Understanding Experimental Data

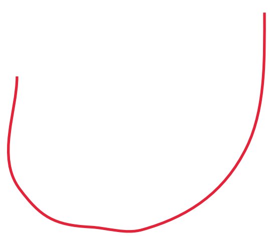
Solving for Least Squares

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

- Use linear regression to find a polynomial

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

$$\sum_{i=0}^{len(observed)-1} (observed[i] - predicted[i])^2$$ ← loss func, minimize it

base on its differentiality

- We will use a degree-one polynomial, $y = ax$+b, as model of our data (we want a 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 values and the corresponding observed values is minimized
- A linear regression problem
- Many algorithms for doing this, including one similar to Newton's method (shown in 6.00.1x)

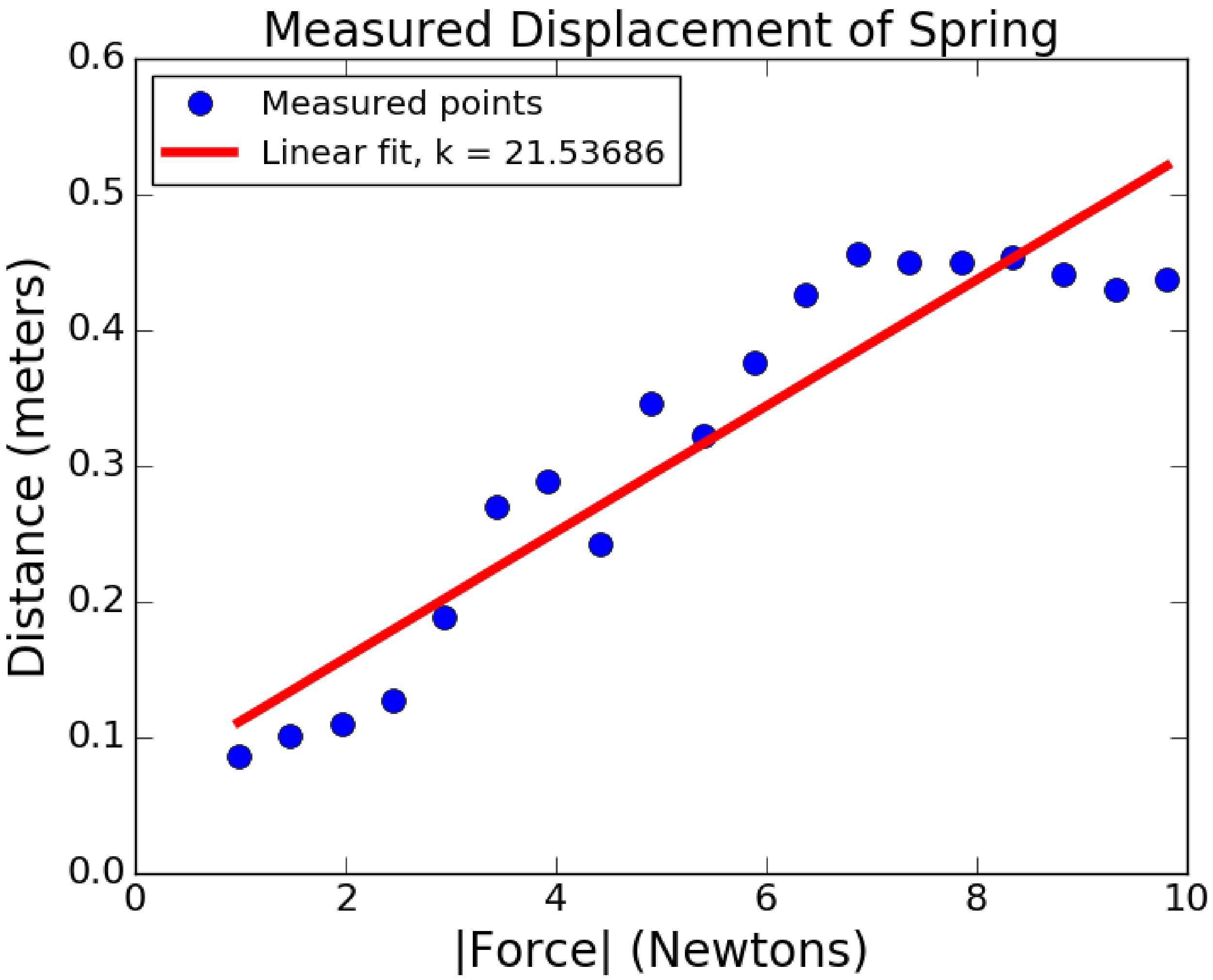
polyFit

- `pylab.polyfit(observedX, obervedY, n)`
- Finds coefficients of a polynomial of degree n, that provides a best least squares fit for the observed data

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')
```

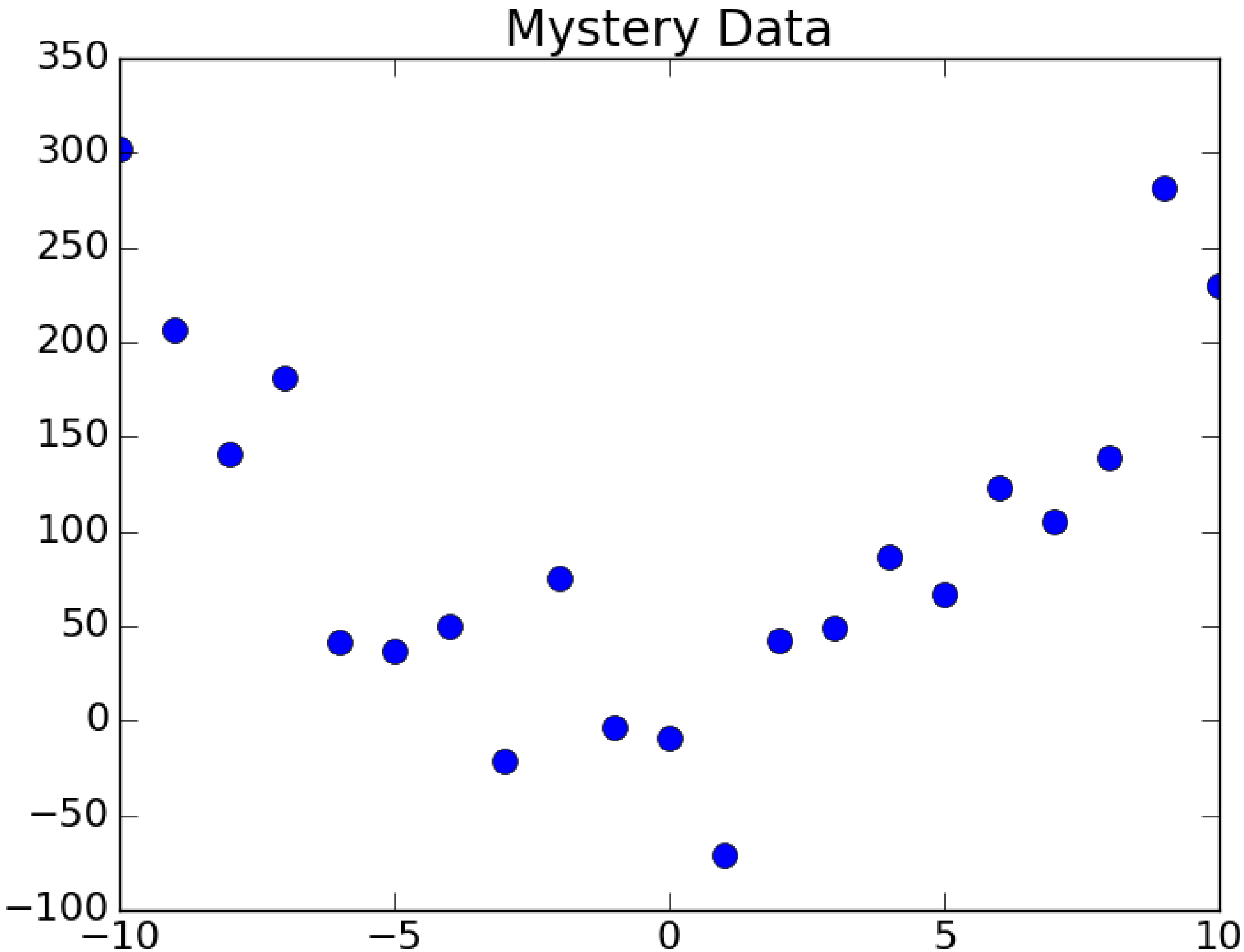
Visualizing the Fit



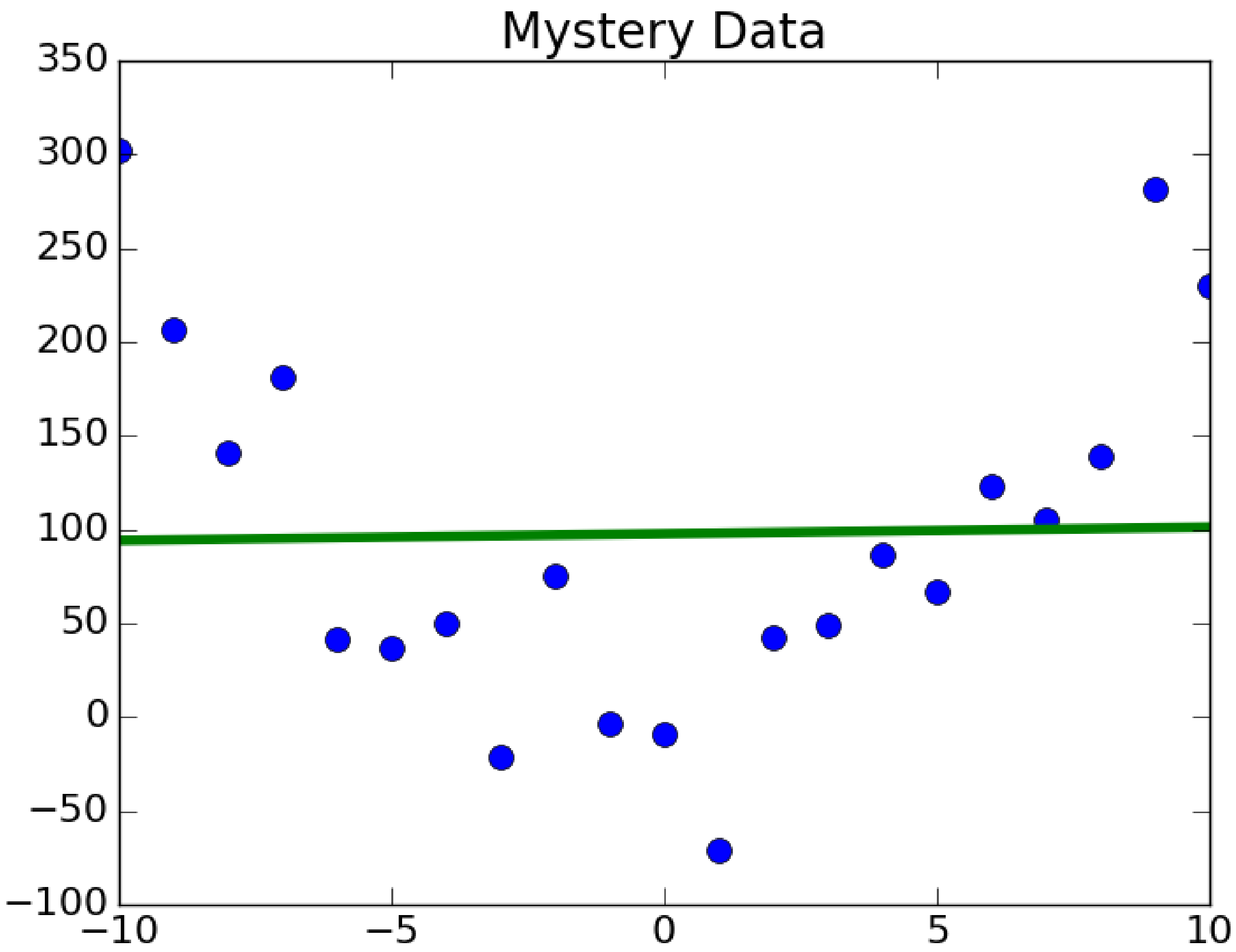
Version Using polyval

```
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')
```

Quadratic Appears to be a Better Fit