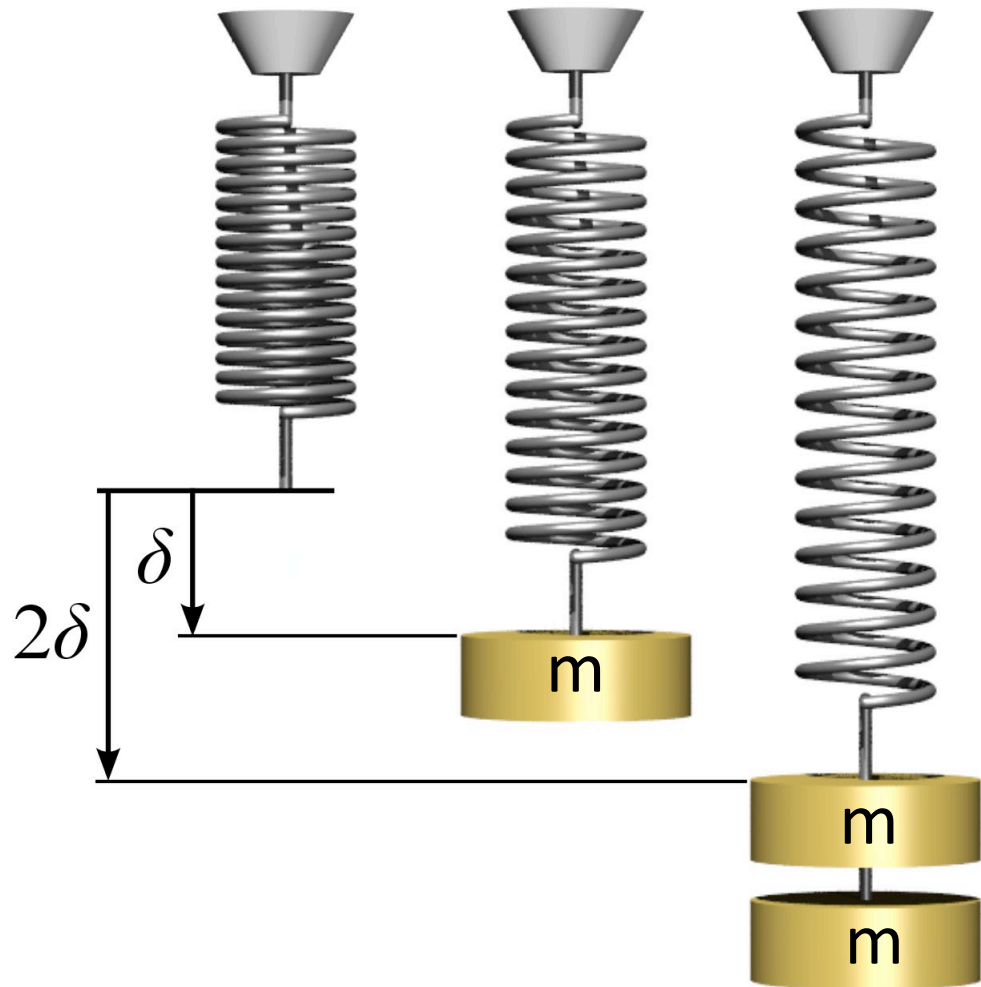
- $F = k\delta$
- $k = F/\delta$
- $k = 9.81 * m/\delta$



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

Some Data

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

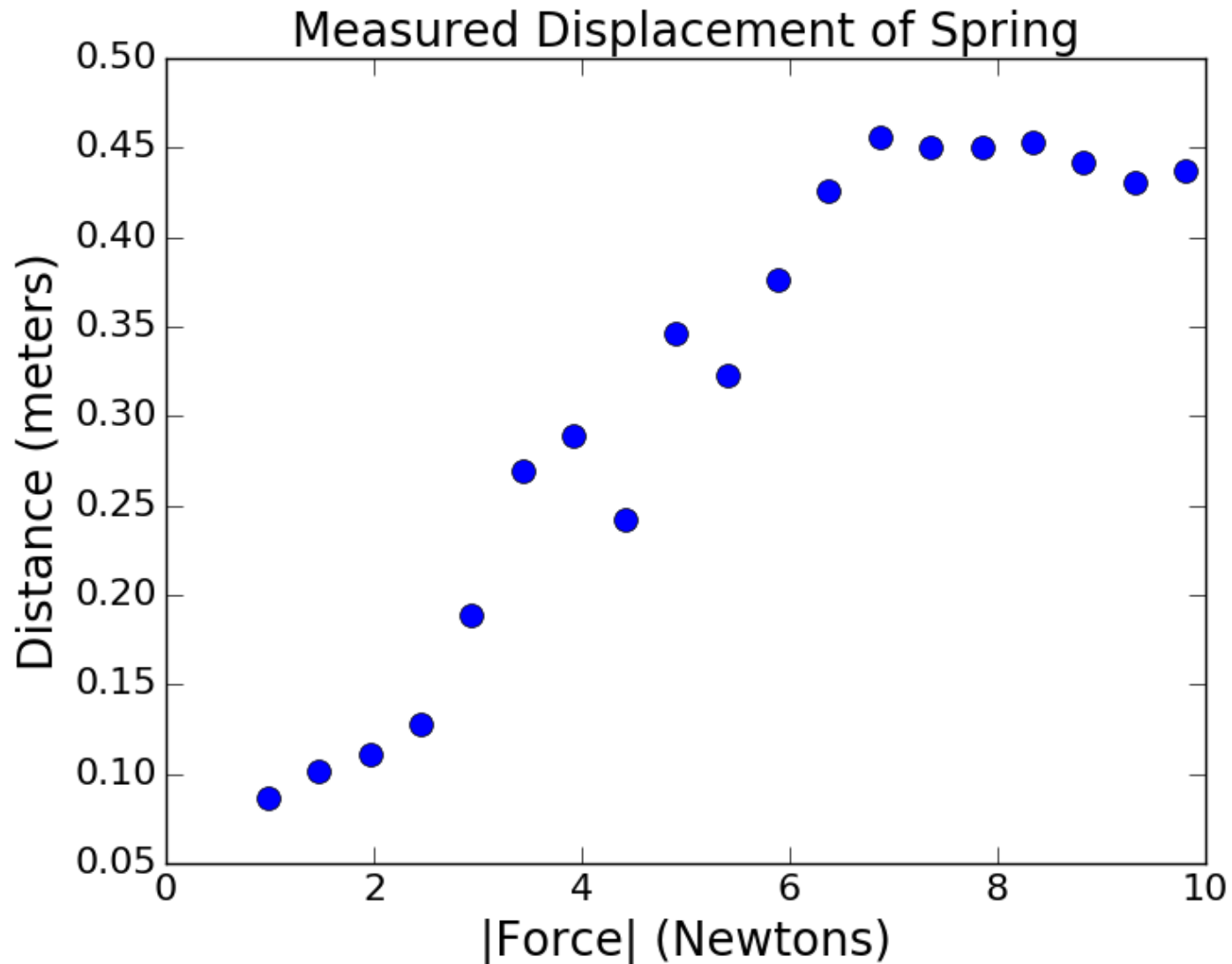


```
def plotData(fileName):  
    xVals, yVals = getData(fileName)  
    xVals = np.array(xVals)  
    yVals = np.array(yVals)  
    xVals = xVals*9.81 #acc. due to gravity  
    plt.plot(xVals, yVals, 'bo',  
             label = 'Measured displacements')  
    labelPlot()
```

A reminder about numpy arrays:

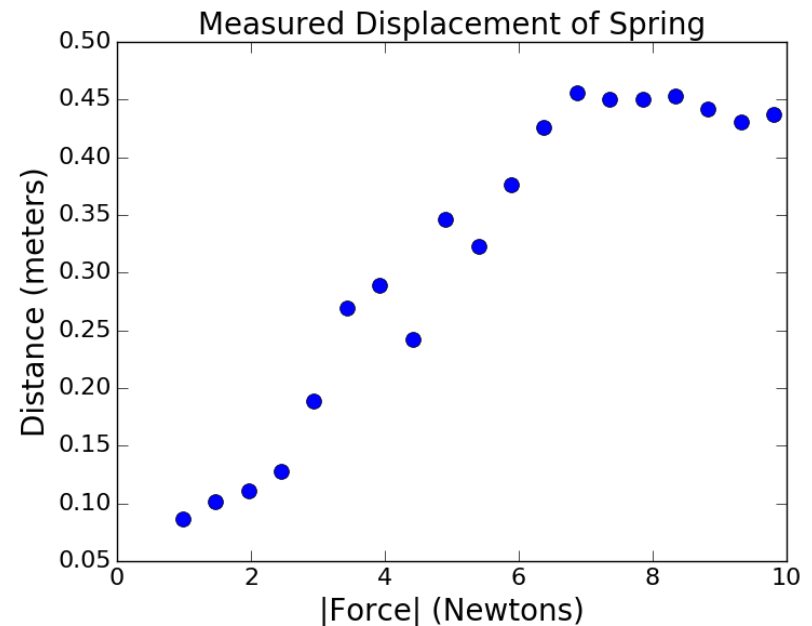
- Converts a list into a linear data structure
- Can treat arrays algebraically; e.g., if a and b are arrays, then:
 - $a*2$ multiplies each element of a by 2
 - $a + 3$ adds 3 to each element of a
 - $a - b$ subtracts each element of b from corresponding element of a
 - $a*b$ multiplies each element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

Taking a Look at the Data



What Can We Do With This Data?

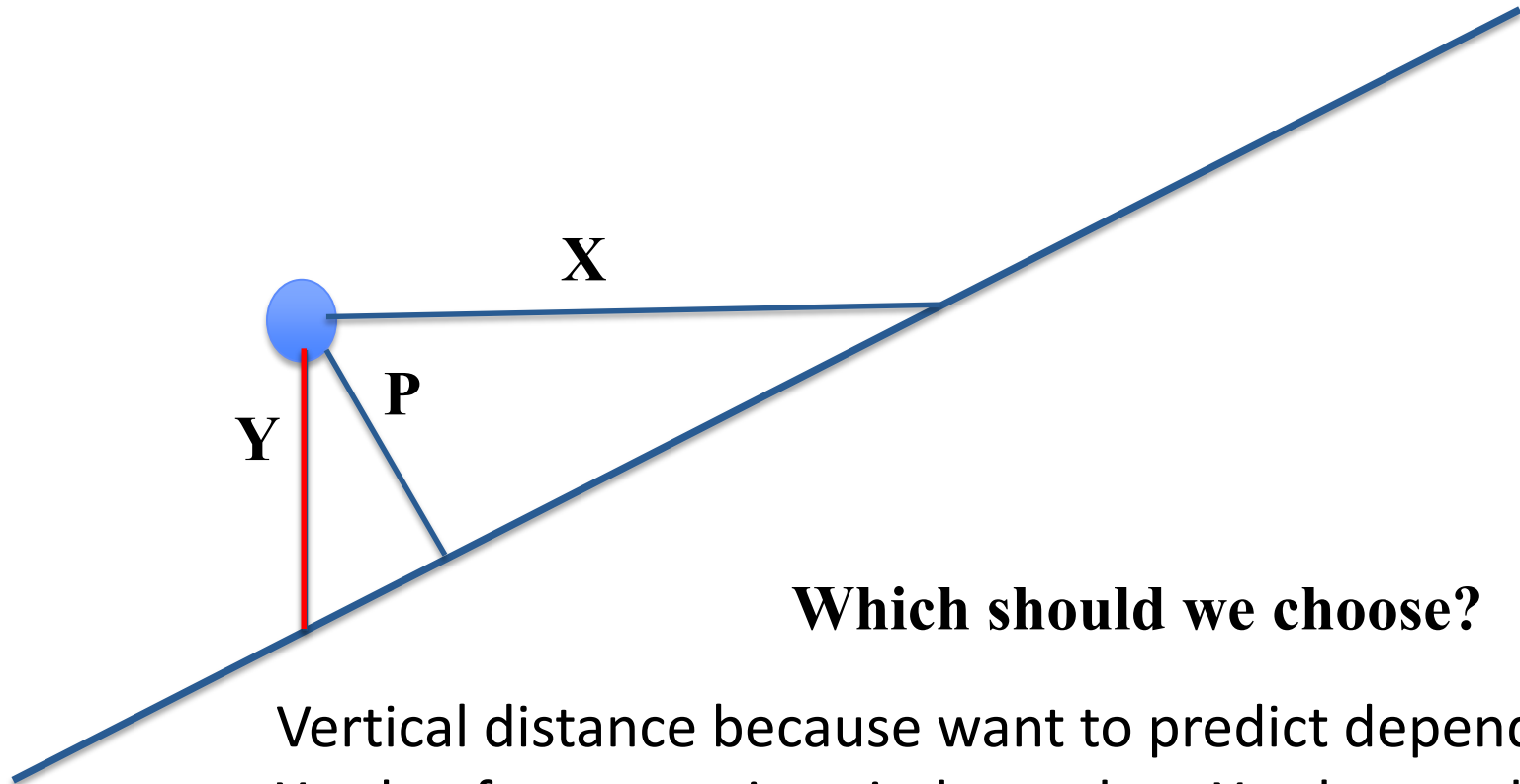
- We've run an experiment
- We can relate observations to measurements (distance d vs. force F)
- Theory predicts a relationship between observations and measurements ($F = -kd$)
- Can we use these measurements to determine k and to validate model?
- Notice that points don't lie on a line. Experiments are noisy!



Fitting Curves to Data

- When we **fit** a curve 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**
- Once we define the objective function, we also need an algorithm to find the curve that minimizes it
- Theory says find a **line** such that some function of the distances from the **line** to the measured points is minimized. The line that best fits the data.

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

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

- Look familiar?
 - This is variance times number of observations

So minimizing this will also minimize the variance

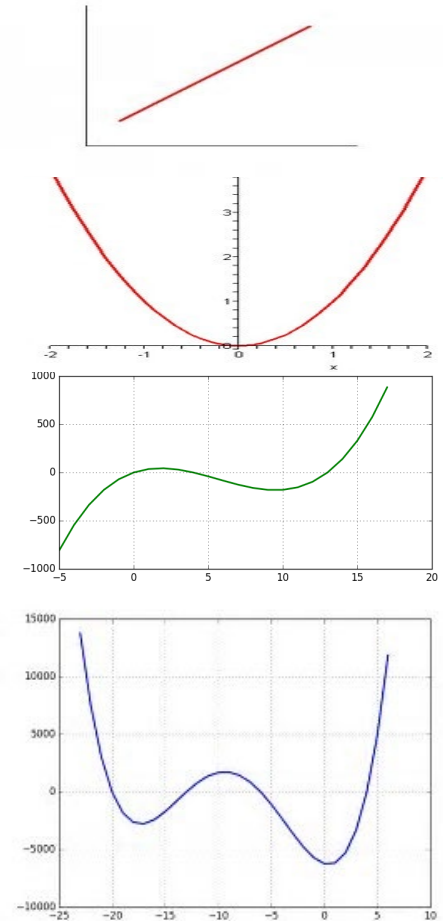
Solving for Least Squares

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

- To minimize this objective function, want to find a curve for the **predicted** observations that leads to minimum value of sum of squared differences
- Need to make a choice on kinds of curves – we will use polynomials of one variable
- 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 of the term, a non-negative integer
- The degree of the polynomial is the largest degree of any 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

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

- 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

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

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

Solving for Least Squares

- For linear case

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

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

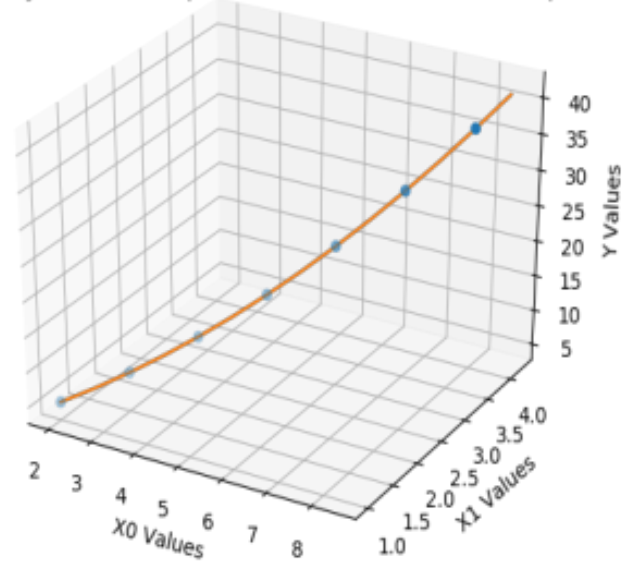
- A **linear regression** problem
- Many algorithms for doing this, including one similar to Newton's method (which you saw in 6.0001)

You Could Write Your Own

Least Squares with Polynomial Features Fit using Pure Python without Numpy or Scipy



Pure Python Least Squares Fit with Two 2nd Order Inputs



<https://integratedmlai.com/least-squares-with-polynomial-features-fit-using-pure-python-without-numpy-or-scipy/>

polyFit

- Good news is that numpy provides a builtin function to find these polynomial fits

- `np.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 $y = ax^3 + bx^2 + cx + d$

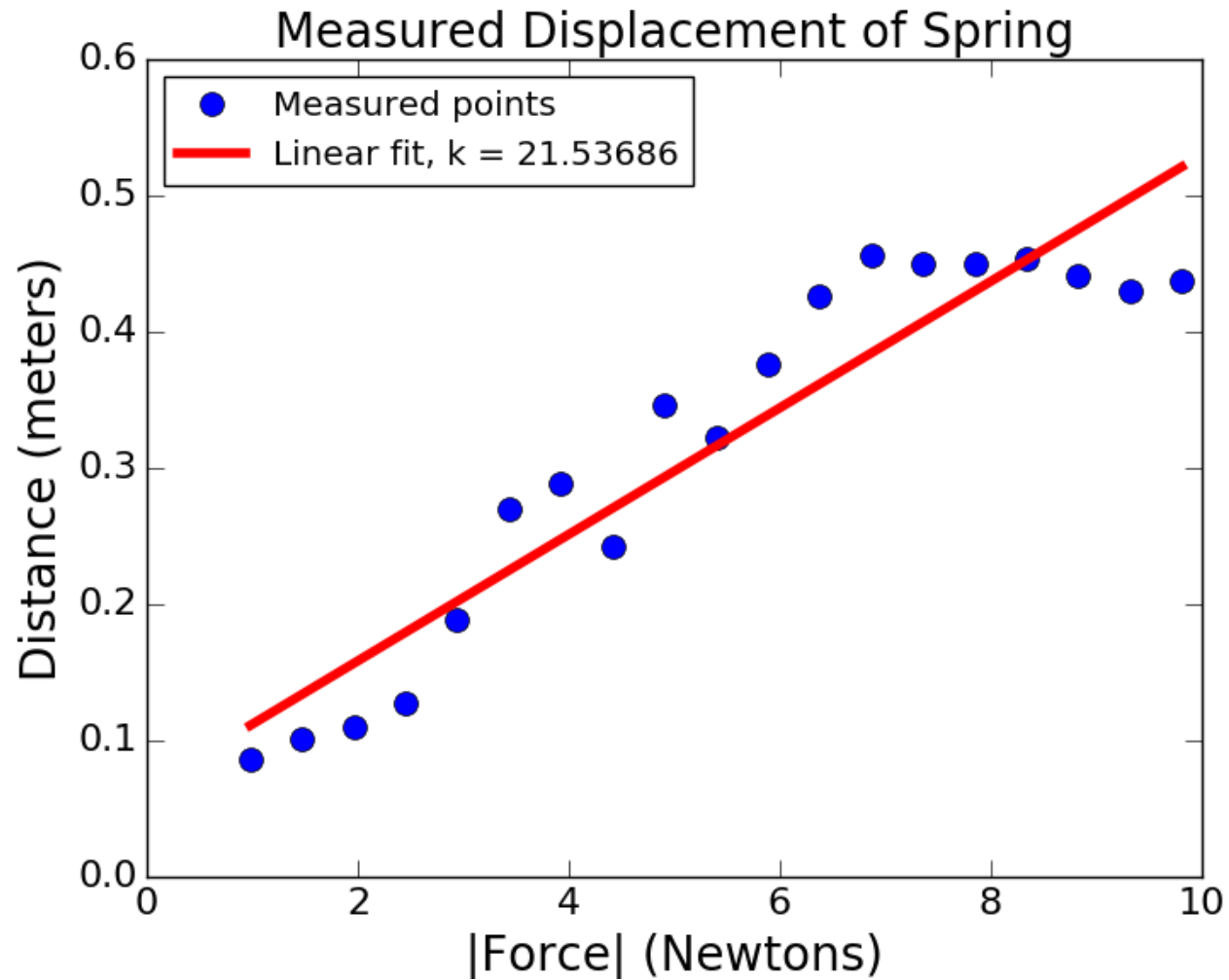
Using polyfit

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

plotData

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

Visualizing the Fit



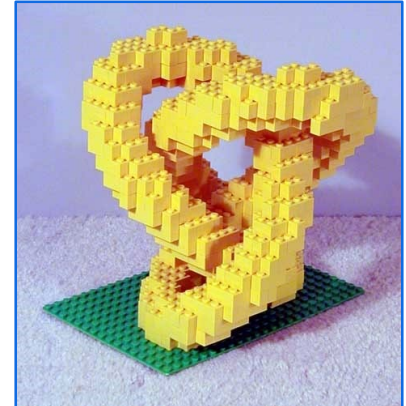
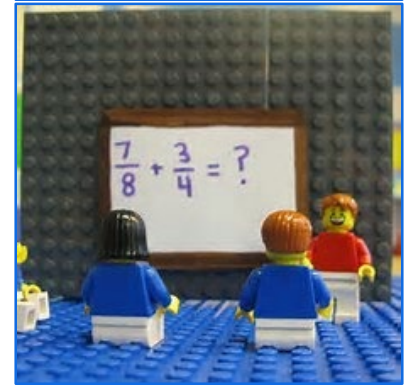
Version Using polyval

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

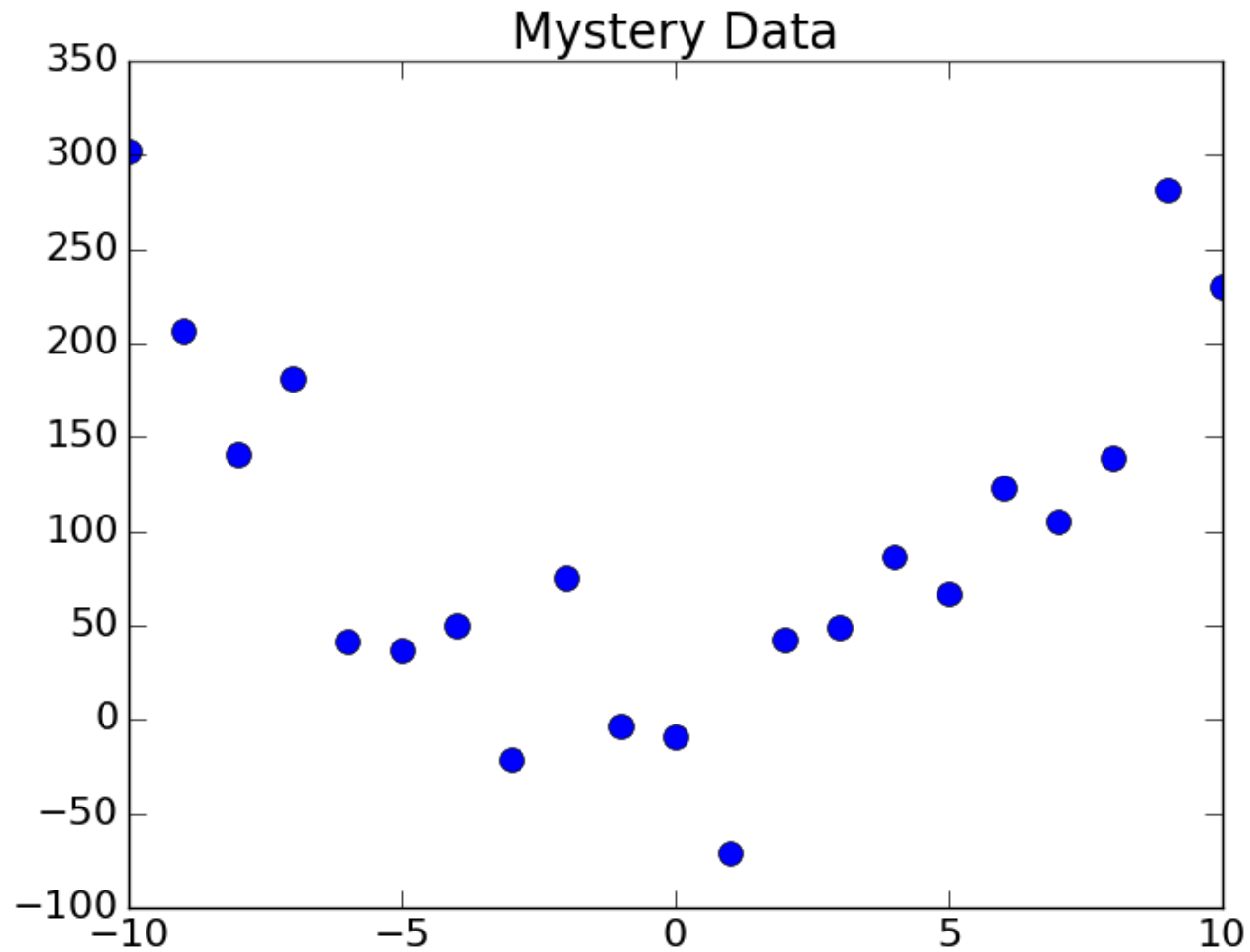
polyval will
fit model
to xVals for
any order
of model

Quick Summary So Far

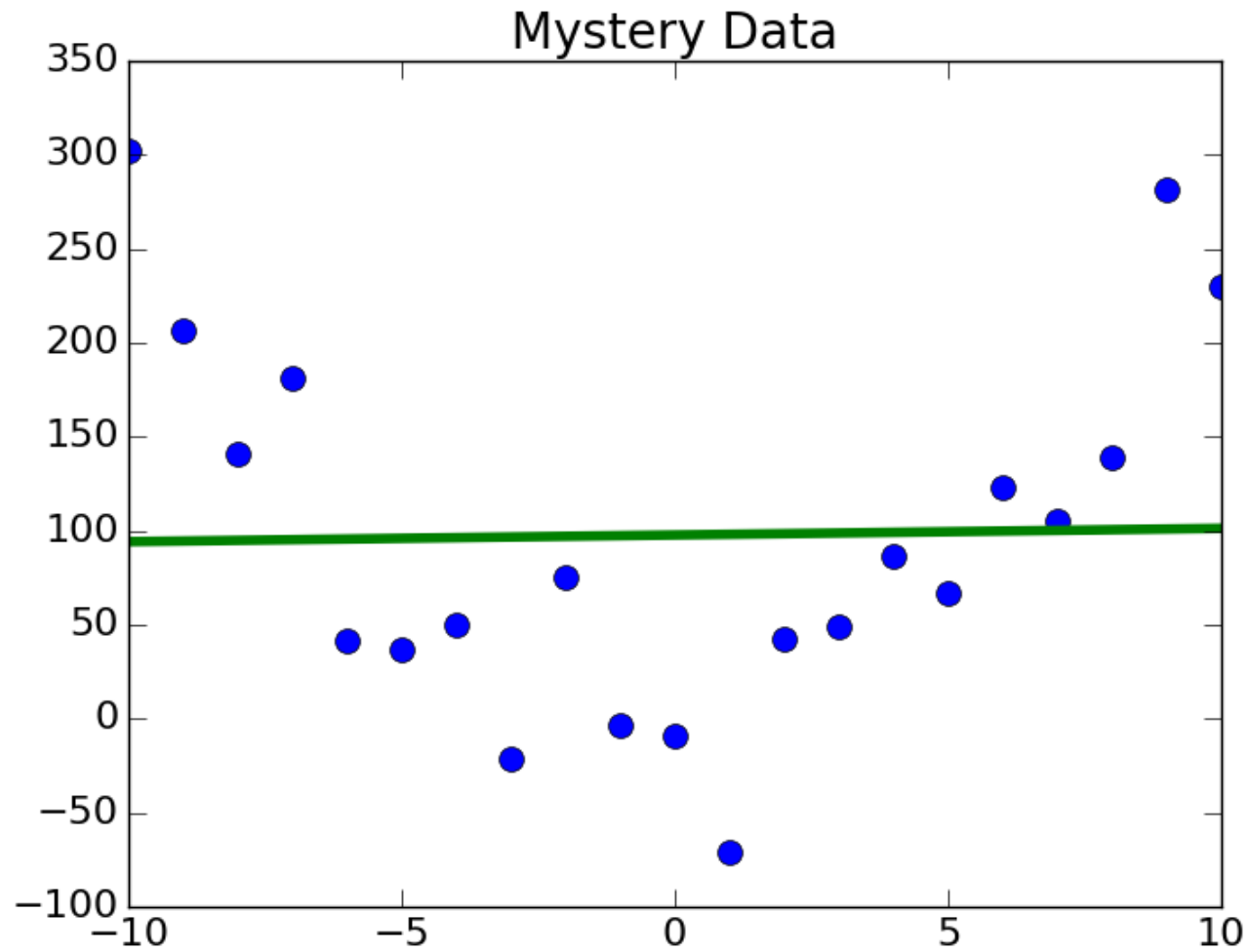
- Ran an experiment to gather data
- Theory predicts relationship between measurements (displacements) and experimental parameters (masses or forces)
- Linear regression lets us fit best model (line in our case) to observed data
 - Best here means minimize sum squared error between observed and predicted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



Another Experiment



Fit a Line



Let's Try a Higher-degree Model

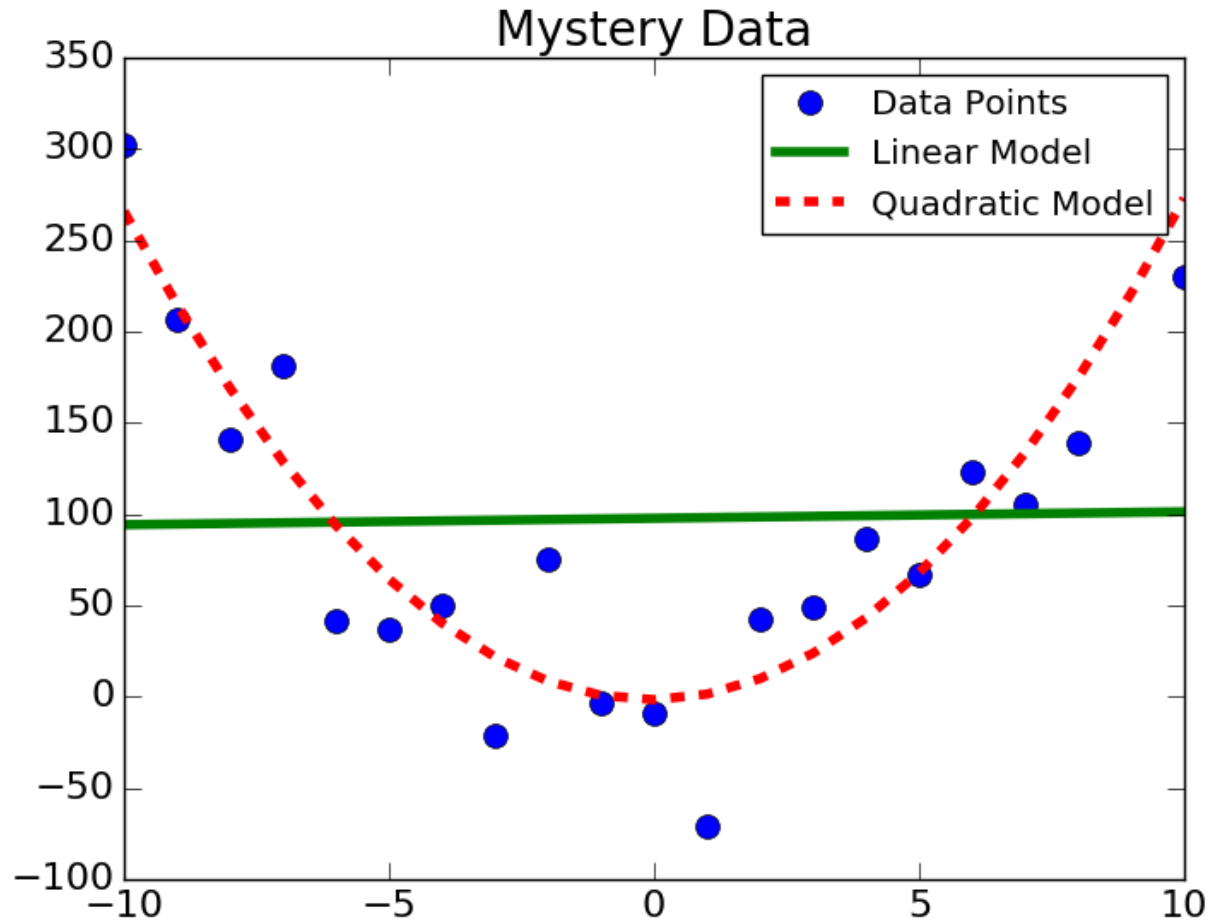
```
model2 = np.polyfit(xVals, yVals, 2)
plt.plot(xVals, np.polyval(model2, xVals),
         'r--', label = 'Quadratic Model')
```

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)

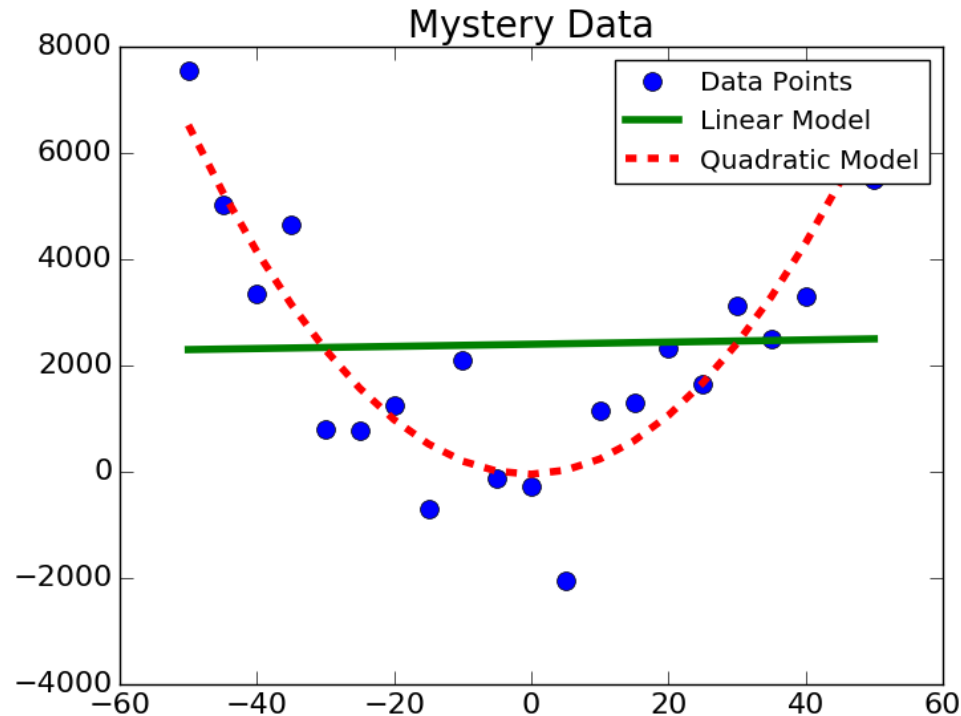
- Objective function depends linearly (additively) on unknowns, which are the **coefficients** of the terms of the polynomial

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

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

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

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
- 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/np.var(observed))
```

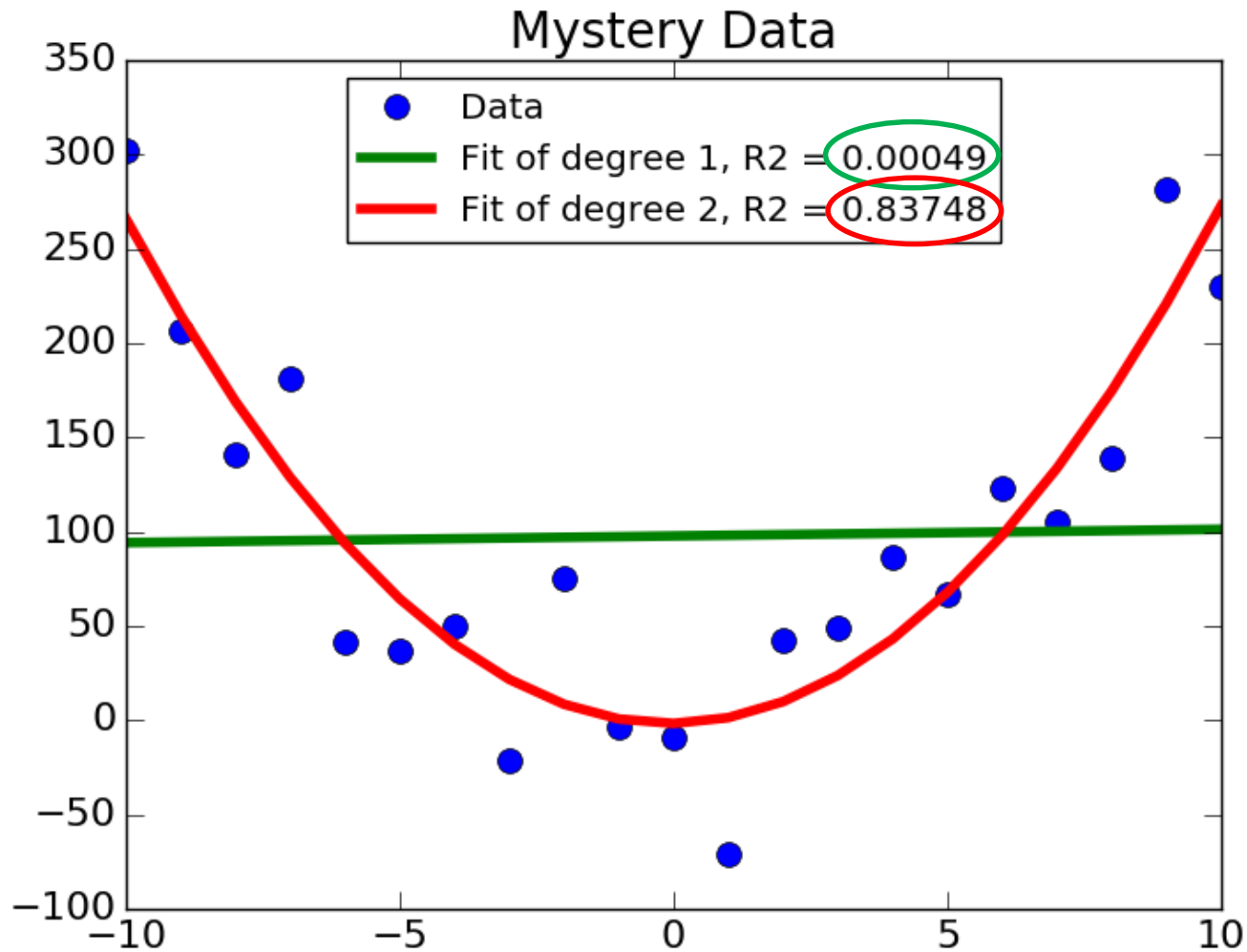
- Numerator is sum of squared errors
- Dividing by number of samples gives average sum-squared-error
- Denominator is variance times number of samples
- So mean SSE/variance is same as R^2 ratio

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

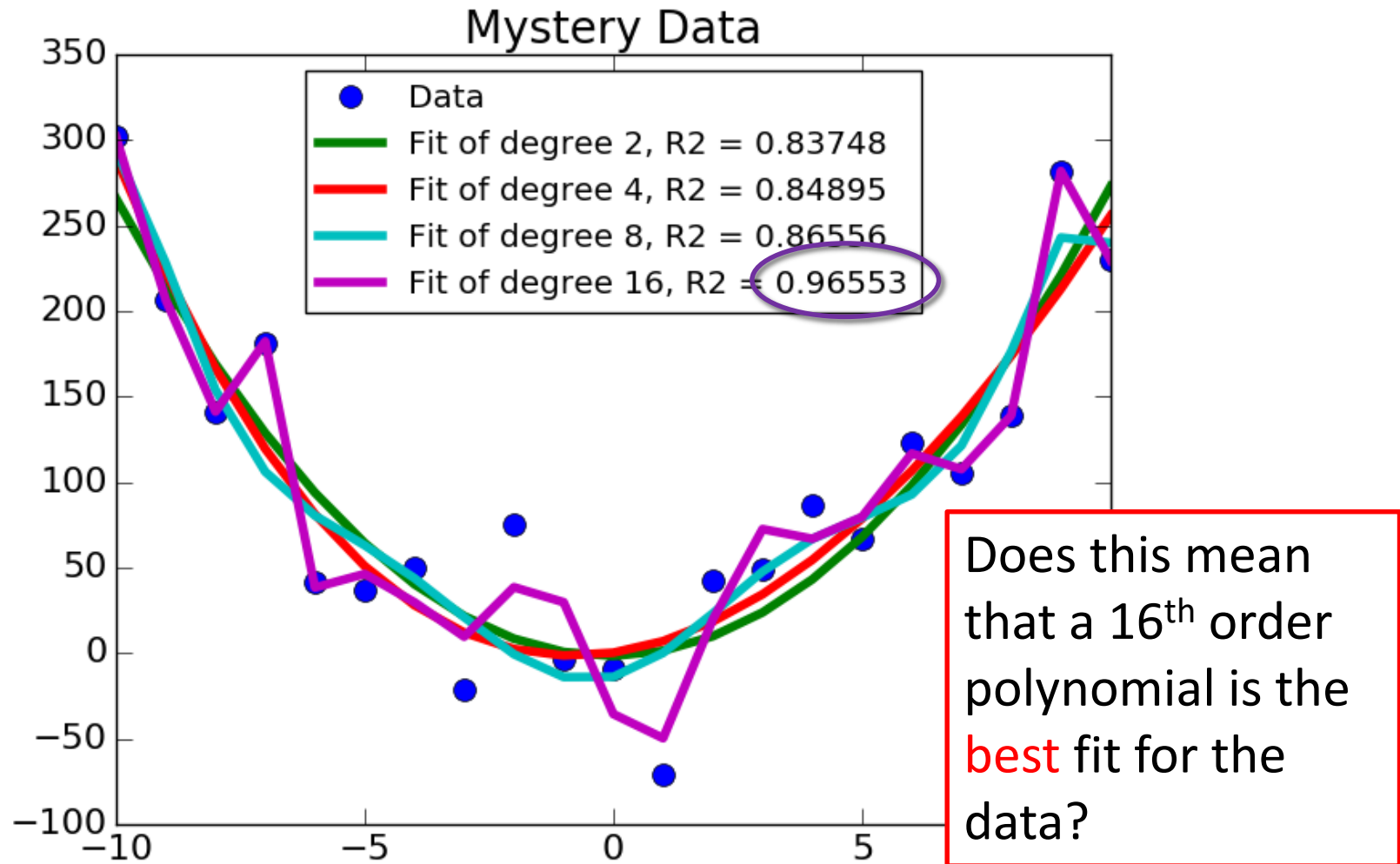
How Well Do Fits Explain Variance?



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?

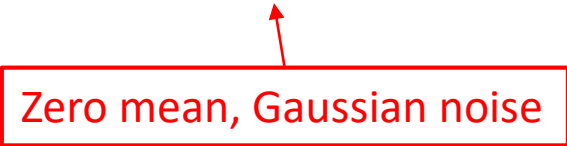


Does Tightest = Best?

- 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()  
  
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')
```



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
 - Might yield terrible predictions for unseen data

