**Distribution types**

dt.BERNOULLI

Parameter:

PMF:

f(x) =

MLE:

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.BERNOULLI)

x.generateSamples(p=0.4, size=1000, seed=123)

x.MLE()

>>> [0.404]

x.MLE(use\_minimizer=True, p0=0.1)

>>> [0.40398438]

samples = x.getSamples()

print(samples)

>>> [0 1 1 0 0 0 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. An initial guess p0 is required. Otherwise, the closed form expression for the MLE is used.

dt.BINOMIAL

Parameters:

PMF:

MLE:

, assuming is known. We do not handle the case where both and are unknown.

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.BINOMIAL)

x.generateSamples(n=10, p=0.56, size=1000, seed=123)

x.MLE(n=10)

>>> [0.5608]

x.MLE(n=10, use\_minimizer=True, p0=0.1)

>>> [0.56078125]

x.GOF(n=10)

>>> [7.31194908, 12.59158724]

x.GOF(n=10, use\_minimizer=True, p0=0.1)

>>> [ 7.313976, 12.59158724]

samples = x.getSamples()

print(samples)

>>> [5 7 7 5 5 6 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. An initial guess p0 is required. Otherwise, the closed form expression for the MLE is used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.GEOMETRIC

Parameter:

PMF:

MLE:

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.GEOMETRIC)

x.generateSamples(p=0.78, size=1000, seed=123)

x.MLE()

>>> [0.78740157]

x.MLE(use\_minimizer=True, p0=0.5)

>>> [0.78740234]

x.GOF()

>>> [2.18746235 5.99146455]

x.GOF(use\_minimizer=True, p0=0.5)

>>> [2.18749391 5.99146455]

samples = x.getSamples()

print(samples)

>>> [1 1 1 1 1 1 3 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. An initial guess p0 is required. Otherwise, the closed form expression for the MLE is used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.UNIFORM

Parameters:

PDF:

MLE:

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.UNIFORM)

x.generateSamples(a=-3.2, b=5.6, size=1000, seed=123)

x.MLE()

>>> [-3.19927939, 5.59048197]

x.GOF()

>>> [8.37, 15.50731306]

samples = x.getSamples()

print(samples)

>>> [2.92892883e+00 -6.81973852e-01 -1.20370721e+00 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.NORMAL

Parameters:

PDF:

for all real numbers

MLE:

, where is the sample variance

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.NORMAL)

x.generateSamples(mean=-3.2, var=2.1, size=1000, seed=123)

x.MLE()

>>> [-3.25733388, 2.10330896]

x.MLE(use\_minimizer=True, mean0=-2, var0=0.3)

>> [-3.25732685, 2.1032861]

x.GOF()

>>> [17.98869157, 28.86929943]

x.GOF(use\_minimizer=True, mean0=-2, var0=0.3)

>>> [17.98876893, 28.86929943]

samples = x.getSamples()

print(samples)

>>> [-4.77322821e+00 -1.75470914e+00 -2.78992520e+00 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. Initial guesses for mean0 and var0 are required. Otherwise, the closed form expressions for the MLE and are used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.EXPONENTIAL

Parameters:

PDF:

MLE:

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.EXPONENTIAL)

x.generateSamples(lambd=3.2, size=1000, seed=123)

x.MLE()

>>> [3.20543007]

x.MLE(use\_minimizer=True, lambd0=2)

>> [3.20543004]

x.GOF()

>>> [16.30352699, 31.41043284]

x.GOF(use\_minimizer=True, lambd0=2)

>>> [16.30352666, 31.41043284]

samples = x.getSamples()

print(samples)

>>> [3.72585045e-01 1.05333588e-01 8.04012750e-02...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. An initial guess lambd0 is required. Otherwise, the closed form expression for the MLE is used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.GAMMA

Parameters:

PDF:

Log-Likelihood:

MLE:

We use the method in <https://en.wikipedia.org/wiki/Gamma_distribution#Maximum_likelihood_estimation> to compute the MLE parameters and . See the gammaMLE function in Helpers.py for the code implementation.

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.GAMMA)

x.generateSamples(a=3, b=0.21, size=1000, seed=123)

x.MLE()

>>> [3.13206806, 0.20397199]

x.MLE(use\_minimizer=True, a0=3.5, b0=0.1)

>>> [3.13202843, 0.20397514]

x.GOF()

>>> [10.72661972, 27.58711164]

x.GOF(use\_minimizer=True, a0=3.5, b0=0.1)

>>> [10.72664892 27.58711164]

samples = x.getSamples()

print(samples)

>>> [0.26411361 0.97637216 0.38409334 1.338683 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. Initial guesses a0 and b0 are required. Otherwise, the method in Helpers.py is used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

dt.WEIBULL

Parameters:

PDF:

Log-Likelihood:

MLE:

We use the method in Simulation & Modeling (Law 5e, pp. 290-292) to compute the MLE parameters and . See the weibullMLE function in Helpers.py for the code implementation.

**Example**:

import simufit as sf

from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.WEIBULL)

x.generateSamples(a=3, b=0.21, size=1000, seed=123)

x.MLE()

>>> [3.00653034, 0.2099457]

x.MLE(use\_minimizer=True, a0=6, b0=0.5)

>>> [3.00650806, 0.20994661]

x.GOF()

>>> [14.4849622 26.2962276]

x.GOF(use\_minimizer=True, a0=6, b0=0.5)

>>> [14.48441248, 26.2962276]

samples = x.getSamples()

print(samples)

>>> [0.22267823 0.14614756 0.13356426 0.1950631 ...]

**Notes:**

* To generate multiple sample sets using generateSamples with a particular *starting seed*, instead use np.random.seed() (before calling generateSamples) and do not set a seed in generateSamples. Setting a seed in generateSamples will produce the same samples repeatedly if the method is called multiple times.
* Calling the MLE method with use\_minimizer=True will use a Nelder-Mead optimizer to minimize the negative log likelihood. Initial guesses a0 and b0 are required. Otherwise, the method in Helpers.py is used.
* The first value returned by the GOF method is and the second is , where , is the number of intervals, and is the number of parameters being estimated. An internal function checks that there are at least 5 samples in a given interval, and merges any intervals that do not satisfy this condition.

**GUI Examples**

**Example 1: Fitting samples generated with the Simufit library**

import simufit as sf

from simufit import DistributionType as dt

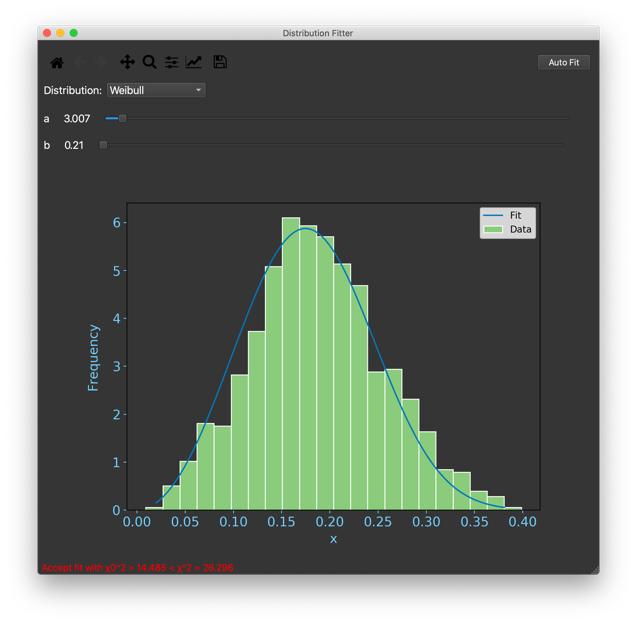
x = sf.Distribution()

x.setDistribution(dt.WEIBULL)

x.generateSamples(a=3, b=0.21, size=1000, seed=123)

x.fit()

The following window will pop up. Slide the **a** and **b** sliders manually to tune the fit, or press the Auto Fit button. The result of a chi-square goodness-of-fit test will be displayed at the bottom of the window.



**Example 2: Fitting data (from an unknown distribution) loaded from a CSV file**

import simufit as sf

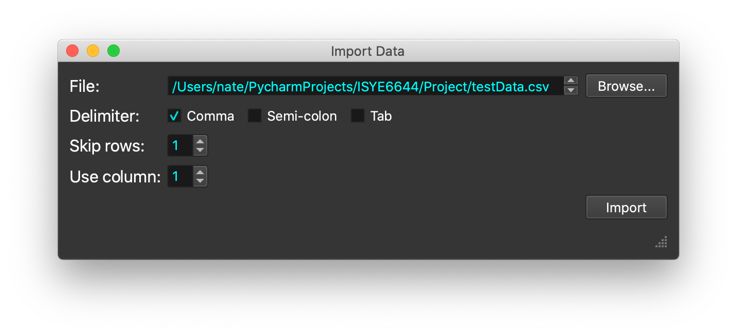
from simufit import DistributionType as dt

x = sf.Distribution()

x.setDistribution(dt.UNKNOWN)

x.fit()

From the File menu of the Distribution Fitter window, select Import Data. Select the file to be loaded using the Browse button, and choose the appropriate options. Press the Import button and close the Import Data window.



You can now try to fit your data to one of the selectable distributions (Bernoulli, Binomial, Normal, etc.)

