Lesson 10

Understanding Experimental Data

This Kind of Spring

$k \approx 1 N / m$

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

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

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?

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

$F = 350N$

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

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

$F = mass * acc$

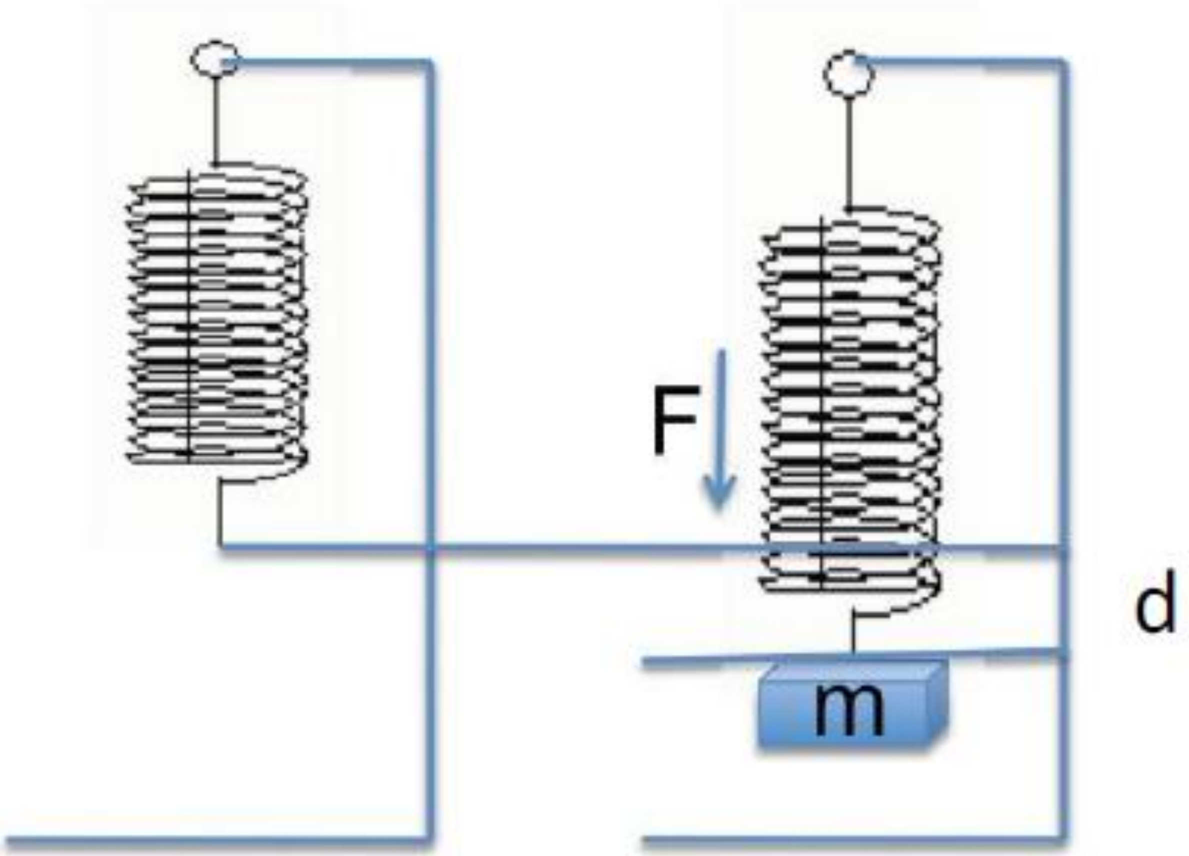
$F = mass * 9.8m/s\uparrow 2$

$mass = 350k/9.81$ ← 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



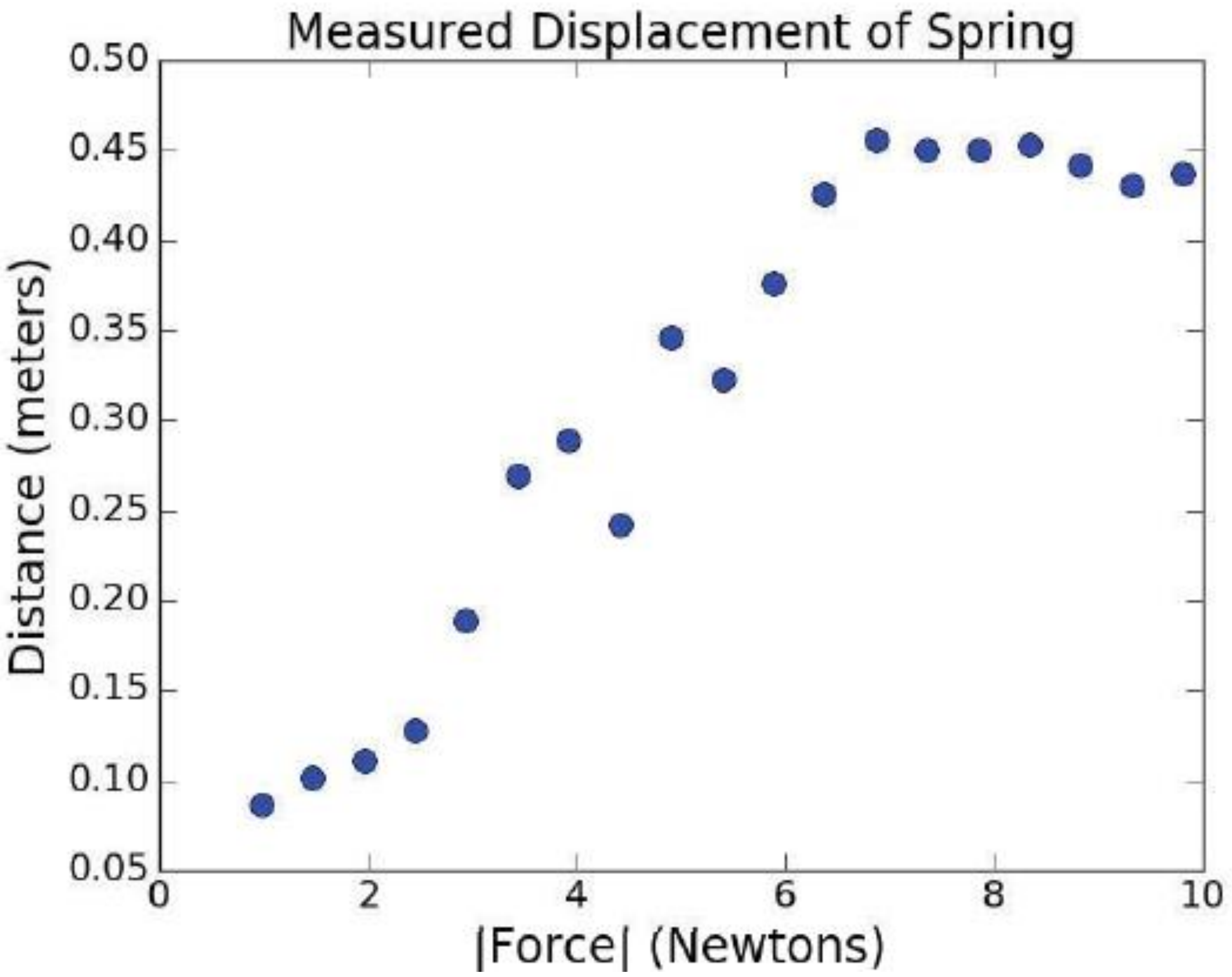
Some Data

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()
```

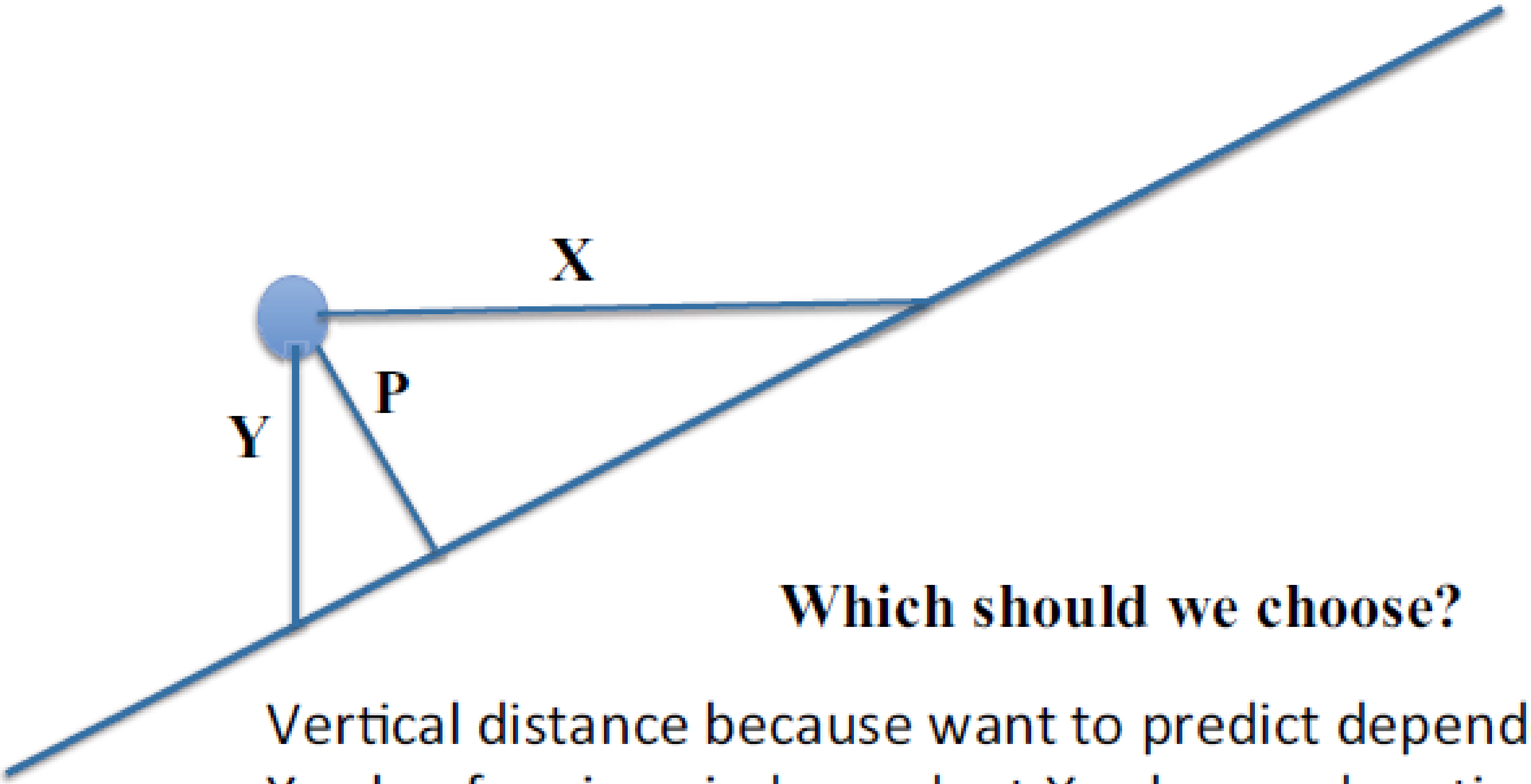
Taking a Look at the Data



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

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 $y = ax^2 + bx + c$

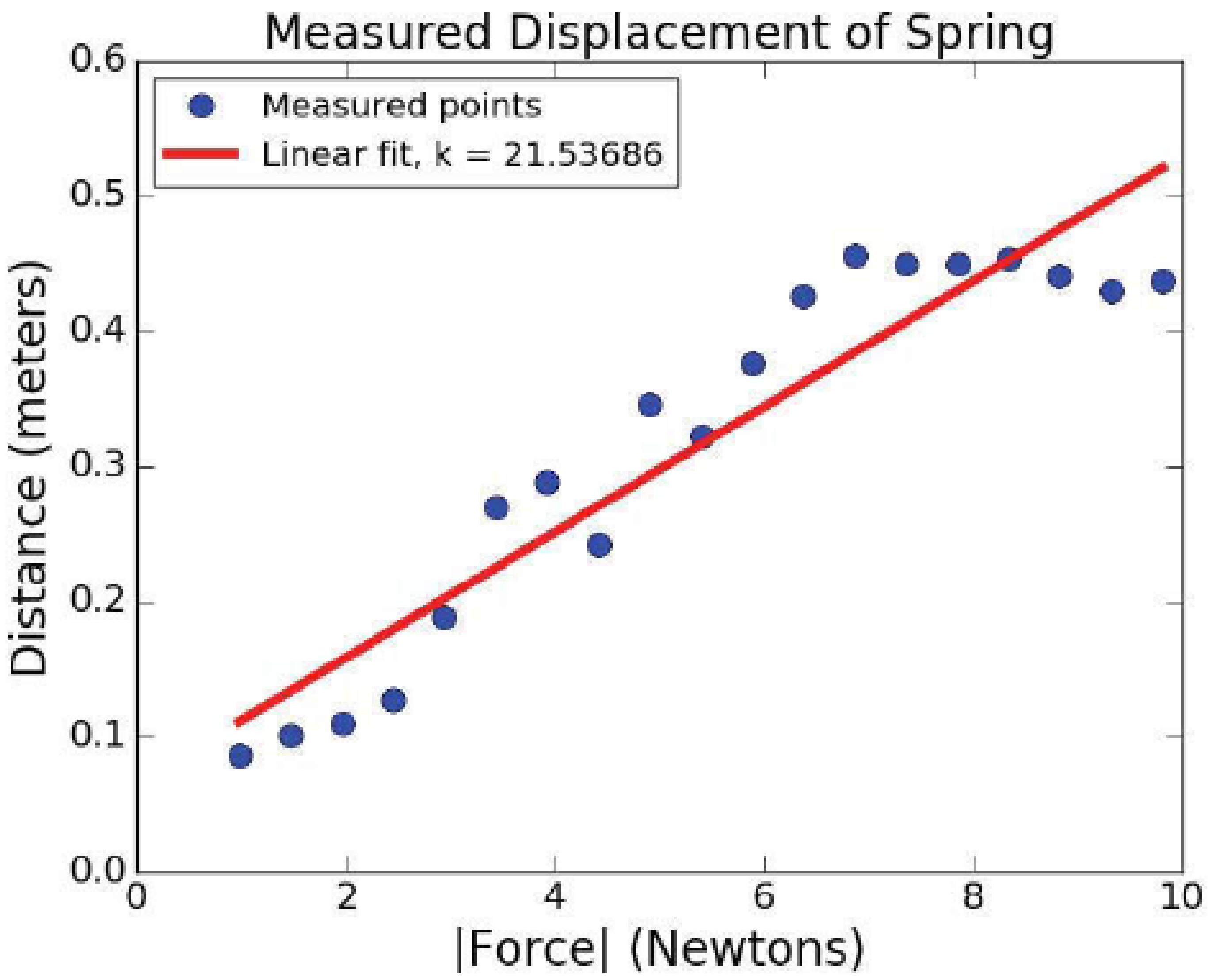
Using polyfit

```
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')
```

How to know whether it is a good fit? How to compare?

r square.

To do.

- data file: springData.txt (can download in IVY)
- Understanding Experimental Data Notebook: https://colab.research.google.com/drive/1q0VzsAUHqadpHbJOGOLkxr3hEm_EWl6y?usp=sharing (modify the code and define r_square)
- Assignment: Anscombe’s Quarter (anscombes.csv)
- Assignment 10

- Additional Resources:
  - fitting a line: https://www.youtube.com/watch?v=PaFPbb66DxQ
  - R_square: https://www.youtube.com/watch?v=2AQKmw14mHM

Additional resource to attempt to understand Gradient Descent.

- Implementation of OLS on p5.js: https://www.youtube.com/watch?v=_cXuvTQl090
- Gradient Descent explained: https://www.youtube.com/watch?v=sDv4f4s2SB8&t=253s
- Essence of Calculus by 3blue1brown: https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr
- Mathematics of Gradient Descent:
  - https://www.youtube.com/watch?v=jc2IthslyzM&list=PLRqwX-V7Uu6bCN8LKrcMa6zF4FPtXyXYj&index=8