ASTR 119: Session 19

Central Limit Theorem

Linear Regression, Revisited

Outline

1) Final projects: **Due Tuesday, December 15, 2020 at 3pm.**
2) Visualization of the Day
3) Central Limit Theorem
4) Final project organization time
5) Save your work to GitHub

Final Projects

1) You will be asked to perform one of four final projects, working in a **group of up to 4 people of your choosing from your SECTION**. If you would like to be assigned to a group, please let me know immediately.

2) Please notify me of your group by **8:00am Thursday Nov 19**. Anyone who does not respond by **8:00am Thursday** letting me know their group or their desire to be assigned to a group will work on their own final project.

3) You will choose from one of the following 4 project topics, described in the following slides:
    1) Damped, driven pendulum
    2) Logistic map and chaos
    3) Astronomical source detection
    4) Monte Carlo Integration

4) Groups must select a final project topic by **Tues Nov 24**. Detailed instructions for each project will be distributed Thursday Nov 19.

5) Groups should be organized through GitHub, invite your group members and us (TAs) as collaborators on your project. Tag us (TAs) when your final project is ready to grade.

6) Each python module should indicate which student authored which part of the code.

7) Final projects are due (tags must happen by) **Tuesday, December 15, 2020 at 3pm.**

CENTRAL LIMIT THEOREM

Much of statistical analysis is underpinned by a remarkable theorem known as the *central limit theorem* (CLT). This theorem states that:

The distribution of* mean *values for an experiment tends to be a Normal distribution (i.e., a Gaussian normalized to unit area), regardless of the distribution from which the mean was derived.

CENTRAL LIMIT THEOREM

Much of statistical analysis is underpinned by a remarkable theorem known as the *central limit theorem* (CLT). This theorem states that:

The distribution of* mean *values for an experiment tends to be a Normal distribution (i.e., a Gaussian normalized to unit area), regardless of the distribution from which the mean was derived.

The CLT goes on to make a more quantitative statement: If your data sample has an average value of μ with a variance σ^2, then the distribution of means will also be centered around μ with a variance of $s^2 = \sigma^2/N$ where N is the number of data points in each sample. In other words, the mean value distribution can *always* be represented in the following form:

$$P(x) = \frac{1}{\sqrt{2\pi s^2}} \exp\left[-\frac{(x-\mu)^2}{2s^2}\right]$$

CENTRAL LIMIT THEOREM

To demonstrate the CLT, consider the following experiment: Toss a coin 100 times. You would expect to see heads 50 times, but in reality sometimes you will get more and sometime you will get less. The CLT tells us that the mean value of the number of heads then has a Gaussian distribution whose mean is 50, despite the fact that the distribution of possible outcomes is by no means Gaussian (it is in fact simply either heads or tails each time).

CENTRAL LIMIT THEOREM

To demonstrate the CLT, consider the following experiment: Toss a coin 100 times. You would expect to see heads 50 times, but in reality sometimes you will get more and sometime you will get less. The CLT tells us that the mean value of the number of heads then has a Gaussian distribution whose mean is 50, despite the fact that the distribution of possible outcomes is by no means Gaussian (it is in fact simply either heads or tails each time).

The reason that the CLT is so important is that in most cases, we are interested in the mean value of a given experiment, so the CLT tells us that no matter what the actual data distribution of the experiment is, the distribution of mean value (the thing we often care about) will always be a Gaussian! This enables us to treat experimental errors, and quantify how well a model fits a set of experimental data, using only a well-behaved single function.

CENTRAL LIMIT THEOREM

Let's use the example of a head/tails experiment to demonstrate the CLT!

Here's what we'll do:

1) Draw 10 uniformly distributed random numbers [0,1), measure the mean.
2) Repeat this experiment 10000 times, histogram the measured means.

The CLT says the distribution of the means will be a Gaussian. What will the variance of the distribution of means be?

CENTRAL LIMIT THEOREM

Central Limit Theorem Demonstration

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Set up a series of random uniform distributions

The central limit theorem states that for random processes, if we measure the mean of independent random variates the distribution of the means of those distributions will be Gaussianly distributed. This result is independent of the character of the random distribution. We can show this by using a collection of uniform random variates, and measuring the means many times.

In [4]:
```
n_exp = 10000 #number of experiments
n_draw = 10   #number of random variates per experiment
n_bins = 100  #number of bins in histogram of the uniform variate means
x_min  = 0.15 #minimum of histogram range
x_max  = 0.85 #maximum of histogram range

means = np.zeros(n_exp)  #means from each experiment
```

CENTRAL LIMIT THEOREM

Let's perform the experiment

In [5]:
```
#loop over the number of experiments
for i in range(n_exp):

    #pull 10 random variates from a uniform distribution
    z = np.random.uniform(0.,1.,n_draw)

    #record the mean
    means[i] = np.mean(z)
```

Define a function to plot a Gaussian

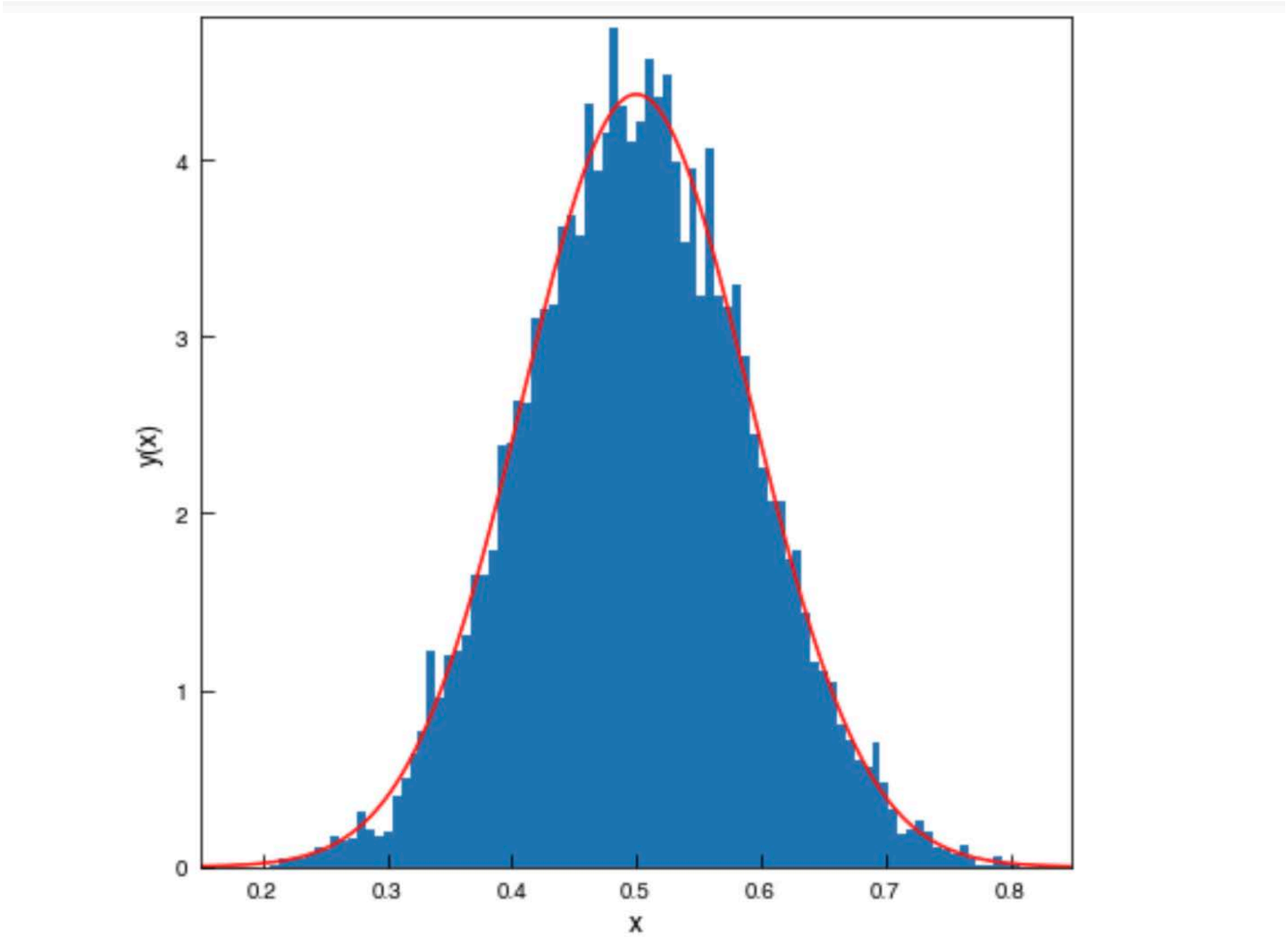
In [8]:
```
def gaussian(x,mu,sigma):
    return 1./(2.0*np.pi*sigma**2)**0.5 * np.exp(-0.5*((x-mu)/sigma)**2)
```

CENTRAL LIMIT THEOREM

Now let's histrogram the means and plot them

In [14]:
```
fig = plt.figure(figsize=(7,7))
y_hist, x_hist, ignored = plt.hist(means, bins=n_bins, range=[x_min,x_max], density=True)
xx = np.linspace(x_min,x_max,1000)

#set the RMS of the gaussian
sigma = 1./12.**0.5 / 10.**0.5

plt.plot(xx,gaussian(xx,0.5,sigma),color="red")
plt.ylim([0,1.1*gaussian(0.5,0.5,sigma)])
plt.xlim([x_min,x_max])
#plt.gca().set_aspect(20)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.show()
```

CENTRAL LIMIT THEOREM



LINEAR REGRESSION

A common circumstance is that your model for noisy data is linear. In this case, the goal is to determine the slope and intercept parameters (that fully specify the linear relation) that best describe a particular data set. This is known as *linear regression.*

Consider the function

$$Y = mX + b$$

where *Y* is the dependent variable, *X* is the control variable, and *m* and *b* are the slope and intercept of the relation that describes the two. The experiment generates a set of *N* data points (X_i, Y_i). The goal is to determine the best estimate for *m* and *b* given this data set, so that it becomes possible to formulate a predictive theory of *Y* given some arbitrary value of *X*.

We did this almost on day one, but we didn't know it!!

LINEAR REGRESSION

How can we determine the best-fit estimates for *m* and *b*? The "best fit" values are defined as the *m* and *b* that minimize the difference between the function *Y* and the data points. This difference can be written as the sum of $|Y_i - (mX_i+b)|$ over all the data points *i*.

For convenience, we will work with the square of this quantity (minimizing the square is equivalent to minimizing the absolute value). So we want to minimize the function

$$\mathcal{M} = \sum_{i=1}^{N} [Y_i - (mX_i + b)]^2$$

To *minimize* a function, we want its derivative with respect to *m* and *b* to be zero. Of course, in principle this can also produce a *maximum* of the function, so we have to be a bit careful. In practice the maximum is always unbounded so no solution exists. In other words, we can always find parameters that are a *worse* fit to the data!

LINEAR REGRESSION

So, let us take a the partial derivative of *M* with respect to *m* and *b,* and set them equal to 0:

$$\frac{\partial M}{\partial m} = -2 \sum_{i=1}^{N} \left[Y_i - (mX_i + b) \right] X_i = 0$$

$$\frac{\partial M}{\partial b} = -2 \sum_{i=1}^{N} \left[Y_i - (mX_i + b) \right] = 0$$

LINEAR REGRESSION

So, let us take a the partial derivative of *M* with respect to *m* and *b,* and set them equal to 0:

$$\frac{\partial M}{\partial m} = -2 \sum_{i=1}^{N} \left[Y_i - (mX_i + b) \right] X_i = 0$$

$$\frac{\partial M}{\partial b} = -2 \sum_{i=1}^{N} \left[Y_i - (mX_i + b) \right] = 0$$

We want to solve these two equation for *m* and *b*. This is not in principle difficult, but it does require a bit of algebra. For instance, we can solve each of the two equations for *b:*

$$b = \frac{\sum_{i=1}^{N} X_i Y_i - m \sum_{i=1}^{N} X_i^2}{\sum_{i=1}^{N} X_i}$$

$$b = \frac{1}{N} \left(\sum_{i=1}^{N} Y_i - m \sum_{i=1}^{N} X_i \right)$$

LINEAR REGRESSION

By equating these two, it is possible to solve for *m* to obtain

$$m = \frac{N\sum X_i Y_i - \sum X_i \sum Y_i}{N\sum X_i^2 - (\sum X_i)^2}$$

The intercept *b* can be obtained similarly, but in fact it is easy to see that it must follow the equation

$$b = \bar{Y} - m\bar{X}$$

where $\bar{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$ **is the average value of Y, and similarly for X.**

These equations hold only when all measurements have the same uncertainty. To tackle the general problem, we will need a more sophisticated method.

LINEAR REGRESSION

Example of linear regression

First we import numpy and matplotlib as usual.

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

LINEAR REGRESSION

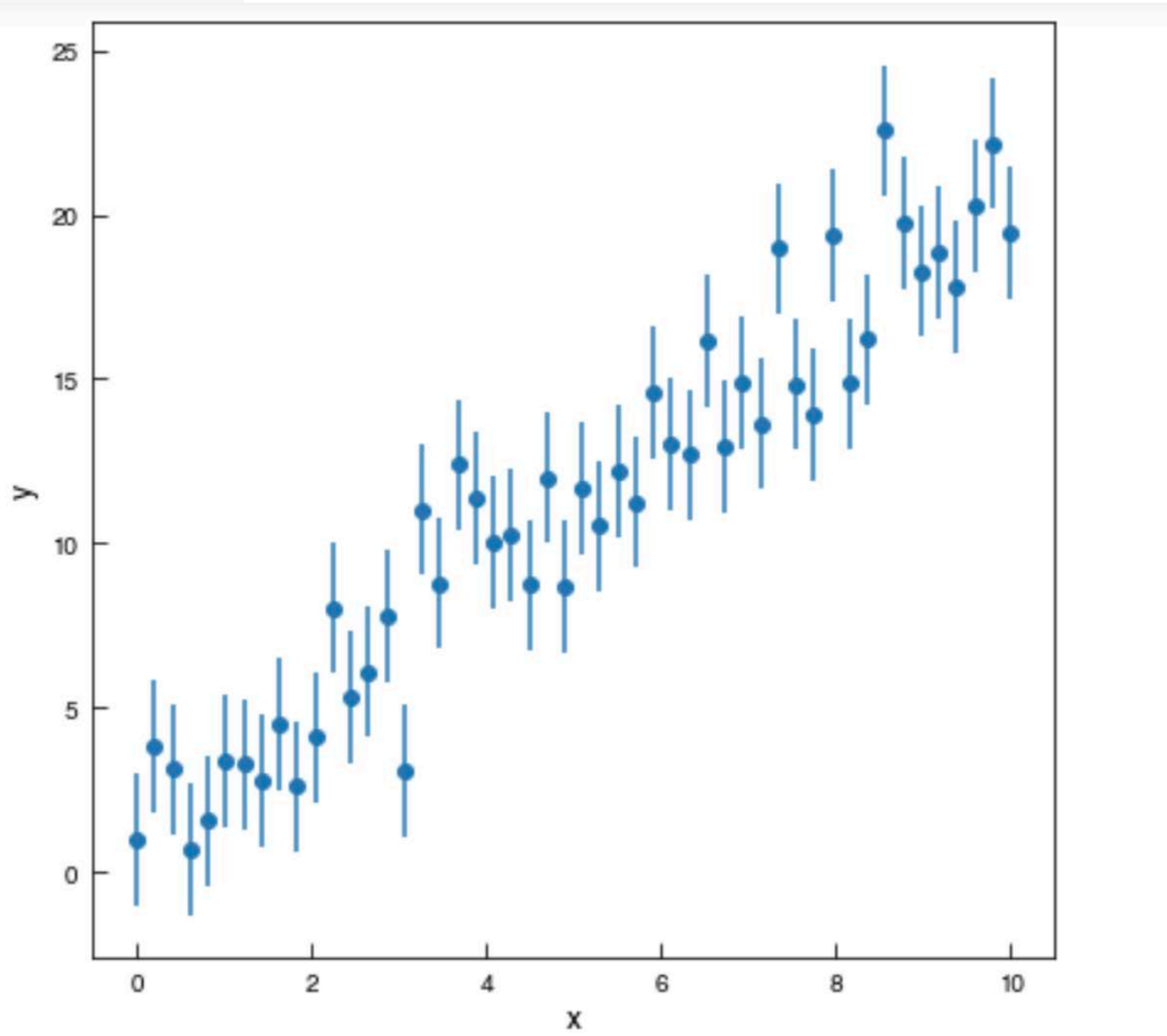
Now, let's generate some random data about a trend line.

In [2]:
```
#set a random number seed
np.random.seed(119)

#set number of data points
npoints = 50

#set x
x = np.linspace(0,10.,npoints)

#set slope, intercept, and scatter rms
m = 2.0
b = 1.0
sigma = 2.0

#generate y points
y = m*x + b + np.random.normal(scale=sigma,size=npoints)
y_err = np.full(npoints,sigma)
```

LINEAR REGRESSION

Let's just plot the data first

In [3]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o')
plt.xlabel('x')
plt.ylabel('y')
```



LINEAR REGRESSION

Method #1, polyfit()

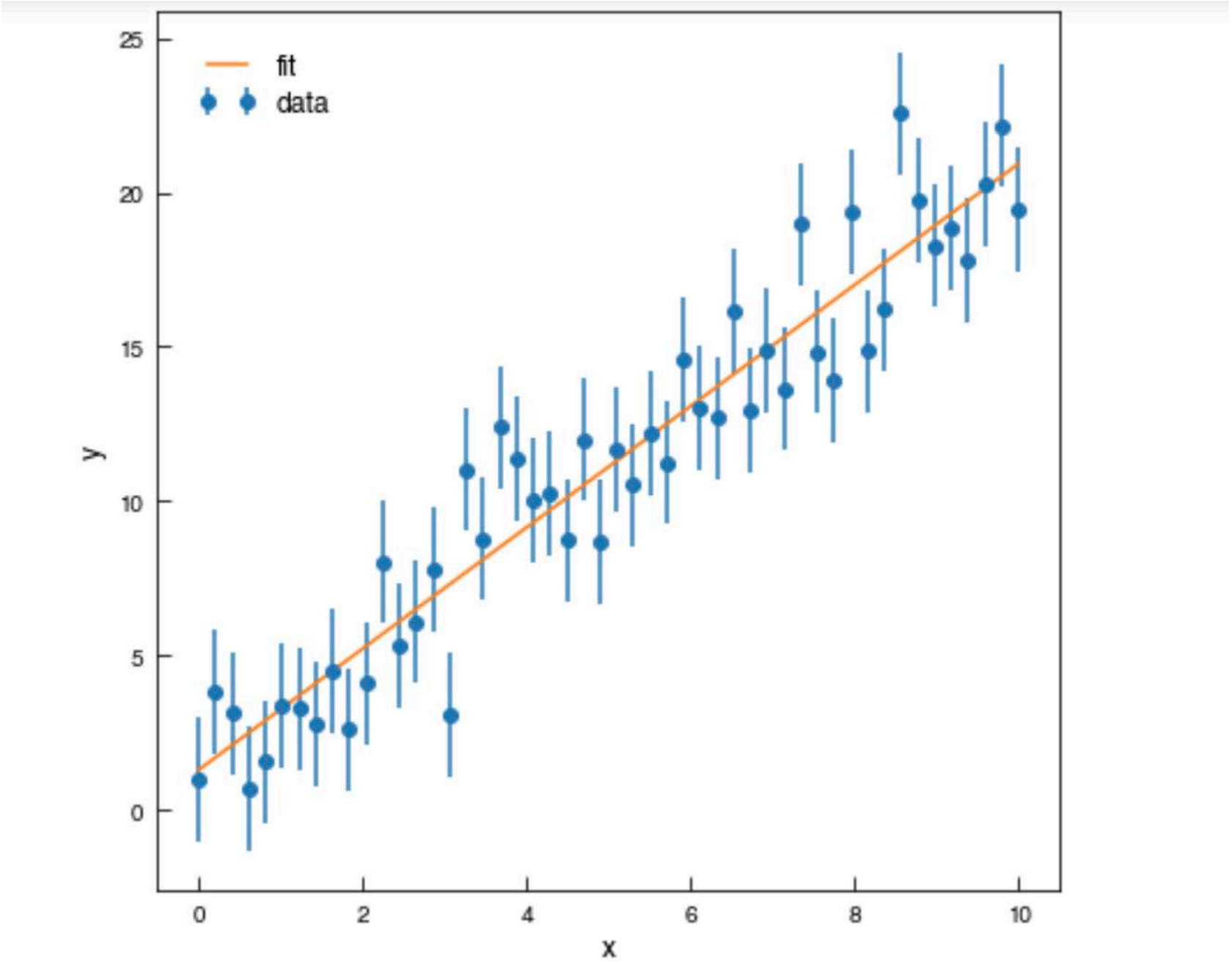
In [4]:
```
m_fit, b_fit = np.poly1d(np.polyfit(x, y, 1, w=1./y_err))  #weight with uncertainties
print(m_fit, b_fit)

y_fit = m_fit * x + b_fit
```

```
1.96340434704 1.2830106813
```

Plot result

In [5]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```

LINEAR REGRESSION



LINEAR REGRESSION

A new hope: linear regression

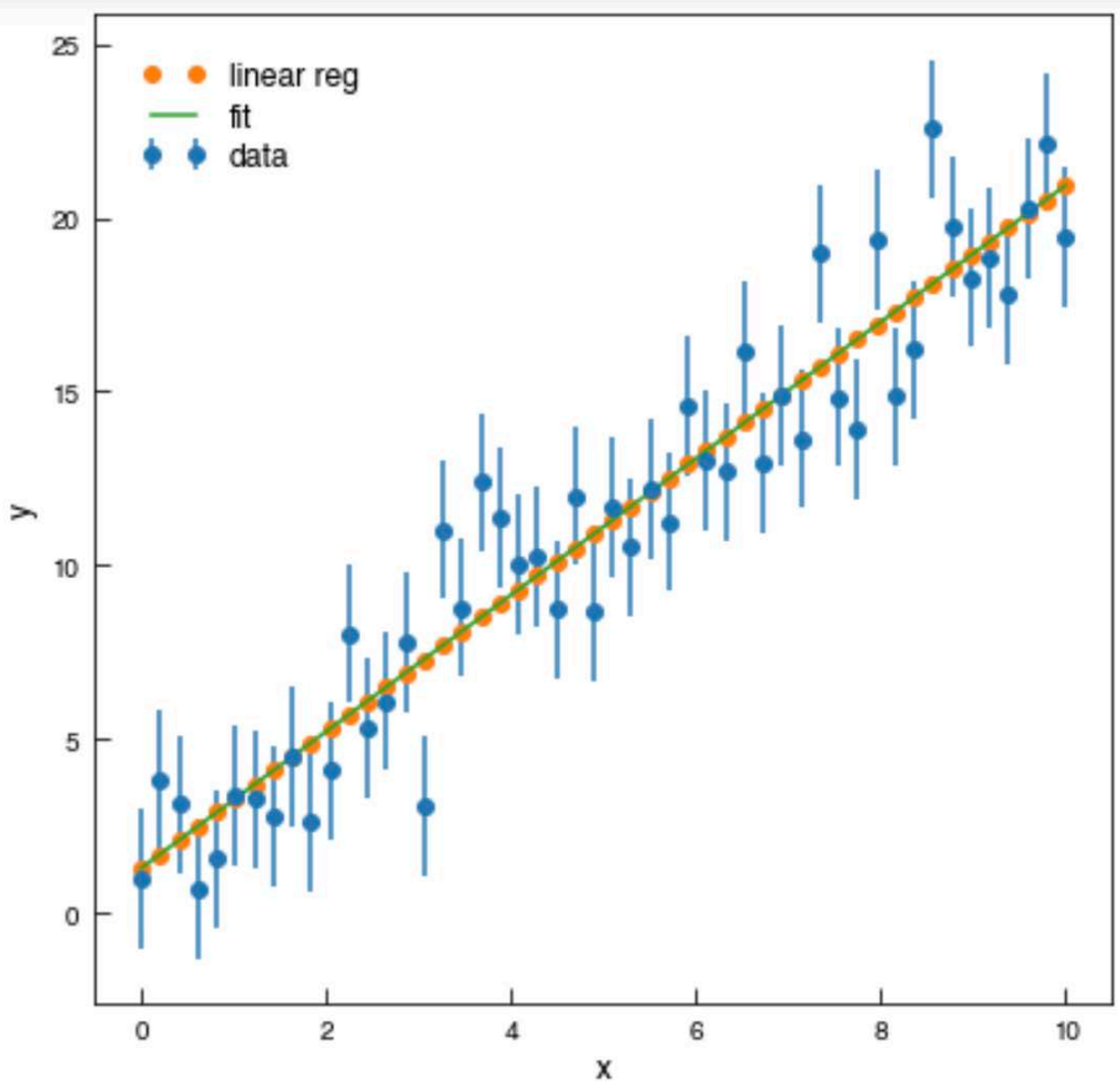
In [8]:
```
m_A = 0.0
m_B = 0.0
m_C = 0.0
m_D = 0.0

m_A = np.sum(x*y)
m_B = np.sum(x)*np.sum(y)
m_C = np.sum(x*x)
m_D = np.sum(x)**2

m_fit_lr = (float(npoints)*m_A - m_B)/(float(npoints)*m_C - m_D)

y_mean = np.mean(y)
x_mean = np.mean(x)

b_fit_lr = y_mean - m_fit_lr*x_mean

y_fit_lr = m_fit_lr * x + b_fit_lr

print(m_fit_lr,b_fit_lr)
```

```
1.96340434704 1.2830106813
```

LINEAR REGRESSION

Plot the result

In [11]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit_lr,'o',label='linear reg')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```



Save Your Work

Make a GitHub project “astr-119-session-19”, and commit the programs you made today.