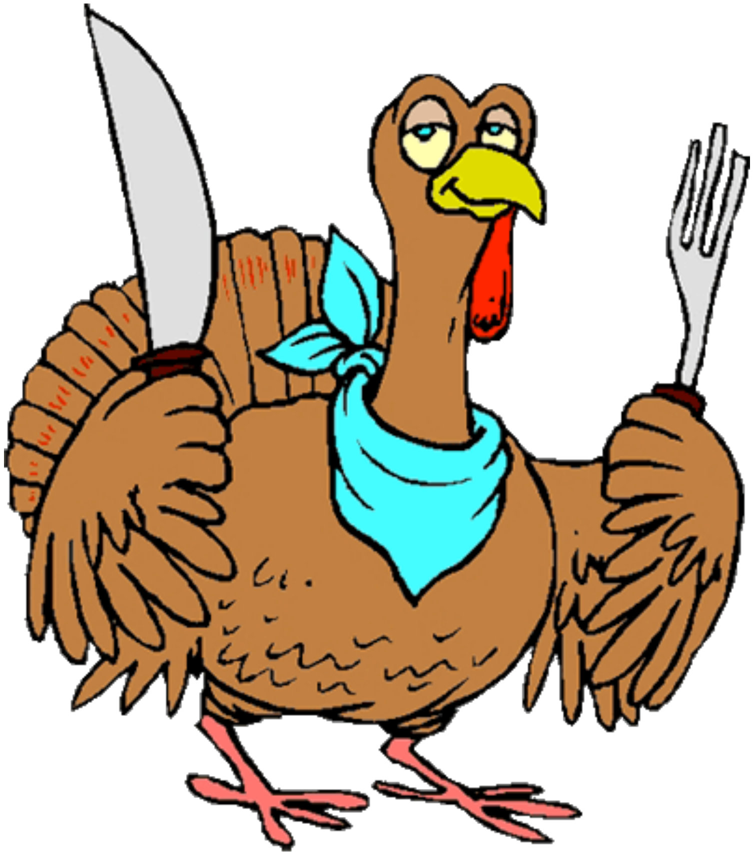
Understanding Experimental Data

Eric Grimson

MIT Department Of Electrical Engineering and Computer Science

Announcements

- Reading: Chapter 18
- No lecture on Wednesday



Statistics Meets Experimental Science

- Conduct an experiment to gather data
  - Physical (e.g., in a biology 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

Net Gain on a
missed jump shot

$$= P(\text{off reb}) \times E[\text{pts for}] - P(\text{def reb}) \times E[\text{pts against}]$$

- Consider, for example, a spring

One Kind of Spring

Another Kind of Spring

Photo by Bachsteize

This Kind of Spring

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

$k \approx 1 N / m$

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

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

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

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

$F=350N$

$mass*9.8m/s\uparrow 2=350N$

$mass=350N/9.81m/s\uparrow 2$

$F=mass*acc$

$F=mass*9.8m/s\uparrow 2$

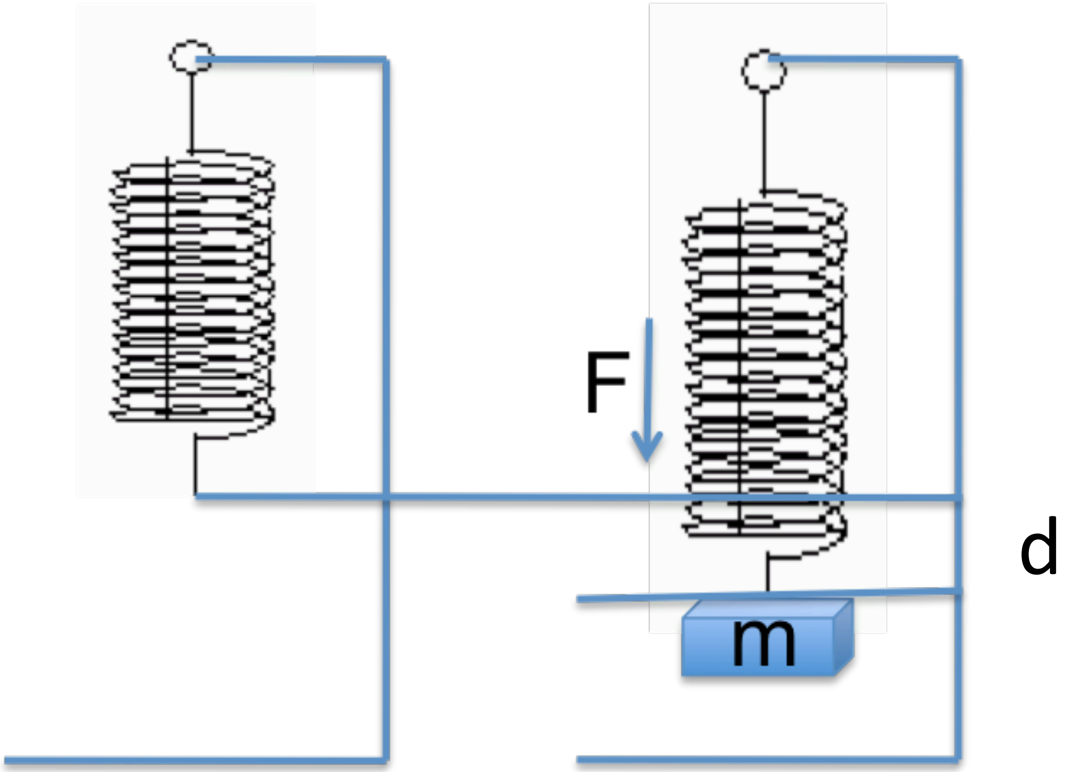
$mass=350k/9.81$

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

$mass\approx 35.68k$

Finding k

- $F = -kd$
- $k = -F/d$
- $k = 9.81 * m/d$



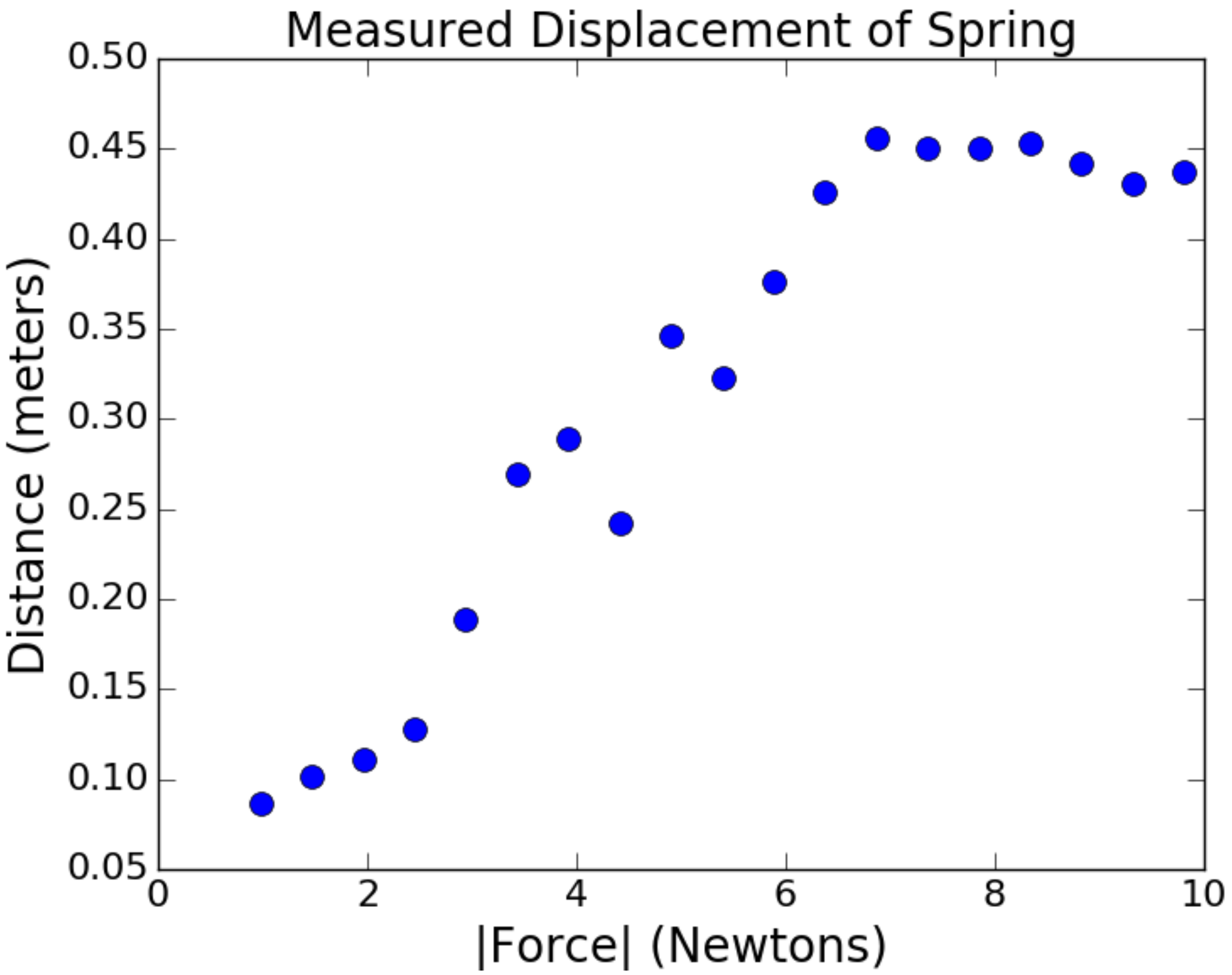
Some Data

Distance (m)	Mass (kg)
0.0865	0.1
0.1015	0.15
0.1106	0.2
0.1279	0.25
0.1892	0.3
0.2695	0.35
0.2888	0.4
0.2425	0.45
0.3465	0.5
0.3225	0.55
0.3764	0.6
0.4263	0.65
0.4562	0.7

Taking a Look at the Data

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

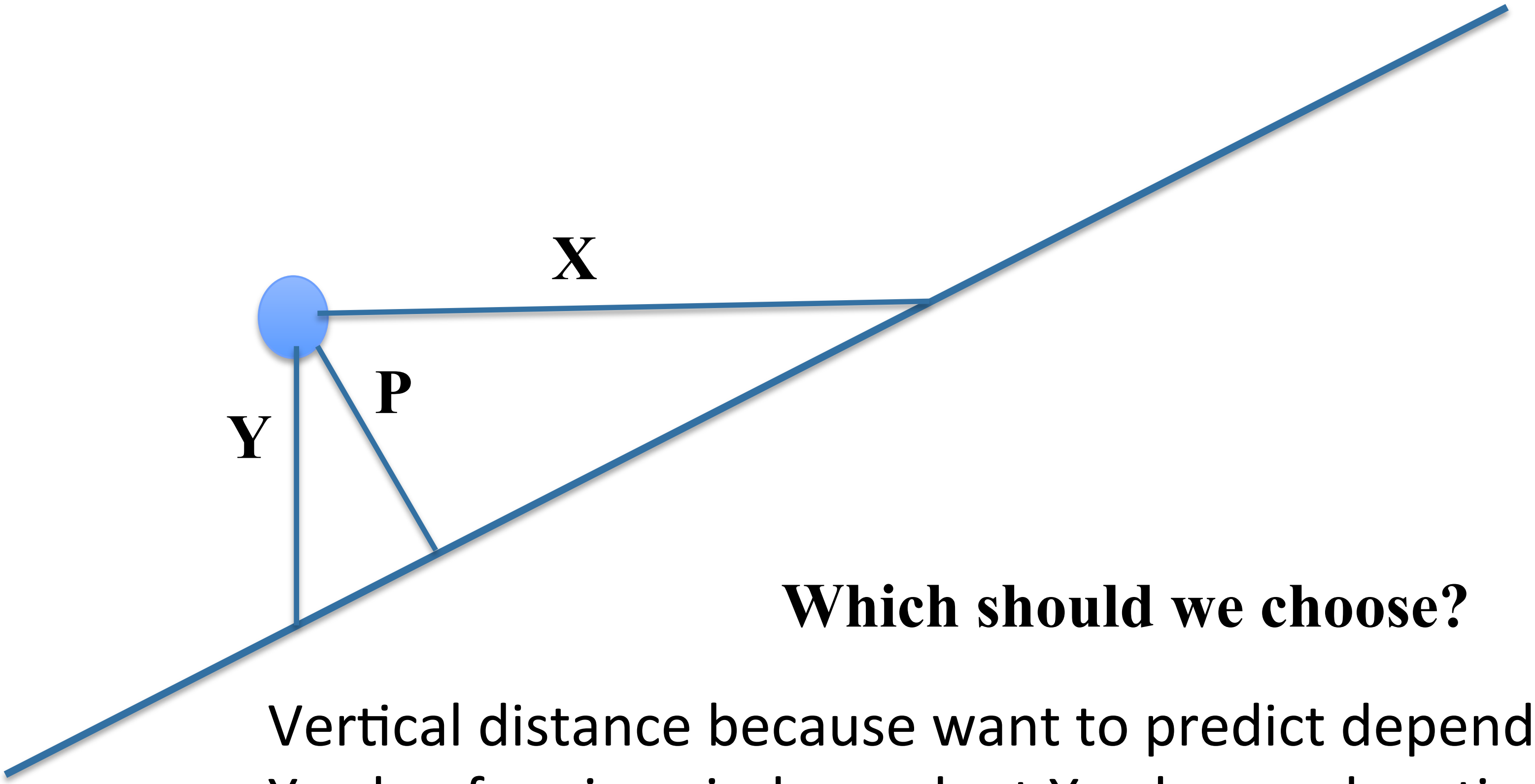
Taking a Look at the Data



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) 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 want to find the curve that minimizes it
- In this case, we want to find a line such that some function of the sum of the distances from the line to the measured points is minimized

Measuring Distance



Which should we choose?

Vertical distance because want to predict dependent Y value for 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
- Use **linear regression** to find a polynomial representation for the predicted model

Polynomials with One Variable (x)

- 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: $ax^2 + bx + c$

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 our data (we want best fitting line)

- Find values of *a* and *b* such that when we use the polynomial to compute *y* values for all of the *x* values in our experiment, the squared difference of these **predicted** values and the 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)

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 $y = ax^2 + bx + c$

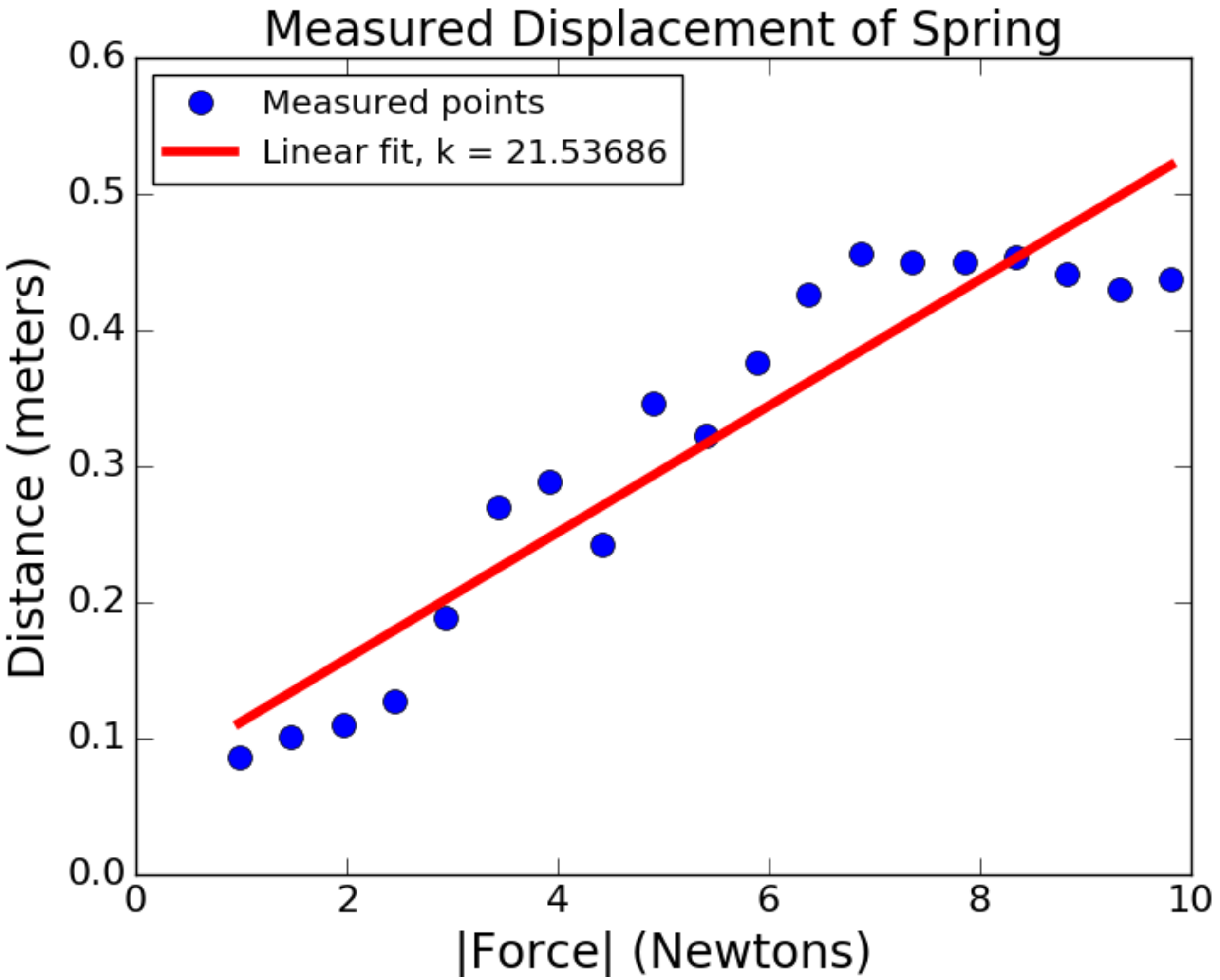
Using polyfit

```
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

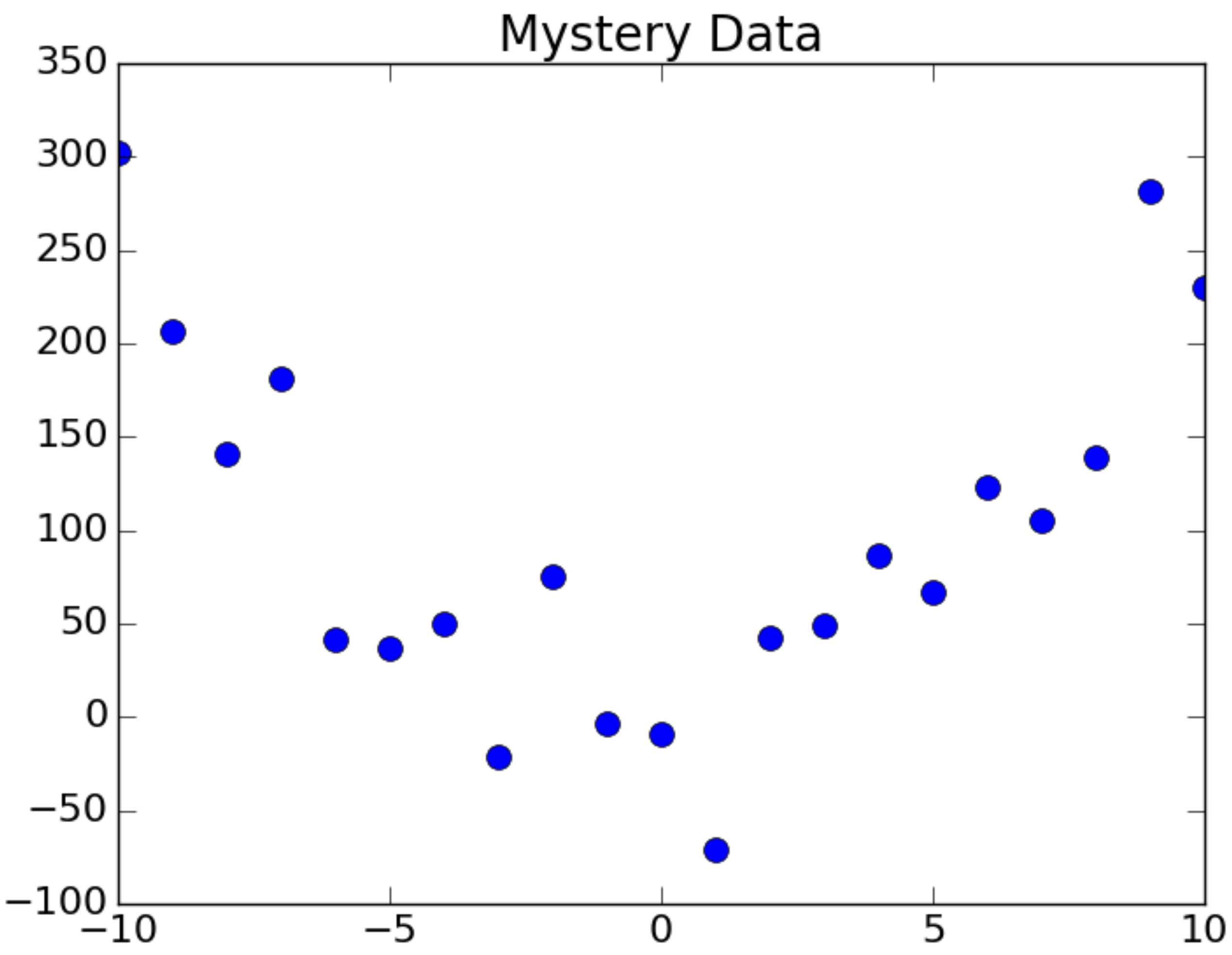
Visualizing the Fit



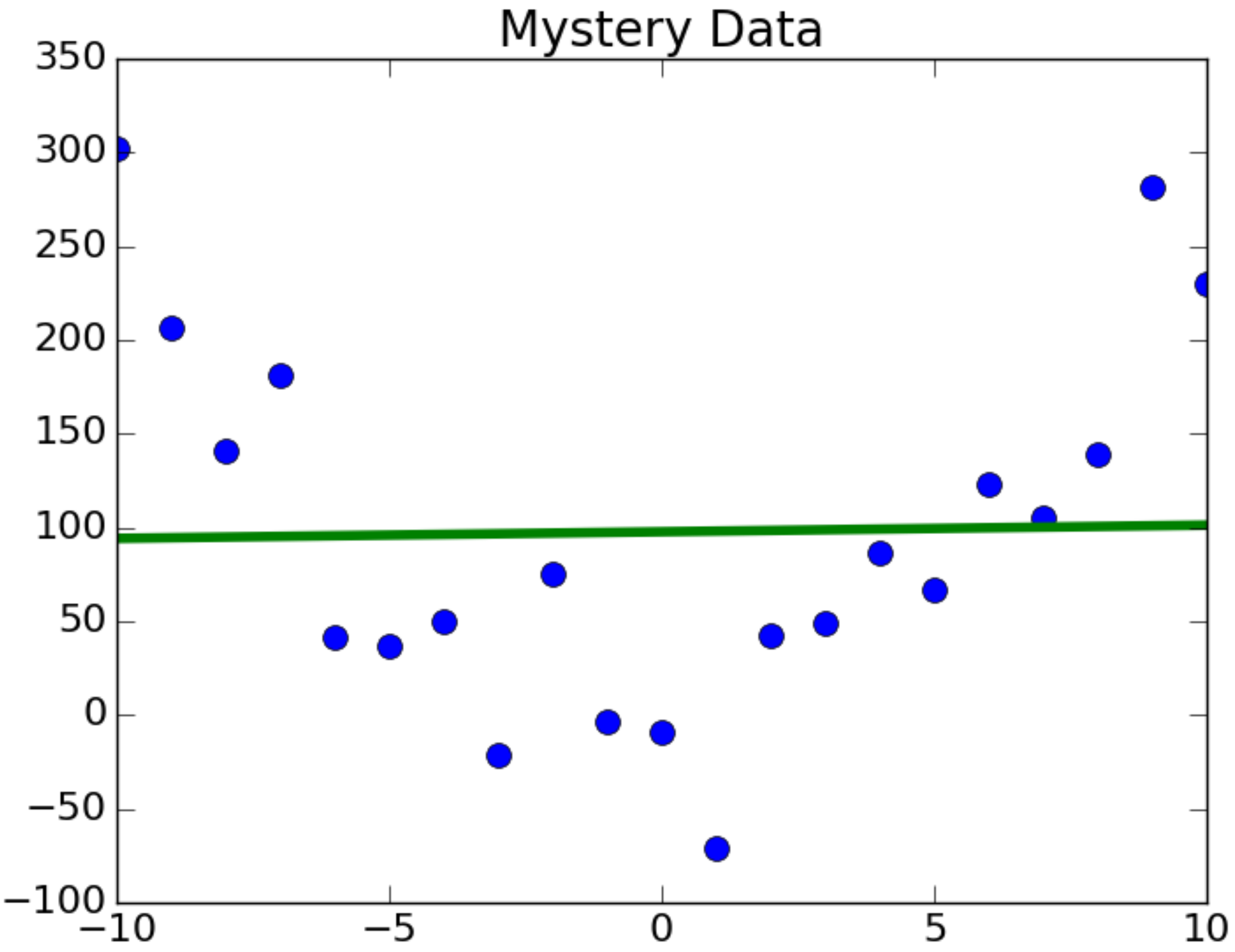
Version Using polyval

```
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')
```

Another Experiment



Fit a Line

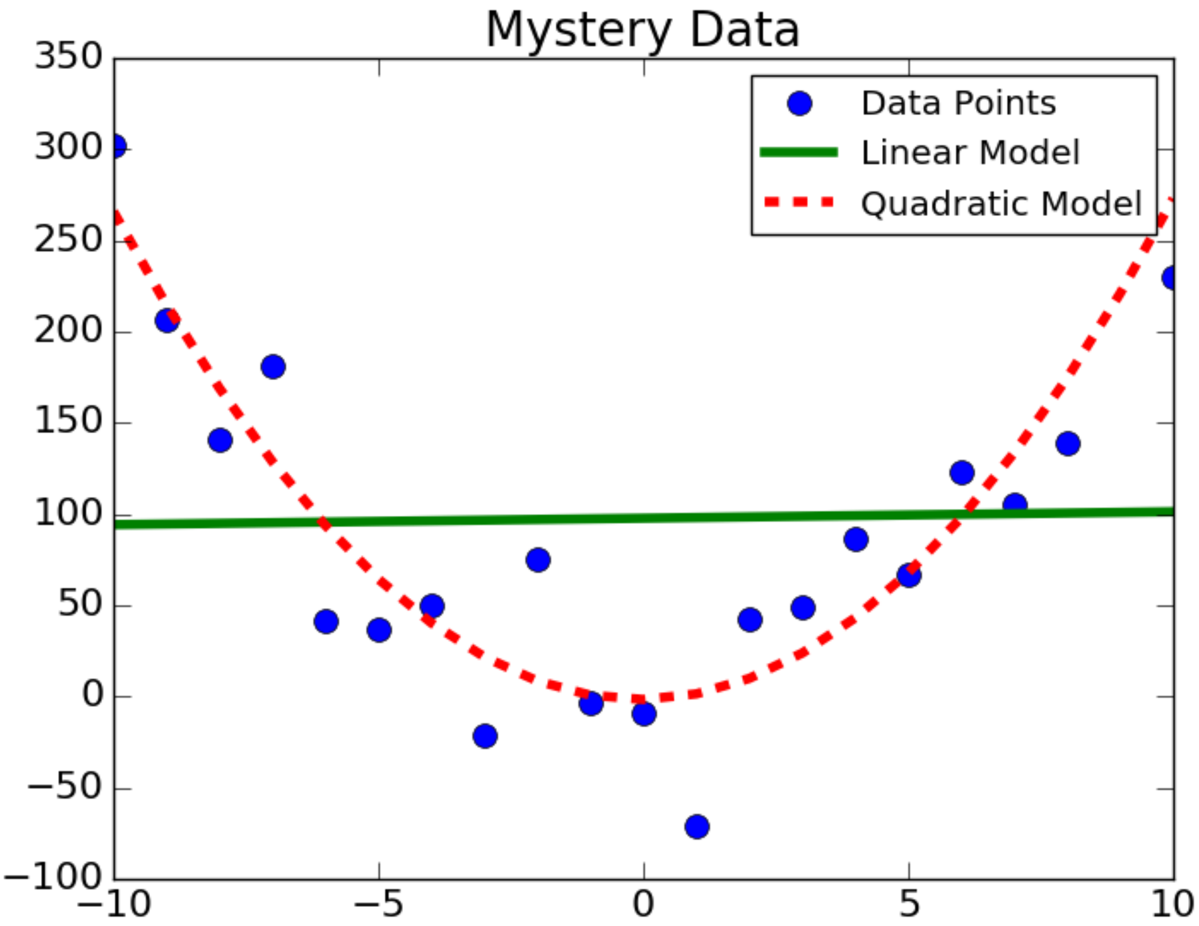


Let’s Try a Higher-degree Model

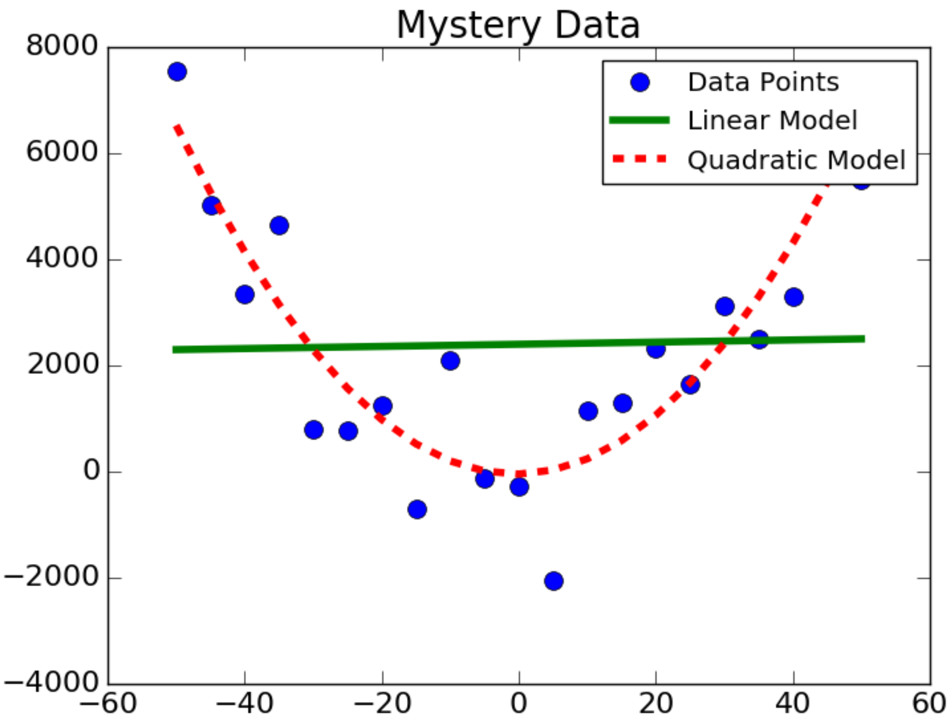
```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

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)

Quadratic Appears to be a Better Fit



How Good Are These Fits?



- Relative to each other
- 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 = 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
```

In an Absolute Sense

- Mean square error useful for comparing two different models for the same data
- Useful for getting a sense of absolute goodness of fit?
  - Is 1524 good?
- Hard to know, since there is no upper bound and not scale independent
- 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

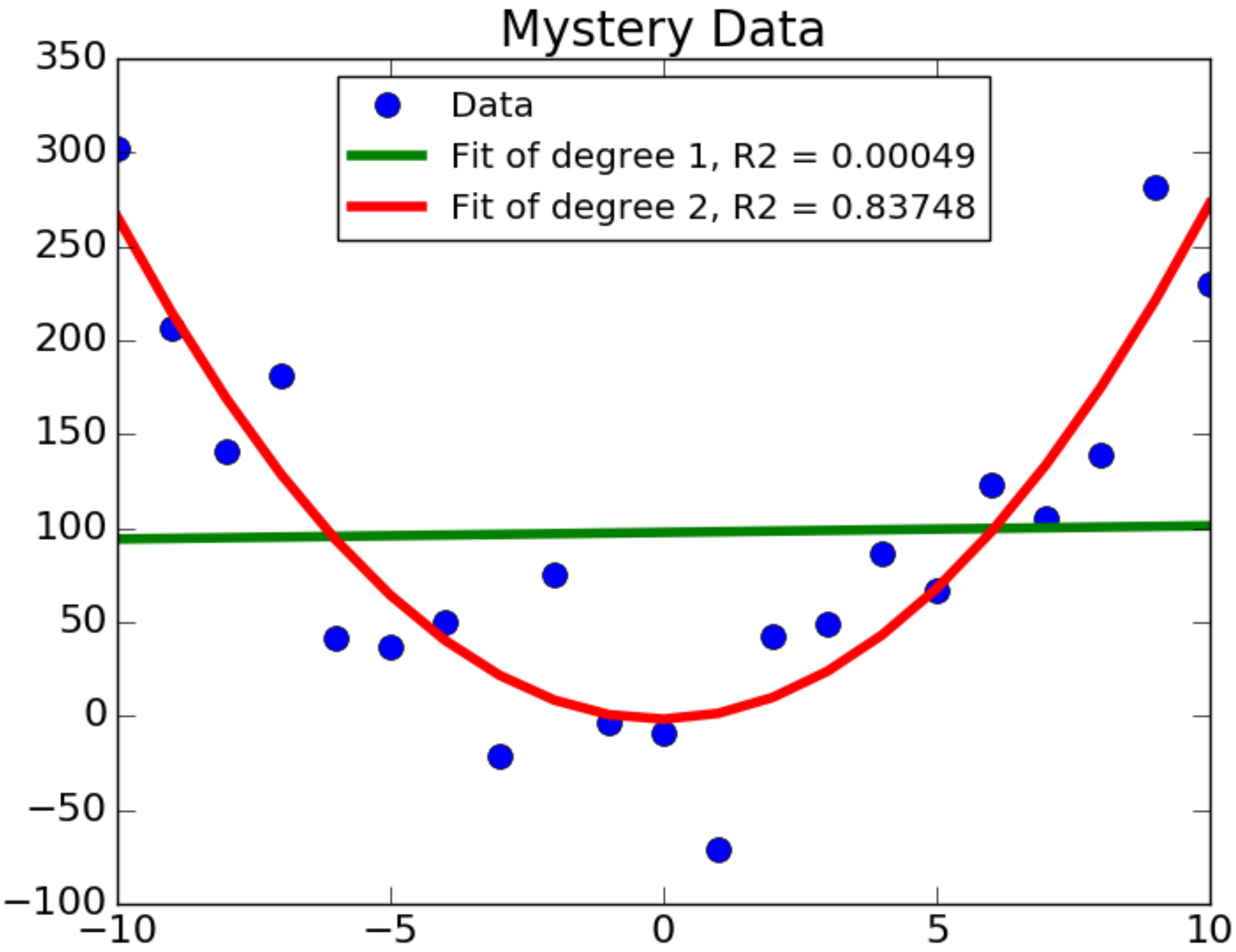
- By comparing the estimation errors (the numerator) with the variability of the original values (the denominator), R^2 is intended to capture the proportion of variability in a data set that is accounted for by the statistical model provided by the fit
- Always between 0 and 1 when fit generated by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



Can We Get a Tighter Fit?