

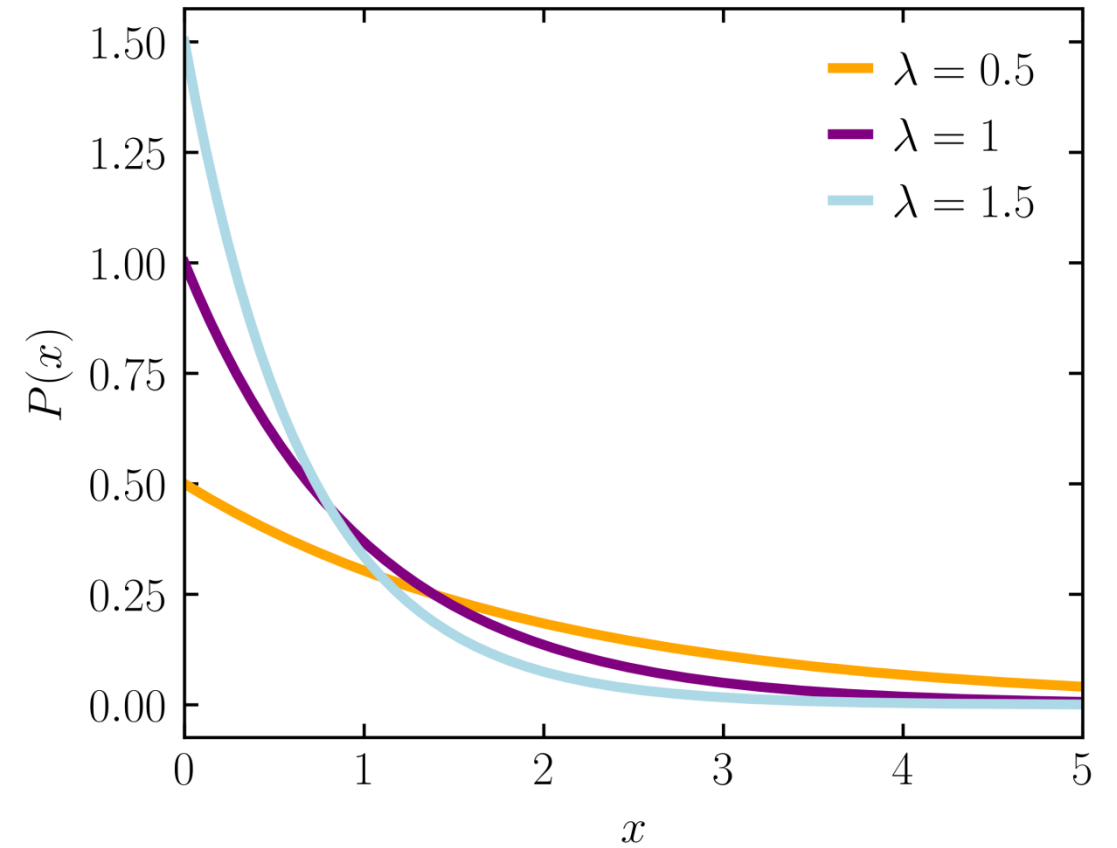
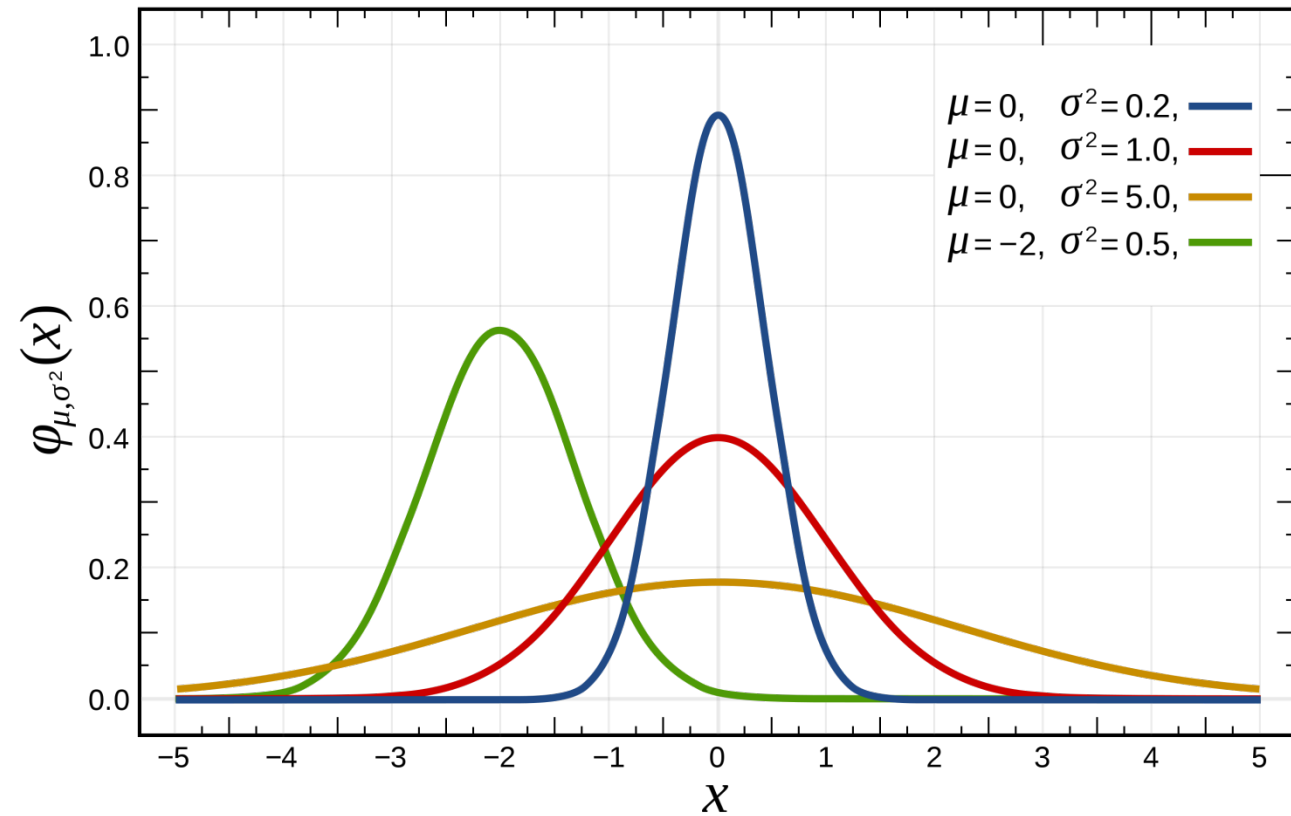
ENGR 298: Engineering Analysis and Decision Making – Distribution Fitting

Dr. Jason Forsyth
Department of Engineering
James Madison University

Overview

- Data can be drawn some testing, experiments, samples... that represent some underlying physical process.
- Helpful to map those samples to a known statistical distribution so they can be better 'described'. Can express properties beyond min/max to average, deviation, skew...etc.
- Placing values in a statistical realm also allows for statistical test (t-test) to make strong claims about improve of one process versus another.

Distributions have shapes and parameters



Fitting Data to Distributions

- Many statistical packages provide approaches to fit a dataset to a particular distribution.
- For a given distribution, the key parameters are estimated.
 - Normal: (μ, σ)
 - Exponential: (λ)
 - Log-Normal: (μ, σ)
- Can attempt to *any* data to *any* distribution. Will explore later how to verify with χ^2 test (hopefully)

Can use both numpy and scipy for these operations

- Numpy provides ability to sample various random distributions.

```
# generate those samples with numpy  
# https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html  
normal_samples = np.random.normal(loc=desired_mu, scale=desired_std, size=num_samples)
```

- Scipy can perform a similar operation but need to provide an array for the random variable where the PDF can be sampled

```
# generate an array of values from -3 to 3 then sample the PDF  
x = np.linspace(start=-3, stop=3, num=10000)  
norm.pdf(x)
```

- Scipy supports significantly more statistics operations and so is generally used for curve fitting, analysis...etc.

Some important points

- Since probability distributions have a variety of parameters, numpy and scipy attempt to abstract each one to general/higher-level params.

`scipy.stats.norm` = `<scipy.stats._continuous_distns.norm_gen object>`

[\[source\]](#)

A normal continuous random variable.

μ

σ

The location (**loc**) keyword specifies the mean. The scale (**scale**) keyword specifies the standard deviation.

`scipy.stats.expon` = `<scipy.stats._continuous_distns.expon_gen object>`

[\[source\]](#)

An exponential continuous random variable.


λ

A common parameterization for **expon** is in terms of the rate parameter **lambda**, such that **pdf** = **lambda** * **exp(-lambda * x)**. This parameterization corresponds to using **scale** = **1 / lambda**.

$$= 1/\lambda = \beta$$

Some import points

- Recommended to import submodules directly. Do not take the long path of `sp.stats.norm...etc`



```
import numpy as np
from scipy.stats import norm, expon
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import platform
```

Let's try this out...

- **First:** sample a distribution randomly for a fixed number of points.
- **Second:** calculate the appropriate statistics for that distribution or use a package to fit a curve to those points.
- **Third:** compute a 'true' distribution based upon ideal values.
- **Fourth:** plot the two curves to visually inspect fit.

Sample the normal distribution...

```
# desired average of normal distribution
desired_mu = 0

# desired standard deviation
desired_std = 1

# number of samples to take
num_samples = 10

# generate those samples with numpy
# https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html
normal_samples = np.random.normal(loc=desired_mu, scale=desired_std, size=num_samples)
```



μ



λ

Calculate relevant statistics... (pretty easy)

```
# generate those samples with numpy  
# https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html  
normal_samples = np.random.normal(loc=desired_mu, scale=desired_std, size=num_samples)  
  
# now, find the mean and std of this distribution  
sample_mean = np.mean(normal_samples)  
sample_std_dev = np.std(normal_samples)
```

Compute a 'true' distribution...

```
# same the 'correct' distribution to generate another plot
x = np.linspace(start=-3, stop=3, num=10000)
y = norm.pdf(x, loc=desired_mu, scale=desired_std)
plt.plot(x, y, label='Ideal Normal')
```

linspace generates a vector of linear spacing between a start and stop point with a certain number of values. Good for just creating an evenly spaced x-axis

Use the norm() imported from sp.stats to generate probabilities of X for distribution

Now plot everything...

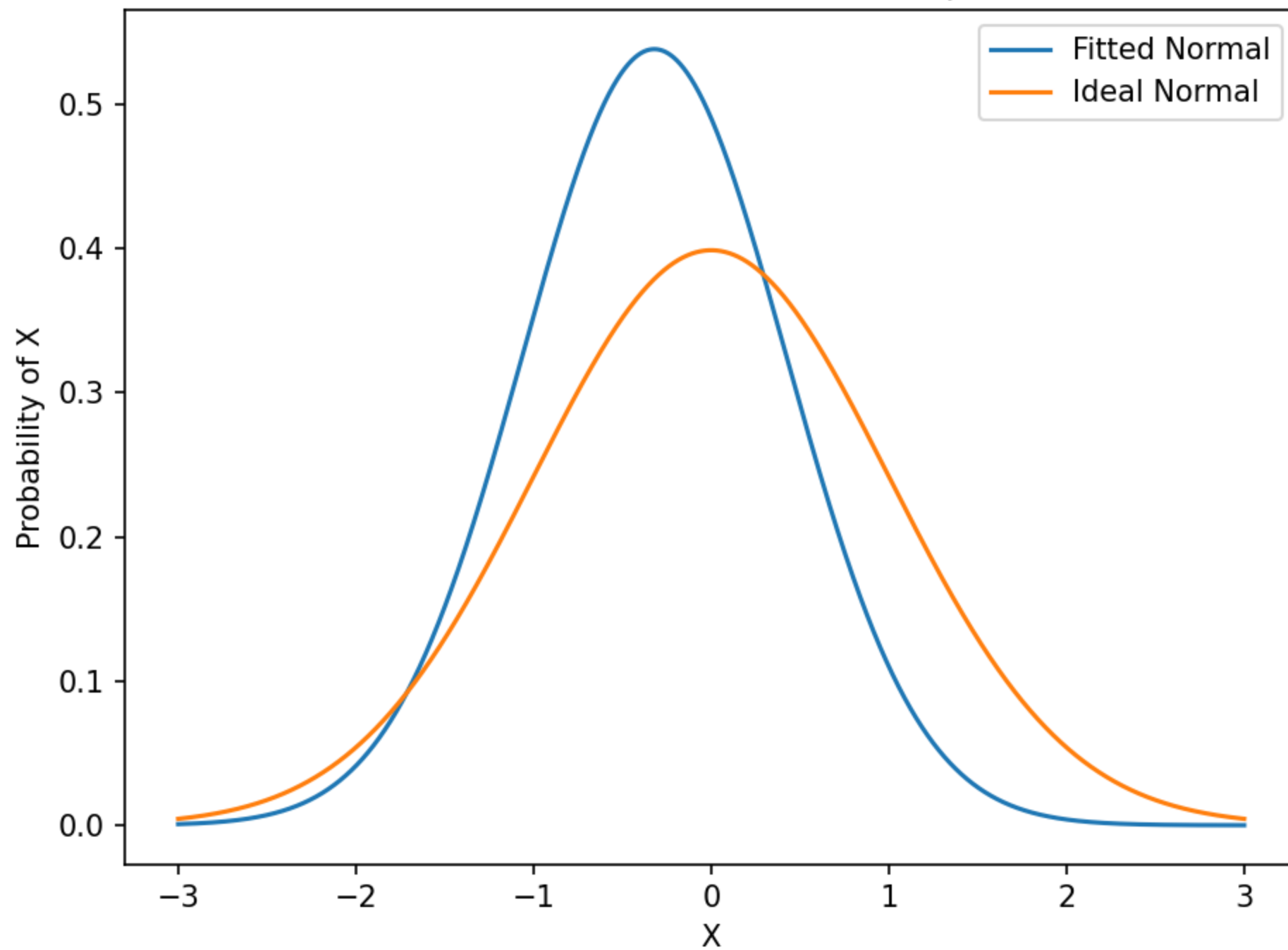
```
# plot the sampled distribution
plt.plot(x, y, label='Fitted Normal')
plt.title('Fitted Normal Distribution for ' + str(num_samples) + ' Sampled Points')
plt.xlabel('X')
plt.ylabel('Probability of X')

# sample the 'correct' distribution to generate another plot
x = np.linspace(start=-3, stop=3, num=10000)
y = norm.pdf(x, loc=desired_mu, scale=desired_std)
plt.plot(x, y, label='Ideal Normal')

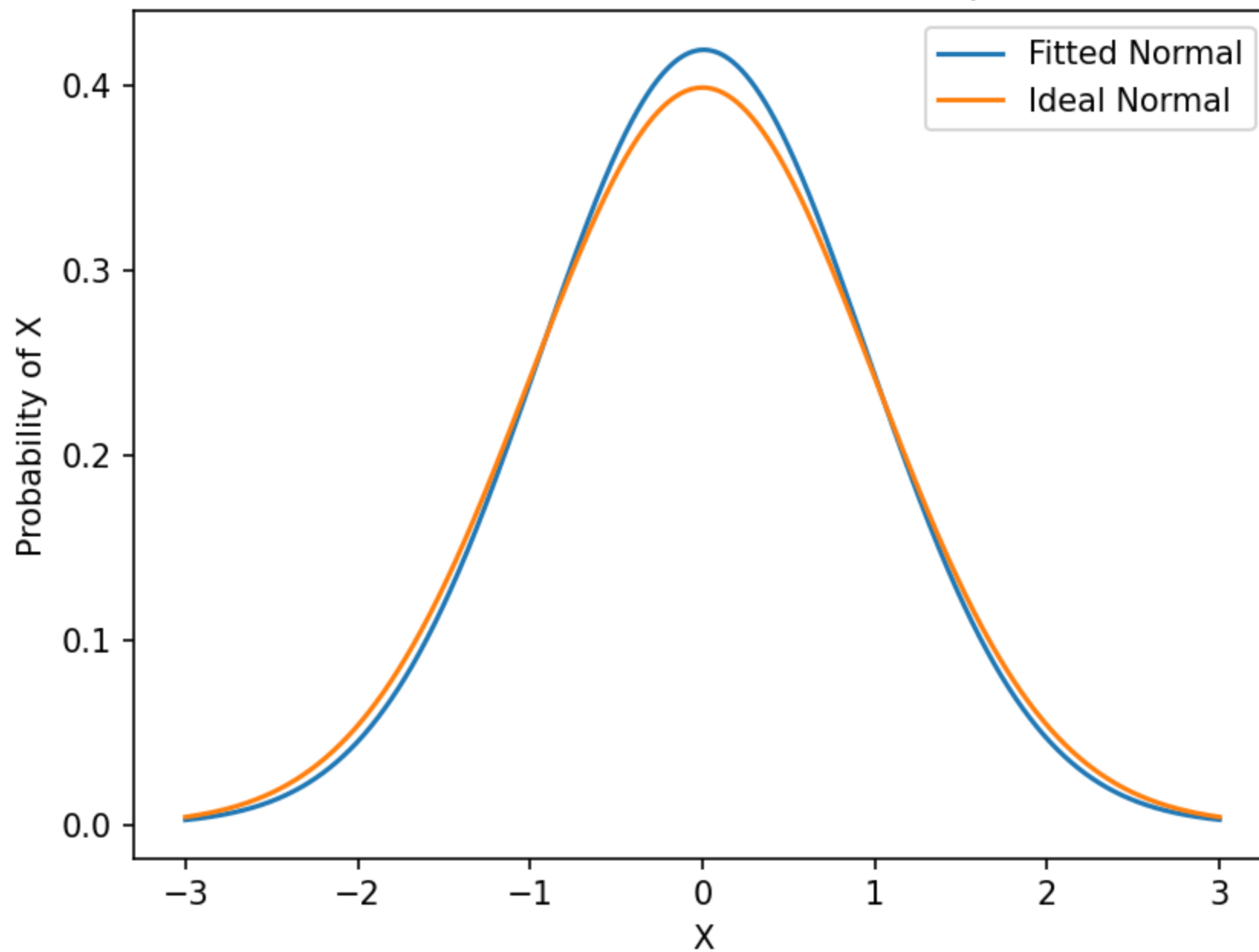
# include a legend
plt.legend()

# show the plot
plt.show()
```

Fitted Normal Distribution for 10 Sampled Points



Fitted Normal Distribution for 100 Sampled Points



If samples are from distribution, then repeated sampling will eventually converge on the 'true' distribution

Where mean and std don't apply

- For some distributions there may not be built-in functions to directly calculate the appropriate parameters.
- No implementation for exponential λ or log-normal (μ, σ) , however, they could be manually estimated/calculated.
- Generally will utilize the `fit()` for a particular distribution to estimate the various parameters. Function is annoying in return types so it may take some practice to figure out what values are of interest.

scipy.stats.rv_continuous.fit

```
rv_continuous.fit(data, *args, **kws)
```

[\[source\]](#)

Return estimates of shape (if applicable), location, and scale parameters from data. The default estimation method is Maximum Likelihood Estimation (MLE), but Method of Moments (MM) is also available.

Parameters: **data** : *array_like*

Data to use in estimating the distribution parameters.

Returns: **parameter_tuple** : *tuple of floats*

Estimates for any shape parameters (if applicable), followed by those for location and scale. For most random variables, shape statistics will be returned, but there are exceptions (e.g. `norm`).

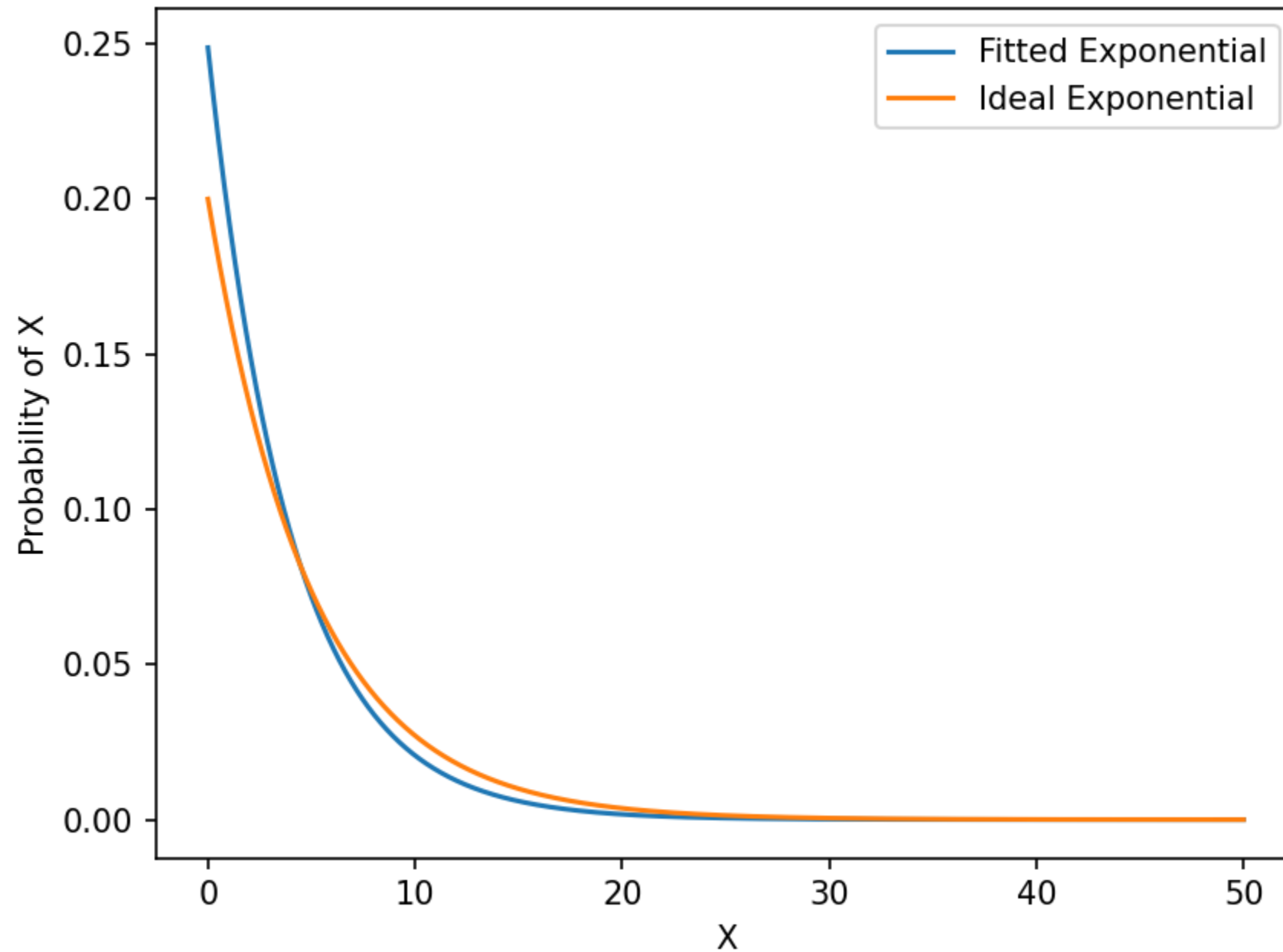
Let's attempt to fit an exponential distribution. Only has one parameter (λ) but function will return location and scale.

```
!# we can't now just find the mean and standard deviation. The points must be 'fit' to the curve  
# use the fit() method to estimate the shape parameters for the exponential distribution  
!# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv\_continuous.fit.html#scipy.stats.rv\_continuous.fit  
(fit_loc, fit_scale) = expon.fit(exponential_samples)  
     $\beta$   
# pull out beta from the fitted distribution  
fit_beta = fit_scale
```

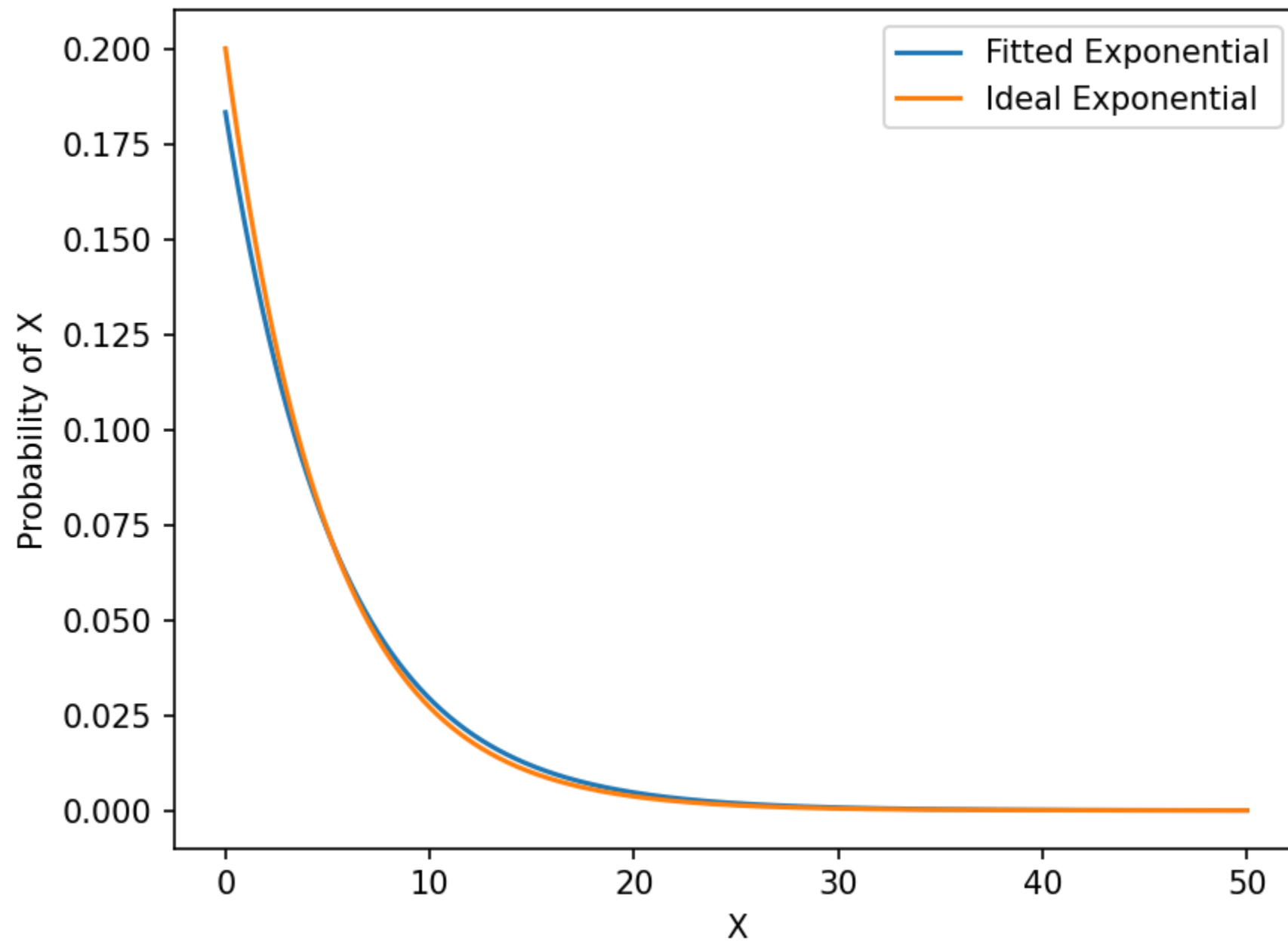
A common parameterization for **expon** is in terms of the rate parameter **lambda**, such that **pdf = lambda * exp(-lambda * x)**. This parameterization corresponds to using **scale = 1 / lambda**.

$$= 1/\lambda = \beta$$

Fitted Exponential for 5 Sampled Points



Fitted Exponential for 50 Sampled Points



Ok, so let's try this with some real data...

RSI Data

- Previously determined RSI from force plate and accelerometer. Data will have error because accelerometer is an 'approximation' of the force plate data.
- Assume that the error is normally distributed. Most things are (they have some average and move around the variance).
- Can load RSI data via Pandas, compute fit parameters, and plot curve.

```
# load up the RSI data results with force plate and accelerometer results
df = pd.read_csv('../data/drop-jump/all_participant_data_rsi.csv')

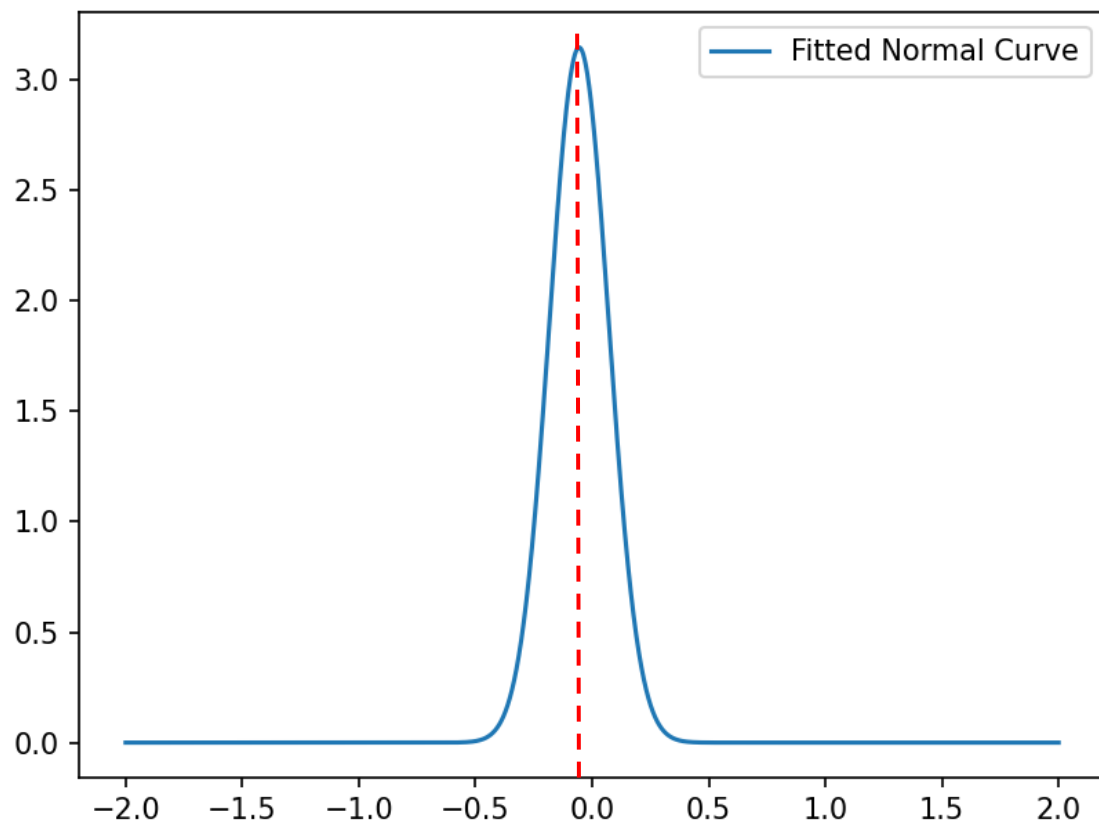
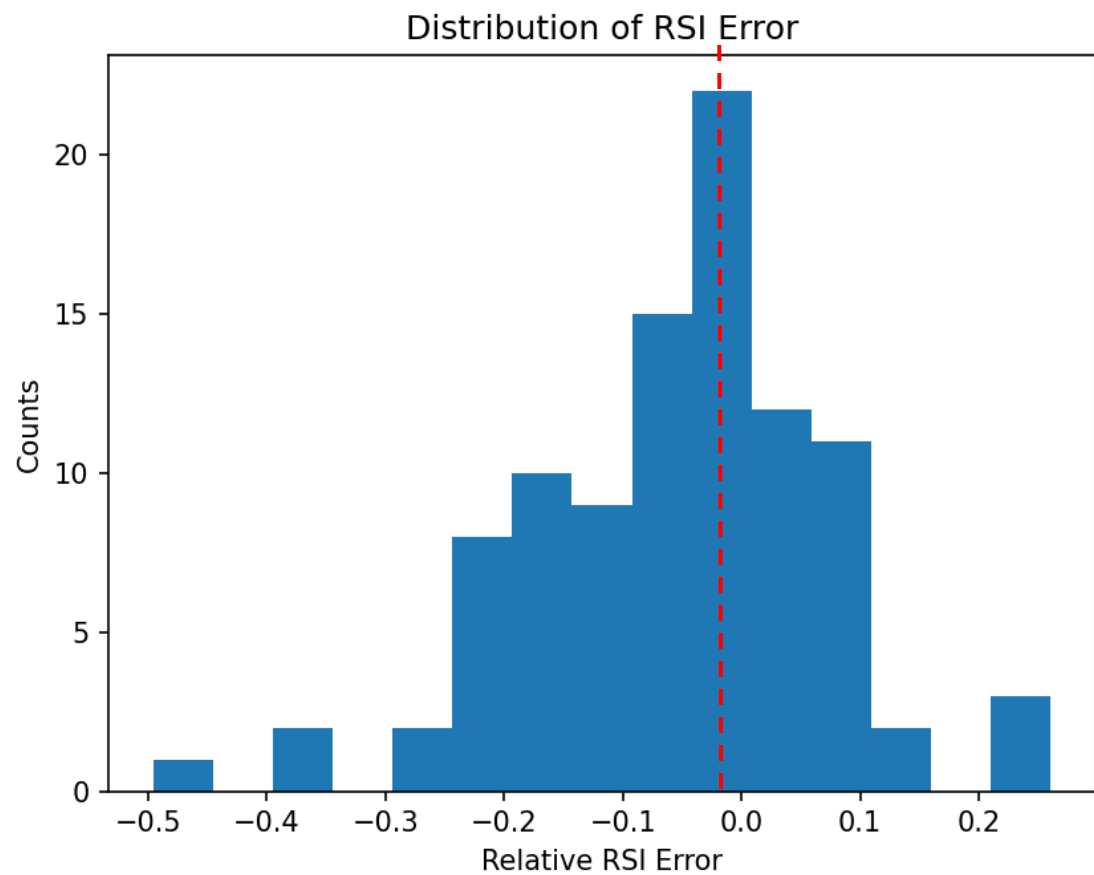
# pull out the force plate column
force_plate_rsi = df['force_plate_rsi'].to_numpy()

# pull out the accelerometer column
accel_rsi = df['accelerometer_rsi'].to_numpy()

# calculate error assuming that force plate was computed correctly
error = force_plate_rsi - accel_rsi

# generate a histogram to see what the data looks like
plt.hist(error, bins=15, label='RSI Error')
plt.xlabel('Relative RSI Error')
plt.ylabel('Counts')
plt.title('Distribution of RSI Error')
plt.show()

# now, let's map the RSI error to the normal distribution with scipy fit
(fitted_mean, fitted_std) = norm.fit(error)
```

Fitted parameters are $\mu = -0.055$ and $\sigma = 0.126$

We can now better assess our error with confidence intervals

7.1.4. What are confidence intervals?

How do we form a confidence interval?

The purpose of taking a random sample from a lot or population and computing a statistic, such as the mean from the data, is to approximate the mean of the population. How well the sample statistic estimates the underlying population value is always an issue. A confidence interval addresses this issue because it provides a range of values which is likely to contain the population parameter of interest.

Confidence levels

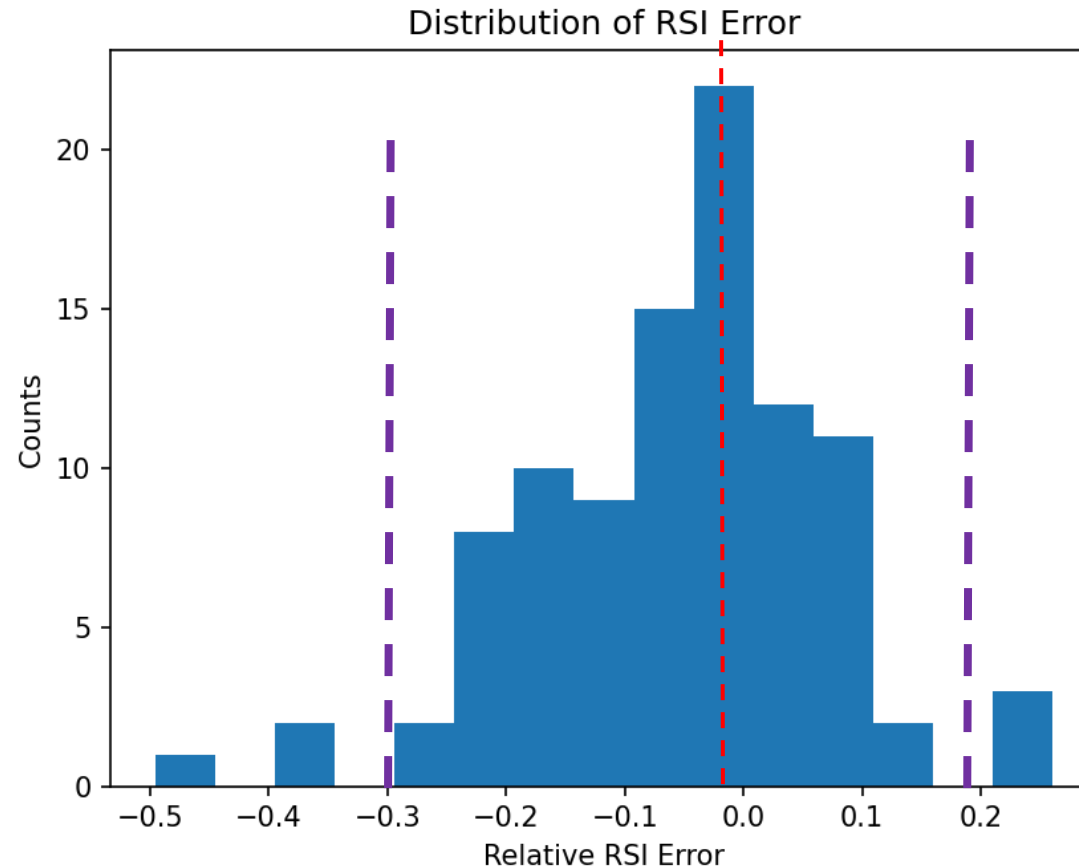
Confidence intervals are constructed at a *confidence level*, such as 95 %, selected by the user. What does this mean? It means that if the same population is sampled on numerous occasions and interval estimates are made on each occasion, the resulting intervals would bracket the true population parameter in approximately 95 % of the cases. A confidence stated at a $1 - \alpha$ level can be thought of as the inverse of a significance level, α .

Pretty easy to do... 😊

```
# since we have the fitted data, we can now determine useful parameters such as the 95%  
# confidence interval with scipy  
(lower_ci, upper_ci) = norm.interval(alpha=0.95, loc=fitted_mean, scale=fitted_std)
```

Parameters for fitted distribution are $\mu = -0.055852411598661694$ and $\text{std} = 0.12674647477702344$
For the fitted RSI data. The 95% confidence interval is: (-0.30427093732904203 , 0.19256611413171865)

Showing mean and confidence intervals.



Summary

- Curve fitting and statistics are powerful tools to add to engineering analysis.
- Very challenging to get correct in applying appropriate methods and drawing conclusions from analysis. Would always have a more stat-friendly person double check the work.
- Will move towards T-tests and χ^2 goodness of fit to provide further power.