# Lecture 3. Introduction to Linear Regression

### CS 109A/AC 209A/STAT 121A Data Science:

### Harvard University

Fall 2016
Instructors: P. Protopapas, K. Rader, W. Pan

## Announcements

- HW due dates changed. **All** homework due times shifted to Wednesday 11:59pm
- Only one submit allowed
- We are not approving anymore enrollments
- Projects: AC209 students can only work with each other and we'll create group for them on Canvas
- Projects Milestone 1 submit separately via google form (due 9/21)
- We will release project topics soon
- HW grading standards: correctness, interpretation, presentation

# Quiz Time

# Outline

- Why linear regression, history, motivation, definitions
- Choices of lost functions
- Estimate coefficients and how well are estimated
- Goodness of fit, RSE, p-value, R^2
- Residuals

## History

- Tobias Mayer (17 February 1723 – 20 February 1762) was a German astronomer famous for his studies of the Moon.

- Euler memoir of the problem of the inequalities in the movement of Saturn and of Jupiter revealed the story of two different approaches.

In calling attention to Euler's statistical failure, I mean to imply no criticism of Euler as a mathematical scientist. Rather by contrasting his work with Mayer's I want to highlight the extremely subtle conceptual advance that was evident in Mayer's work. Euler's memoir made a significant contribution to the mathematical theory of attraction. Even his crude empirical solution, setting most of the correction terms equal to zero, provided him with an improvement over existing tables of Saturn's motion. His inability to resolve the major inequalities in the planets' motions was in the end due to the inadequacy of his theory as well as to his lack of statistical technique. Euler himself was satisfied, correctly, that no values of his correction factors could adequately account for the planet's motions, but he suggested as a possible cause for this the failure of Newton's inverse square law of attraction to hold exactly over large distances! The problem in fact withstood successive assaults by Lagrange, Lambert, and Laplace before finally yielding to Laplace in 1787. The lesson Euler's work has for the history of statistics is that even though before 1750 mathematical astronomers were willing to average simple measurements (combining observational evidence from several days or observers into a single number), it was only after 1750 that the conceptual advance of combining observational equations (with varying coefficients for several unknowns testifying to the differing circumstances under which the observations had been made) began to appear.

## Why Linear regression

- Simple and sligtly boring method
- Has been around for long time as we saw above
- Supervised learning
- Still used in many areas and it is the foundation of many other more sophisticated models

- Helps us understand
  - How to justify the relation between the dependend and independend variables?
  - How strong that relation is?
  - Which of the independend variables (X) are significant?
  - How we understand the quality of our prediction?
  - What is the relationship between X and Y. Is it linear or quadratic etc?
  - If there is interaction amongst the independend variables ($x_1$, $x_2$, ...)

# Simple Linear Regression

- Straightforward simple linear approach for predicting a quantitative response Y on the basis of a single predictor variable X.
- It assumes that there is approximately a linear relationship between X and Y

$$Y \approx \beta_0 + \beta_1 X$$

Why $\approx$?

Remember:

$$Y = f(X) + \epsilon$$

So $\approx$ means $f(X)$ is approximated to be this simple linear relationship

$$f(X) \approx \beta_0 + \beta_1 X$$

and $\approx$ takes into account $\epsilon$

Note: Regressing Y on X (or Y onto X).

$\beta_0$ and $\beta_1$ are:

- two unknown constants (intercept and slope) or the model coefficients
- something we need to estimate
- how can we estimate them?

We can use Mayer's approach: Use all the data we have to do so

Say we have X=[1,2] and Y=[0.6, 0.9]

Then we can two equations and find out $\beta_0$ and $\beta_1$ as

$$0.6 = \beta_0 + \beta_1$$
$$0.9 = \beta_0 + 2\beta_1$$

$\beta_1 = 0.3$ and $\beta_0 = 0.3$

But what if we have more data? Say X= [1,2,3] and Y = [0.6, 0.9, 1.6] Then

$$0.6 = \beta_0 + \beta_1$$
$$0.9 = \beta_0 + 2\beta_1$$
$$1.6 = \beta_0 + 3\beta_1$$

Too many equations and not consistent ($\beta_1 = 0.3$ and $\beta_0 = 0.3$ does not work for the last equation)

What to do?

If the universe was perfect (no noise) and our measurements were perfect then the three equations should be consistent and we would only need two measurements. X= [1,2,3] and Y = [0.6, 0.9, **1.2** ] or X= [1,2,3] and Y=[0.5, 1.0, 1.5]

Two things are respnsible for this inconsistency

1. Noise in the data or stochastic behavior in the response function
2. The true model is not exactly linear

So lets use all the data to best estimate the coefficients hoping that these errors will cancel out !!

Think like a statistician not as a mathematician. Can beat errors by adding more measurements (we will see more below)

We use our data aka training data to produce estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$

and predict values of $y$ for any $x$

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Note: We use ˆ , to denote the estimated value for an unknown parameter, and the predicted value of the response

## Estimating the Coefficients

We use data to estimate the coefficients. For $n$ observations,

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

Goal is to obtain coefficient estimates $\beta_0$ and $\beta_1$ such that the linear model fits the available data **well**

Now we need to address what **well** means

For sure $e_i = y_i - \hat{y}_i$, the residuals, represents the difference between the measurement and the estimate. LETS USE THIS

- How?
- And how we combine all residuals (e) for all measurements?

**Max absolute deviation** Count only the biggest "error"

$$L(\beta_0, \beta_1) = \max_i (y_i - \hat{y}_i)$$

**Sum of absolute deviations**: Add up (or take the average) all the absolute values of the "errors"

$$L(\beta_0, \beta_1) = \sum_i |y_i - \hat{y}_i|$$

**Sum of squared errors**: Add up (or take the average) of the squares of the "errors"

$$L(\beta_0, \beta_1) = \sum_i (y_i - \hat{y}_i)^2$$

$L(\beta_0, \beta_1)$ is called the **lost** (or cost) function

Our goal is to find $(\beta_0, \beta_1)$ such that the lost, $L(\beta_0, \beta_1)$, is minimal

$$(\hat{\beta}_0, \hat{\beta}_1) = arg\min_{\beta_0, \beta_1} \; L(\beta_0, \beta_1)$$

Finding the optimal values $\hat{\beta}_0, \hat{\beta}_1$ is called fitting the linear model.

However, by far the most common approach involves minimizing least squares (sum of the square errors)

$$RSS = \sum_i e_i^2 = \sum_i (y_i - \hat{y}_i)^2$$

and it is easy to prove (my grandma can do this)

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{y} = \frac{1}{n} \sum_i y_i$ and $\bar{x} = \frac{1}{n} \sum_i x_i$ are the sample means.

Alternatively, in case that L can not be differentiated, as in the case of Sum of absolute deviations, we find $\hat{\beta}_0, \hat{\beta}_1$ with numerical methods (more on this later).

## Why sum of the squared errors?

Lets revisit the error term $\epsilon$. So far we stated that we assume the model is linear but we never discussed the nature of $\epsilon$.

Suppose that $\epsilon \sim N(0, \sigma_e)$ then what would be the probability distribution for Y?

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Since $\beta_0, \beta_1$ and X have no stochasticity then

$$Y \sim N(\beta_0 + \beta_1 X, \sigma_e)$$

And the likelihood (probability that the data came from this model) of measuring $y_i$ is given by:

$$L(y_i) = \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\left\{ -\frac{(y_i - (\beta_0 + \beta_1 X))^2}{2} \right\}$$

And for all measurements?

$$L(Y) = \prod_i L(y_i)$$

One can take the log-likelihood of this and can show that

$$\mathcal{L} = -\frac{1}{2}\sum_i (y_i - (\beta_0 + \beta_1 X))^2$$

Therefore finding the minimum of sum of squared errors is the same (for normally uncorrelated measurements) to the Maximum Likelihood Estimation ( **MLE**)

## How well did we do finding the coefficients?

Remember there are two reasons for not be "exact" or perfect estimate

1. Noise in the data
2. Actual model is not linear

For now lets assume that the underline model is actual linear. This is called the population regression line.

How well can we estimate the coefficients?

We wonder how close $\hat{\beta}_0$ and $\hat{\beta}_1$ are to the $\beta_0$ and $\beta_1$

Simple example to demonstrate this. Generate 100 points from equation

$$Y = 2.2 + 3.0\,X + \epsilon$$

Estimate the coefficients and compare to 2.2 and 3.0. Do you think they will be the same? If we generate a new 100 points do you expect $\hat{\beta}_0$ and $\hat{\beta}_1$ to be the same?

In [163]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression as Lin_Reg
```

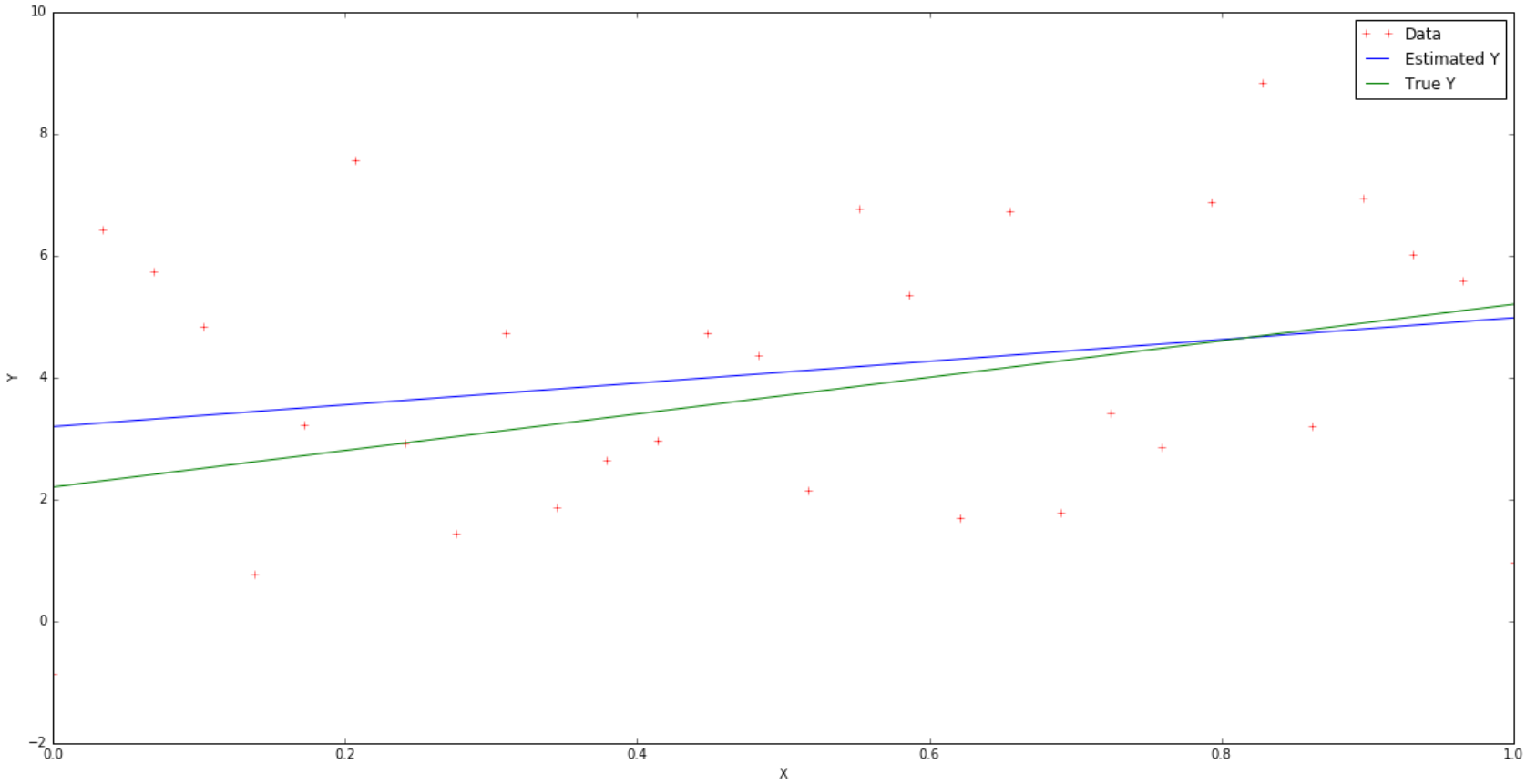
In [493]:
```
## GENERATE ANOTHER DATASET AND EXAMINE RESIDUALS
def GenerateDataLinearFun(N=1000, beta0=2.2, beta1=3.0, sigma=10.0, Xmax=1.0):
    epsilon=np.random.normal(0,sigma,N)  # Random normally distributed points
    X = np.linspace(0,Xmax, N)
    Y = beta0 + beta1 * X + epsilon
    return X, Y


def GenerateDataNotLinearFun(N=1000, beta0=2.2, beta1=3.0, sigma=1.0, Xmax=1.0, alpha=1):
    epsilon=np.random.normal(0,sigma,N)  # Random normally distributed points
    X = np.linspace(0,Xmax, N)
    Y = beta0 + beta1 * X + alpha* np.sin(6*X)+ epsilon
    #Y =  beta0 + beta1 * X +4*beta1*(X>0.5) + epsilon
    #Y =  beta0 + beta1 * X + alpha * X*X*X + epsilon
    return X, Y

def FitLinearModel(X,Y):
    # Estimate the coefficients
    beta1 =  np.sum((X-np.mean(X)) * (Y-np.mean(Y)))/np.sum( (X-np.mean(X))**2)
    beta0 = np.mean(Y) - beta1*np.mean(X)
    return beta0, beta1

def PredictLinearModel(X, beta0, beta1):
    Y = beta0 + beta1 * X
    return Y
```

In [466]:
```
# Generate data

X,Y = GenerateDataLinearFun(N=30, sigma=2.1)


# Estimate the coefficients
beta0_hat, beta1_hat =  FitLinearModel(X,Y)

print("estimated coefficients", beta0_hat, beta1_hat)

# estimate y
Y_hat  = PredictLinearModel(X,beta0_hat, beta1_hat)
```

```
estimated coefficients 3.19215009967 1.78514057619
```

In [467]:
```
plt.figure(figsize=(20,10))
plt.plot(X,Y, 'r+', label='Data')
plt.plot(X, Y_hat, label='Estimated Y')
plt.plot(X, beta0 + beta1 * X, label='True Y' )

plt.xlabel('X')
plt.ylabel('Y')
plt.legend(loc='best')
```

Out[467]: <matplotlib.legend.Legend at 0x12604dbe0>



How does this discrepancy depend on

- Number of points?
- Error in the data?
- Other factors?

Lets investigate. For now since we know the true $\beta_0, \beta_1$ we can just take the error on the coefficient and see how it changes as a function of this parametes.

First lets do the $\beta_1$

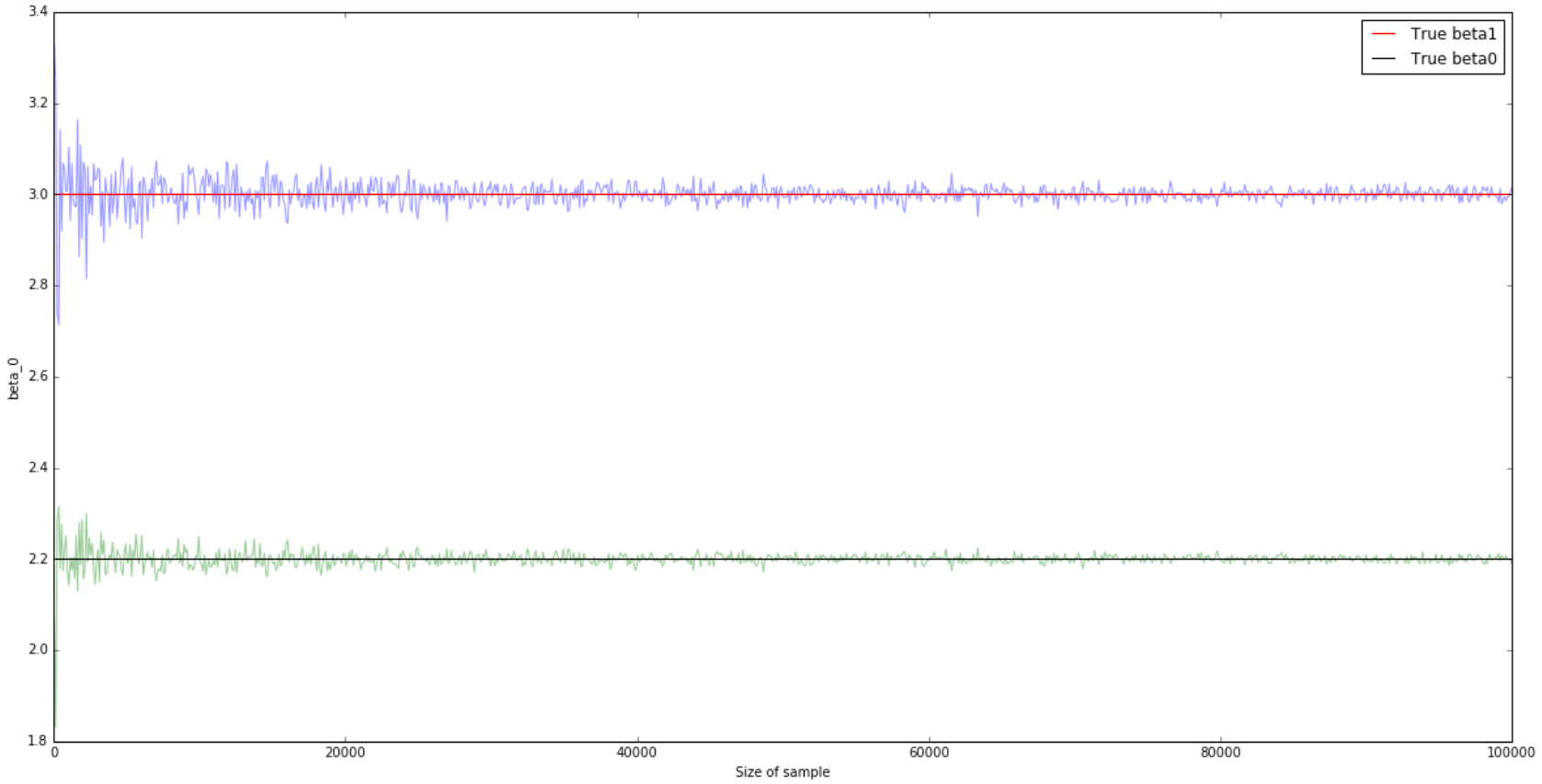
I will generate different samples sizes and check its value

In [477]:
```
Na = np.linspace(50, 100000, num=1000)
beta0_hat_a =np.zeros(np.size(Na))
beta1_hat_a =np.zeros(np.size(Na))

# Evaluate coefficients for different N

for i, N in enumerate(Na):
    X, Y = GenerateDataLinearFun(N, sigma=1)

    # Estimate the coefficients
    beta0_hat_a[i], beta1_hat_a[i] = FitLinearModel(X,Y)
```

```
/Users/pavlos/anaconda/envs/py35/lib/python3.5/site-packages/ipykernel/__main__.py:3: DeprecationWar
ning: using a non-integer number instead of an integer will result in an error in the future
  app.launch_new_instance()
```

In [478]:
```
plt.figure(figsize=(20,10))
plt.plot(Na,beta1_hat_a, alpha=0.4)
plt.axhline(y=beta1, color='r', label='True beta1')
plt.xlabel('Size of sample')
plt.ylabel('beta_1')
plt.legend()

plt.plot(Na,beta0_hat_a, alpha=0.4)
plt.axhline(y=beta0, color='k', label='True beta0')
plt.xlabel('Size of sample')
plt.ylabel('beta_0')
plt.legend(loc='best')
```

```
Out[478]: <matplotlib.legend.Legend at 0x126cf5a20>
```



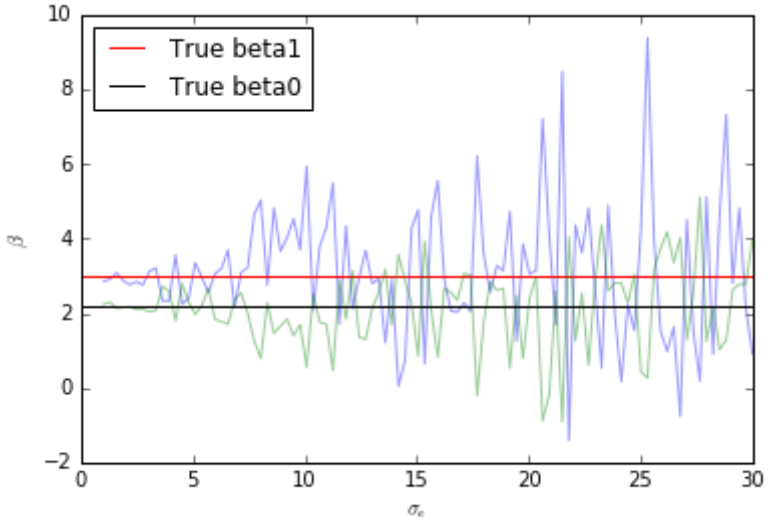

Obviously there is a $1/n$ relationship

Now let's look at the relationship wrt to size of the error ($\epsilon$)

In [429]:
```
# UGLY CODE BUT CLEANER IN PRESENTATION
# Investigate the relationship wrt to sigma_e

sigma_e_a = np.linspace(1.0, 30, num=100)
beta0_hat_a =np.zeros(np.size(sigma_e_a))
beta1_hat_a =np.zeros(np.size(sigma_e_a))



N = 1000

for i, sigma_e in enumerate(sigma_e_a):
    X, Y = GenerateDataLinearFun(N, sigma=sigma_e)

     # Estimate the coefficients
    beta0_hat_a[i], beta1_hat_a[i] = FitLinearModel(X,Y)
```

In [430]:
```
plt.plot(sigma_e_a,beta1_hat_a, alpha=0.4)
plt.axhline(y=beta1, color='r', label='True beta1')
plt.xlabel('Size of sample')
plt.ylabel('beta_1')
plt.legend()

plt.plot(sigma_e_a,beta0_hat_a, alpha=0.4)
plt.axhline(y=beta0, color='k', label='True beta0')
plt.xlabel(r'$\sigma_e$')
plt.ylabel(r'$\beta$')
plt.legend(loc='best')
```

Out[430]: <matplotlib.legend.Legend at 0x1200b85c0>




[ADD XRNAGE]

One can think of this the same way as the estimate of the mean of a variable.

[add mean stuff]

The standard errors associated with $\hat{\beta}_0$ and $\hat{\beta}_1$, we use the following formulas:

$$SE(\hat{\beta}_0)^2 = \sigma^2 \left[ \frac{1}{n} + \frac{\bar{x}^2}{\sum_i (x_i - \bar{x})^2} \right]$$

$$SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_i (x_i - \bar{x})^2}$$

Observe:

- What happens if $n \to \infty$
- What happens if the scatter in x is larger?

Wait !!!

What about $\sigma^2$?

1. $\sigma^2$ is the variance (or error) in our data $\mathrm{Var}(\epsilon)$, which **we do not know**
2. For this to hold these errors must be uncorrelated (we revisit this later)
3. For this to hold we assume a normal distribution

We can assume 2 and 3 to be correct and we can estimate $\sigma^2$ from the data. The estimate also known as **residual standard error**, RSE is:

$$\sigma \approx RSE = \sqrt{\frac{RSS}{n-2}}$$

$$RSS = \sum_i (y_i - \hat{y}_i)^2 = \sum_i e_i^2$$

And when we use RSS instead of $\sigma$ we

$$SE(\hat{\beta}_0) \to \hat{SE}(\hat{\beta}_0)$$

but we usually ignore the hat

So far:

- We know how to estimate the coefficients
- We know how to estimate the error on the estimate

Next:

- We should be able to say how confident are we that the estimates of the coefficients are close to their true values
- Say something about the strength of the relationship between X and Y. If Y is independent of X then $\beta_1$ is zero. Can we rule out that? Can we give a statistical significance to this statement? This is called hypothesis testing.

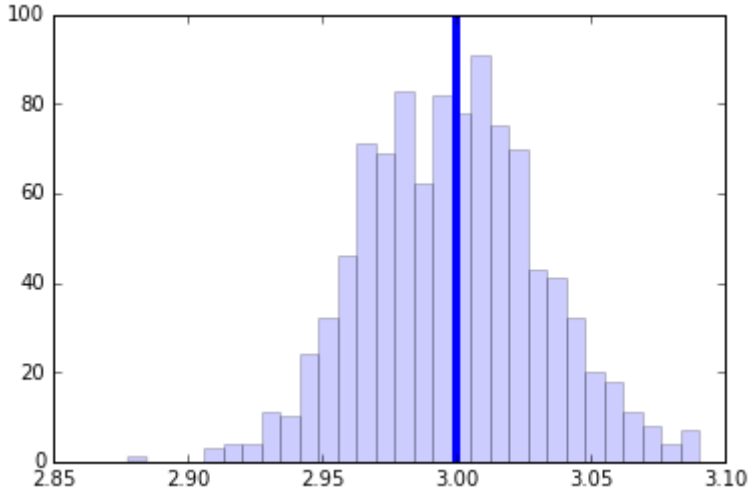
## Confidence intervals

A p confidence interval is defined as a range of values such that with p probability range the range will contain the true unknown value of the parameter. The range is defined in terms of lower and upper limits estimated from the data.

This can be demonstrated with multiple samples (realizations) of the same line.

In [479]:
```
Ns=1000   # NUMBER OF IMAGINARY REALIZATIONS OR SAMPLES
N = 1000
beta0_hat_a =np.zeros(Ns)
beta1_hat_a =np.zeros(Ns)

i=0
for k in np.arange(Ns):
    X,Y = GenerateDataLinearFun(N, sigma=.3)
    # Estimate the coefficients
    beta0_hat_a[i], beta1_hat_a[i] = FitLinearModel(X,Y)
    i=i+1


plt.hist(beta1_hat_a, bins=30, alpha=0.2)  ;

plt.axvline(x=3.0, linewidth=4)
```

Out[479]: <matplotlib.lines.Line2D at 0x126cfebe0>



In real life we only have one realization of data (bootstrapping that will see later is another way to get around this).

The best we can do is to define the confidence intervals based on the SE (standard error on the estimate)

For linear regression, the 95% confidence interval for $\beta_1$ can be approximated as:

$$\hat{\beta}_1 \pm 2SE(\hat{\beta}_1)$$

and similarly for $\beta_0$

$$\hat{\beta}_0 \pm 2SE(\hat{\beta}_0)$$

In [482]:
```
# CALCUALTE FOR ONE REALIZATION THE SE AND THE 95% CONFIDENCE INTERVAL
N=1000

X,Y = GenerateDataLinearFun(N)
beta0_hat, beta1_hat= FitLinearModel(X,Y)
Y_hat= PredictLinearModel(X,beta0_hat, beta1_hat )

e = Y-Y_hat
RSS = np.sum( e**2)
RSE = np.sqrt( RSS/(N-2))

sigma = RSE

SE_b1= np.sqrt( sigma**2/np.sum((X-np.mean(X))**2))

print( "Standard error on beta1" , SE_b1)
print( "Estimate value of beta1" , beta1_hat )
print( "95% confidence interval is: [", beta1_hat-2*SE_b1,beta1_hat+2*SE_b1, "]")
print( "Correct value is: ", 3)
```

```
Standard error on beta1 1.0555104145
Estimate value of beta1 2.50771564434
95% confidence interval is: [ 0.396694815352 4.61873647333 ]
Correct value is:  3
```

# Hypothesis Testing

The standard erros can be used to perform hypothesis testing.

The most interesting in our case is if $\beta_1 = 0$ or in other words there is no relationship between X and Y

H0 : Null hypothesis. There is no relationship between X and Y: $\beta_1 = 0$ H1 : Alternative hypothesis: There is some relationship between X and Y

We need to determine if $\beta_1$ is far from zero and how far is far. Intuitively is how many SEs far from zero is the estimate. [see example above, with the imaginary parallel universes]

The standard way of doing this is with t-statistic given by

$$\hat{t} = \frac{\hat{\beta_1} - 0}{SE(\hat{\beta_1})}$$

t follows the t-distribution but practically (for any sample size larger than 30) it converges to the normal distribution. One calculates the probability of having $t > \hat{t}$ or the p-value. [more on this on the next lecture]

## Model Accuracy

So far we never questioned the assumption that $f(X) = \beta_0 + \beta_1 X$.

We saw above that if the sample size is large and the range of X is large and $\sigma$ is small, the estimated coefficients converge to the true one.

But how about if the linear assumption is no good? Can we still estimate the model accuracy?

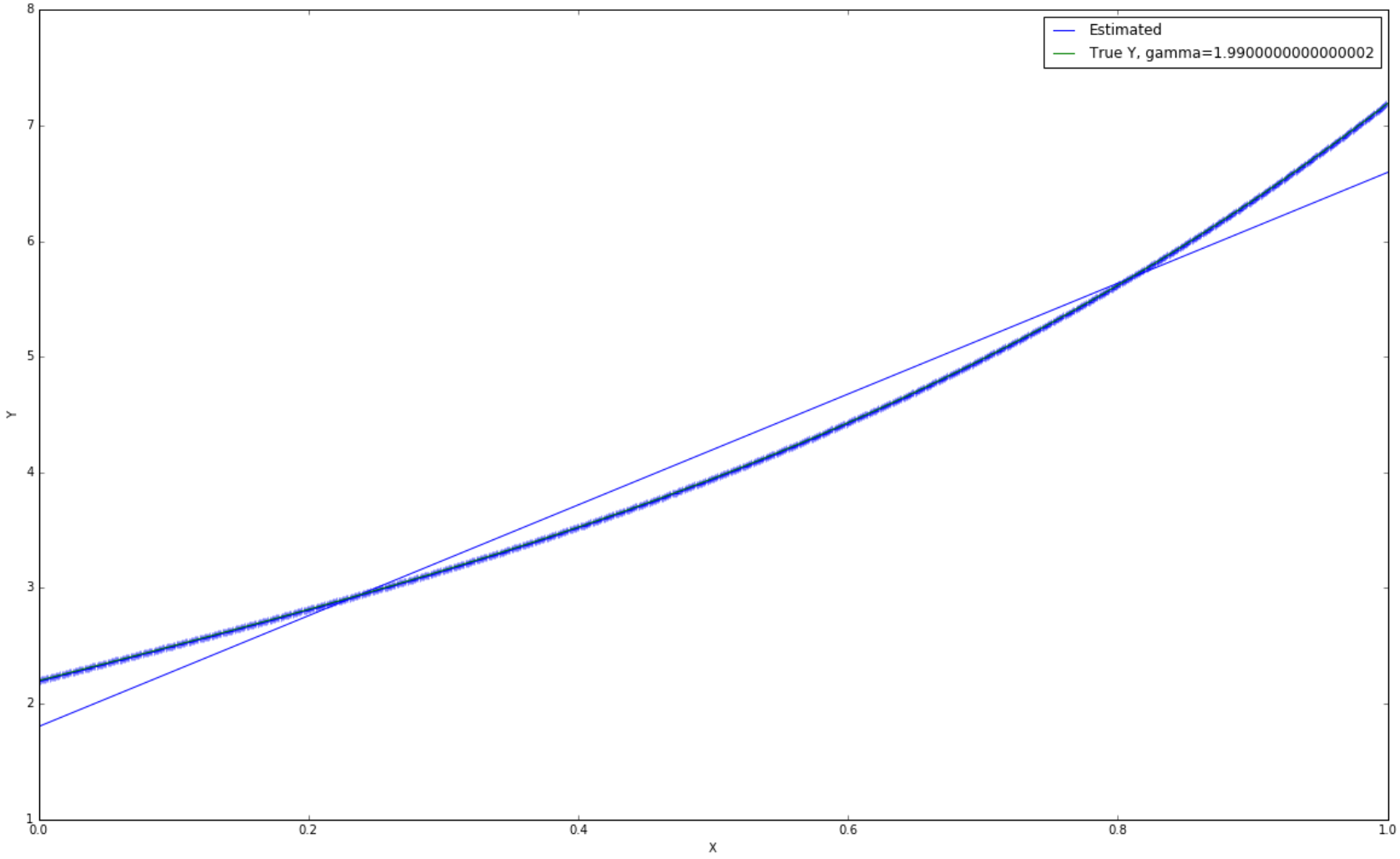
Due to the presence of error terms $\epsilon$, even if we knew the true regression line, we would not be able to perfectly predict Y from X.

In [400]:
```
from ipywidgets import interact,  FloatSlider, RadioButtons
```

In [486]:
```
def plotit(gamma):
    # GENERATE DATA THAT ARE NOT FROM A LINEAR FUNCTION
    N = 1000 # number of points
    X,Y = GenerateDataNotLinearFun(N, sigma=0.001, alpha=gamma)

    beta0_hat, beta1_hat = FitLinearModel(X,Y)


    print("estimated coefficients", beta0_hat, beta1_hat)

    # estimate y
    Y_hat   = PredictLinearModel(X, beta0_hat, beta1_hat)

    # calculate the RSE, R2
    RSS = np.sum((Y-Y_hat)**2)
    RSE = np.sqrt(np.sum((Y-Y_hat)**2))/(N-2)
    TSS = np.sum( (Y-np.mean(Y))**2)

    plt.figure(figsize=(20,12))
    R2 = 1 - RSS/TSS
    plt.plot(X,Y, 'b+')
    plt.plot(X, Y_hat, label="Estimated" )
    plt.plot(X, beta0 + beta1 * X + gamma*X*X*X, label="True Y, gamma="+str(gamma) )
    plt.legend(loc='best')
    plt.xlabel('X')
    plt.ylabel('Y')

    print("RSE="+str(RSE))
    print("R2="+str(R2))

interact(plotit, gamma=(-5,10,.01))
```

```
estimated coefficients 1.80224644385 4.79155092682
RSE=0.00716465239392
R2=0.974023466679
```

Out[486]: <function __main__.plotit>



# $R^2$

The RSE is an estimate of the standard error, or the average deviation of the response from the true value. Recall:

$$RSE = \sqrt{\frac{RSS}{n-2}}$$

- It is considered a measure of the lack of fit of the model
- Absolute measure since it is in units of Y

Alternatively we can use $R^2$ statistics that is propotion of deviation in units of variance

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

where $TSS = \sum_i (y_i - \bar{y})^2$ is the total sum of squares, or the deviation of Y before we regress and RSS is the deviation left after the regression.

The ration tells as how well the regression performed to remove deviations on Y

## Why not correlation?

If what we are looking is a measure of the relation between X and Y then why not use the correlation.

$$Cor(X,Y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

It so happens for simple linear regression one can prove that $R^2$ and $Cor(X,Y)^2$ are identical. [Not the case for other cases]

# Residuals

Linear regresion is a great tool and it is used very extensively. Detemining the coeffients is generally easy, but understanding a) the accuracy of the model b) the fitness of the model and c) how well we determined the coefficients d) the importance of the variables and e) the validity of the linearity are among the things you will be spending most of the time.

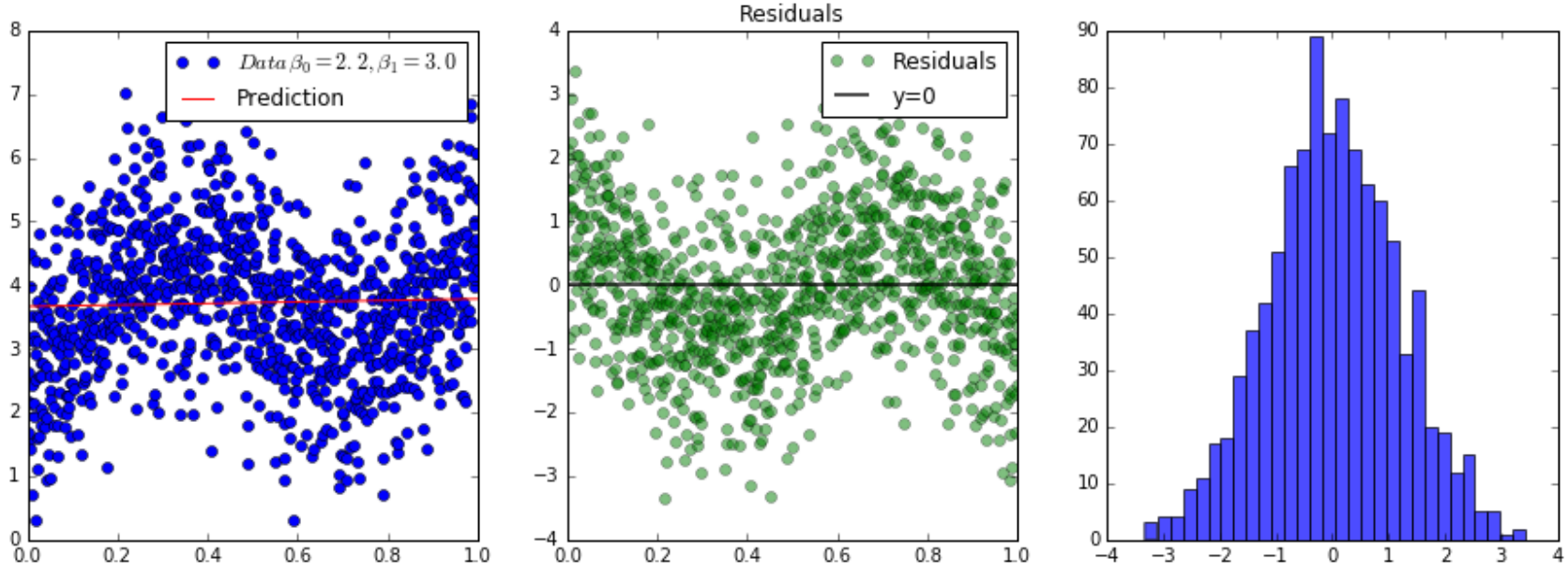
Another important tool is to look at the residuals themselves,

$$e_i = y_i - \hat{y}_i$$

In the limit that $\hat{Y}$ is the exact correct values of $Y = \beta_0 + \beta_1 X$ then the residuals are mainly equal to the noise term $\epsilon_i$ and therefore should have the distribution of the noise. If we believe the error is iid normally distributed then the residuals themselves should follow that distribution.

In [247]:

In [496]:
```
N=1000
X, Y = GenerateDataNotLinearFun(N, alpha=1.4)
X = X.reshape((N,1))

regression =Lin_Reg()
regression.fit(X,Y)
Y_h=regression.predict(X)

# Residuals
e = Y_h-Y

fig, ((ax1, ax2, ax3)) = plt.subplots(1, 3, figsize=(15, 5))
ax1.plot(X,Y, 'o', label=r'$Data \, \beta_0=2.2, \beta_1=3.0$')
ax1.plot(X, Y_h, color='r', label="Prediction")
ax1.legend()

ax2.set_title("Residuals")
ax2.plot(X,e, 'go', alpha=0.5, label='Residuals')
ax2.axhline(y=0, color='black', linewidth=2.0, alpha=0.7, label='y=0')
ax2.legend()

ax3.hist( e, bins=30, alpha=0.7);
```



In [ ]: