Simufit

A Distribution Simulator and Identification Python Package

ISYE-6644 - Simulation and Modeling for Engineering and Science

ABSTRACT

Simufit is a Python Package designed for the simulation and identification of the following currently supported statistical distributions: Bernoulli, Binomial, Geometric, Uniform, Normal, Exponential, Gamma and Weibull. Providing both API endpoints and a GUI interface, sample sets can be generated using Simufit or loaded from external datasets (CSV) into a Distribution object for exploration and identification. To fit the samples to a distribution, first maximum likelihood estimation (MLE) parameters are calculated followed by a chi-square goodness of fit (GOF) test, the results of which are then scored and ranked against all supported distributions. Each distribution was tested in 300 trials with an overall accuracy rating of 0.76 (Weibull, 0.69; Bernoulli, 0.99; Binomial, 0.66; Geometric, 0.75; Uniform, 0.81; Normal, 0.88; Exponential, 0.64; Gamma, 0.69). These results include tests containing very small sample sizes (as low as 20 observations); the overall performance increases significantly when larger sample sizes of 1000 or greater are used, 0.86 (Weibull, 0.95; Bernoulli, 1.0; Binomial, 0.75; Geometric, 0.91; Uniform, 0.94; Normal, 0.95; Exponential, 0.55; Gamma, 0.94). A WHL package is available for distribution to Mac, Windows 10, and Linux built using a continuous integration process on the Azure DevOps platform (Azure DevOps Services, 2020).

1 PROJECT OVERVIEW

1.1 Background

The identification of a statistical distribution is a key aspect to data analysis; without properly establishing this fundamental component any evaluation of the data has the potential to be flawed, not

including subsequent issues with any simulation models based on the samples collected. Similar to other distribution-fitting software, such as ExpertFit by Averill M. Law & Associates, Simufit is designed to evaluate a collection of samples, and using statistical approaches attempt to identify the best fit distribution (Law, 2015). To accomplish this, Simufit is designed as a Python package to be implemented across Mac, Windows 10 and Linux operating systems installed using a Python Wheel Package file (WHL) supporting a Python 3 environment.

This software package was designed as a deliverable to satisfy the project requirements for Program-Oriented Problem #20 as defined in the "ISyE 6644 – Fall 2020 – Projects!" document for the Simulation and Modeling for Engineering and Science class ISYE-6644 at the Georgia Institute of Technology.

1.2 Approach

An Agile Method approach was utilized as part of this project to support the need and timeline. A collection of requirements was generated based off of the problem statement and grouped into features. To support the software development lifecycle (SDLC), Microsoft Azure DevOps was utilized for requirements management, automated builds, version control and continuous integration (Azure DevOps Services, 2020). Each feature and requirement was loaded into the Azure DevOps platform and assigned to a given sprint in the development timeline. The project was split into 4 development sprints and 1 finalization sprint for documentation and bug triage.

All project text communication was maintained through the Slack messaging application and video communication was managed through the Zoom video conferencing software (Slack, 2020; Zoom, 2020). Status update meetings took place with Zoom in general on a weekly basis, while text communication was almost daily. Team members resided in both the United States and Japan, so communication and coordination were imperative.

Statistical methods used in the application were derived primarily from Chapter 6 of the Law book with additional support being cited within the application where appropriate (Law, 2015).

1.3 Documentation

This document is designed to provide an overview of the Simufit software application, its development process, user instructions, evaluation of the product and finally a summary depicting the success of the project and future intentions. A collection of supporting items have been added to the appendix at the end of this document to provide more information about some of the content described in this document.

In addition to this report, a README file is included with the Simufit package for users. This file includes instructions on how to load and use the WHL file, import the module, generate and identify distributions, use the command-line interface and GUI.

2 DEVELOPMENT

2.1 Requirements

A collection of requirements was established based on the project preface, which includes the goal of fitting a specific distribution type to a given observation Set. This process should utilize a maximum likelihood estimate (MLE) and a goodness of fit test (GOF) for at least the following distribution types: Bernoulli, Geometric, Exponential, Gamma, Normal, Weibull. Additional requirements were added for improved user interact and stretch goals to improve the user experience. Each collection of requirements was grouped into a specific feature collection: Package, User Interface, Import, Generate Distributions, Summary Statistics, Distribution Identification and Documentation. The collection of requirements can be found in the Appendix (Appendix 7.2) grouped by feature, each requirement is denoted as essential by an asterisk (*), since stretch goals are not necessary for this project they do not have an asterisk.

2.2 Architecture

Simufit is comprised of 7 modules: Dataset, dist_generator, Distribution, Helpers, IDistribution, Report and Types. In addition to these a “tests” module contains a “Tests” class with methods specific to each distribution type to evaluate performance. The architecture diagram for Simufit can be found in Appendix 7.1. The following sections describe the functionality and purpose of each module within the package.

2.2.1 *Distribution, IDistribution*

The package is designed to be used either as a text interface or with a graphical user interface (GUI). The two different approaches provide the user with the ability to manually fit curves in a graphical interface to the observations, use mathematical evaluations only, or a hybrid approach for distributions that are found to be more challenging. The foundational class of Simufit is the Distribution class, as it is a stateful object that has the ability to store a given set of samples, generate a set of samples, read a set of samples and identify the best fit distribution. Structurally the Distribution class implements the IDistribution abstract class used to define the base functionality required of any Distribution implementation. When data are loaded from an external CSV or text file initially into the Distribution class the initial fit is defined as “Unknown”. The method “identifyDistribution” executes the MLE for each supported distribution (when applicable) and generates a GOF Chi-squared value and associated probability. Each of these values are assigned a score based on a proportional distance from their probability value and an ordered list of potential best fit is stored in the Distribution object. This class allows a user to load and analyze data, or create a Distribution of a particular type and generate data specific to it (e.g. make the Distribution object a Binomial and then generate samples). Users who instead activate “run_fitter()” can bypass the Distribution class and instead temporarily load data into the GUI for a manual evaluation. To load a dataset from a CSV file into a Distribution object requires use of “readCsv” method; this sample set will be loaded into the object and set to a distribution type of “Unknown”.

2.2.2 *dist_generator*

In both use cases, the same distribution methods for data generation, MLE and GOF are called. Each distribution type is defined as a class within the dist_generator file, where it is assigned methods specific to that type (e.g. MLE, GOF). The Distribution class instantiates classes as need be, whether during assignment or detection, to allow the Distribution object to be treated as if a specific distribution type was instantiated (e.g. x.setDistribution(dt.WEIBULL); x.MLE();). The GUI will allow a user to select a given distribution from a pulldown menu, this will instantiate the class of the appropriate distribution type (i.e. Binomial, Gamma, etc..) with the samples loaded. An auto-fit button allows the system to pass the variables loaded in the GUI to the class directly without use of the Distribution class.

2.2.3 *Display*

The Display module supports the display features of the application, with the only exception being the graphical approach to reading CSV files (e.g. Dataset module). This includes the "run_fitter" method which opens a blank GUI and allows a user to import a CSV with data. The user can then manually select different distribution types and auto-fit, or manually adjust the curve to the data shown in a histogram. Also within the Display module is the Histogram class, which allows a user to create a histogram of the data or create an overlapped visualization of two sample collections. The Histogram class is only accessible through an instantiated Distribution object.

2.2.4 *Dataset*

The Dataset module provides a basic visual interface for selecting a CSV file while using the GUI. The dialog box that appears provides the user options to browse for the file, use column headers, select a specific delimiter and also identify the column containing the data. The samples loaded are then displayed on the existing graph. Dataset passes the loaded the dataset into a Distribution object as an "Unknown" distribution type.

2.2.5 *Helpers*

The Helpers module provides support to the dist_generator class for identifying the MLE of some distributions when they do not require a minimized optimization. Additionally, it assists in the identification of bins necessary for calculating a Chi-squared GOF test (i.e. ensures that each bin has a minimum of five samples, and adjusts the bin edges accordingly when possible).

2.2.6 *Report*

The Distribution class has an internal parameter, "_distribution_report", which is designed to collect a report of each distribution evaluated against a given sample set for best fit. The Report module contains the DistributionReport class which is a single instance of one of the items to fill the this list. For each evaluation a DistributionReport is instantiated containing basic information about the distribution (e.g. Name, whether it is Continuous or Discrete, Number of Unique Elements) and the resultant evaluation information such as MLE and GOF. Evaluations are also provided a score based on the GOF, if applicable.

For the reports, a scoring system was devised to compare GOF results for best fit. This score helps quantify the relative distance between the Chi-squared value and probability $\rho \equiv \chi^2_{\alpha,k-1-s}$ based on the appropriate degrees of freedom ($k-1-s$). Another way to state this would be: the smaller χ^2 is relative to ρ, the higher the score. Data are normalized against ρ for direct comparison across all distribution types, when applicable. The score value is calculated as follows:

$$score(\chi^2, \rho) = \frac{1}{e^{\frac{(\chi^2 - \rho)}{\rho}}}$$

Where χ^2 is the Chi-squared value returned from by the GOF test,

and ρ is the probability based on the degrees of freedom calculated, i.e. $\chi^2_{\alpha,k-1-s}$.

Figure 1 shows how the inverse exponential function would tend toward returning a higher value the more negative the result. If Chi-squared is sufficiently lower than ρ, it would yield a higher score, and if

multiple scores were sufficiently lower they would clump together at higher values. Poor GOF test values would in turn provide very low scores very quickly.

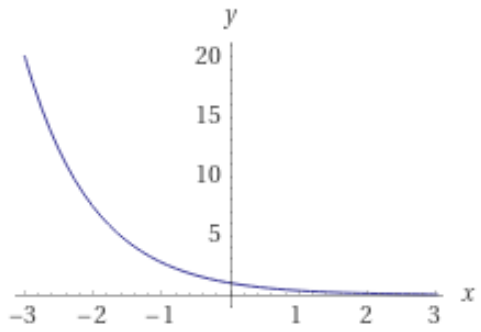


Figure 1 Example Score Distribution

The following example output shows how the Weibull, Gamma and Exponential scores for a given dataset are relatively close together as can be seen by the relative distances in GOF values (Chi-squared, ρ). Additionally it is easy to see that the Normal and Uniform distributions are far from being a good fit.

```
Distribution Type:  Weibull
Goodness of Fit:  [11.38387133 21.02606982]
Overall Score:  1.5818309649103066
-------------
Distribution Type:  Gamma
Goodness of Fit:  [11.51247593 21.02606982]
Overall Score:  1.5721853266225467
-------------
Distribution Type:  Exponential
Goodness of Fit:  [12.80054565 21.02606982]
Overall Score:  1.4787630721138922
-------------
Distribution Type:  Normal
Goodness of Fit:  [396.04277934  18.30703805]
Overall Score:  1.094069190258449e-09
-------------
Distribution Type:  Uniform
Goodness of Fit:  [1509.67708333   33.92443847]
Overall Score:  1.28139625957322e-19
```

2.2.7 *Types*

The Types module provides 2 Enum classes that define the supported distribution types in the system and the measure types (e.g. Discrete, Continuous). The class DistributionType contains the

following possible values: GEOMETRIC, UNIFORM, NORMAL, EXPONENTIAL, GAMMA, BERNOULLI, BINOMIAL, WEIBULL and UNKNOWN. The “UNKNOWN” value is a default value for the Distribution class that prevents it from performing some actions since they are not possible without a known distribution type. These types must first undergo identification, or be assigned a type other than “UNKNOWN”. MeasureType is the other class in the Types module, it consists of: DISCRETE, CONTINUOUS and UNKNOWN. Similar to the DistributionType, the “UNKNOWN” measure type indicates that this distribution has not yet been identified.

2.3 Support Infrastructure and Tooling

2.3.1 *Tooling*

Development and packaging of Simufit was performed in the Python Language using the Visual Studio Code, Sublime Text and iPython development tools. It was programmed and tested in the following operating systems for cross-platform support: Mac OS X, Windows 10 and Ubuntu Linux 18.04 LTS. The operating systems are made available through additional tools designed to allow for a secondary system: Mac Boot Camp and the Microsoft Windows Subsystem for Linux version 2 (WSL 2). The Python environments were created and maintained using Anaconda (Conda). A YML file is maintained for each operating system to ensure consistent packages are loaded for each development environment.

2.3.2 *Support Infrastructure*

The Microsoft Azure DevOps cloud platform provided by The Georgia Institute of Technology was used to manage and maintain the project throughout the development lifecycle. This system included a Git based repository for version control, project management tools, a Wiki and automated builds to support a continuous integration (CI) process. Azure DevOps provided the ability to perform code reviews and tag commits with a version after a successful build. Each CI build is performed on an Ubuntu Linux 18.04 LTS instance after a code review has been completed and the branch is merged with master.

2.4 Packaging and Distribution

The build process produces a single WHL file (simufit-<*major*>-<*minor* >-<*patch*>-<*build*>-py3-none-any.whl) compatible with all supported operating systems. This file is stored as an artifact of the versioned build. In the instance that this software is released as open source in the future, the same pipeline will be extended to include a continuous deployment (CD) component to push any generated packages to the PyPi server. For this project, the WHL file will be downloaded and submitted.

3 APPLICATION

3.1 Installation

The installation of the Simufit package is very straightforward: once the WHL file is located on the machine where it will be installed, the user must create a virtual environment using venv, Conda, or any other preferred system. Once the user activates the environment, they can type the following and press Enter: pip install <path_to_whl_folder>/ simufit-<*major*>-<*minor*>-<*patch*>-<*build*>-py3-none-any.whl. The package will install all pinned versions of the items listed in the Requirements.txt file.

3.2 Interface and Usage

This section describes the three discrete and five continuous distributions supported by Simufit, and provides some example code to demonstrate generating RVs, calculating MLE parameters, and performing a goodness of fit test to evaluate the MLE. The PMF/PDFs are provided here to make clear the notation utilized for each distribution. In addition, log-likelihood expressions for the gamma and Weibull distributions are provided for reference, as these are relatively harder to find than those for the other distributions. Finally, a few examples are given for using the Simufit GUI.

Distribution types

dt.BERNOULLI

Interpretation: experiments with two possible outcomes, e.g. (biased) coin flip, picking a particular card out of a deck

Parameter:

p: float $\in (0, 1)$

PMF:*

$$f(x) = \begin{cases} 1-p & \text{if } x = 0 \\ p & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases}$$

MLE:

$$\hat{p} = \bar{X}$$

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 $\hat{p}$ is used.

* Source: Law 5e, page 306

dt.BINOMIAL

Interpretation: n repeated Bernoulli trials, each with success probability p. The PMF describes the probability of achieving x number of successes, for particular values of n and p.

Parameters:

n: int ≥ 0
p: float $\in (0, 1)$

PMF:*

$$f(x) = \begin{cases} \binom{n}{x} p^x (1-p)^{n-x} & \text{if } x \in \{0, 1, \cdots, n\} \\ 0 & \text{otherwise} \end{cases}$$

MLE:
$\hat{p} = \bar{X}/n$, assuming n is known. We do not handle the case where both n and p 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 $\hat{p}$ is used.
- The first value returned by the GOF method is χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Law 5e, page 308

dt.GEOMETRIC

Interpretation: The PMF describes the distribution of the number of Bernoulli trials required until the first "success".

Parameter:

p: float $\in (0, 1)$

PMF:*

$$f(x) = \begin{cases} p(1-p)^{x-1} & \text{if } x \in \{1, 2, \cdots\} \\ 0 & \text{otherwise} \end{cases}$$

MLE:

$$\hat{p} = 1/\bar{X}$$

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 $\hat{p}$ is used.
- The first value returned by the GOF method is χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: https://numpy.org/doc/stable/reference/random/generated/numpy.random.geometric.html

dt.UNIFORM

Interpretation: PDF is a horizontal line between endpoints a and b, i.e. samples are equally likely to be drawn along this interval. Often used when no information is known about samples other than the minimum and maximum values.

Parameters:

a: float ∈ $(-inf, inf)$
b: float ∈ $(-inf, inf)$

PDF:*

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

MLE:

$$\hat{a} = \min(X)$$
$$\hat{b} = \max(X)$$

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 χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Law 5e pg. 286

dt.NORMAL

Applications: Often seen in nature, e.g. distribution of heights and weights

Parameters:

μ: float ∈ $(-inf, inf)$
σ^2: float ∈ $[0, inf)$

PDF:*

$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)}$ for all real numbers x

MLE:

$\hat{\mu} = \bar{X}$
$\widehat{\sigma^2} = \frac{n-1}{n} S^2$, where S^2 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 $\hat{\mu}$ and $\widehat{\sigma^2}$ are used.
- The first value returned by the GOF method is χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Law 5e pg. 292

dt.EXPONENTIAL

Applications: Interarrival times from a Poisson process, failure times, etc.

Parameters:

λ: float > 0

PDF:*

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

MLE:

$$\hat{\lambda} = 1/\bar{X}$$

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 $\hat{\lambda}$ is used.
- The first value returned by the GOF method is χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Goldsman Lecture Notes

dt.GAMMA

Applications: Service/wait time modeling

Parameters:

a: float > 0
b: float > 0

PDF:*

$$f(x) = \begin{cases} \frac{b^{-a}x^{a-1}e^{-x/b}}{\Gamma(a)} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Log-Likelihood:

$$l(x) = (a-1)\sum_{i=1}^{n} \log X_i - n\Gamma(a) - na\log b - \frac{1}{b}\sum_{i=1}^{n} X_i$$

MLE:

We use the method in https://en.wikipedia.org/wiki/Gamma_distribution#Maximum_likelihood_estimation to compute the MLE parameters $\hat{a}$ and $\hat{b}$. 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 χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Law 5e pg. 288 with modified notation

dt.WEIBULL

Applications: Time for task completion, failure times, etc.

Parameters:

a: float > 0
b: float > 0

PDF:*

$$f(x) = \begin{cases} ab^{-a}x^{a-1}e^{-(x/b)^a} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Log-Likelihood:

$$n \log a - n \log b + (a-1)\sum_{i=1}^{n} \log \frac{X_i}{b} - \sum_{i=1}^{n} \left(\frac{X_i}{b}\right)^a$$

MLE:

We use the method in Simulation & Modeling (Law 5e, pp. 290-292) to compute the MLE parameters $\hat{a}$ and $\hat{b}$. 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 χ_0^2 and the second is $\chi^2_{\alpha,k-1-s}$, where $\alpha = 0.05$, k is the number of intervals, and s 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.

* Source: Law 5e pg. 290

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.

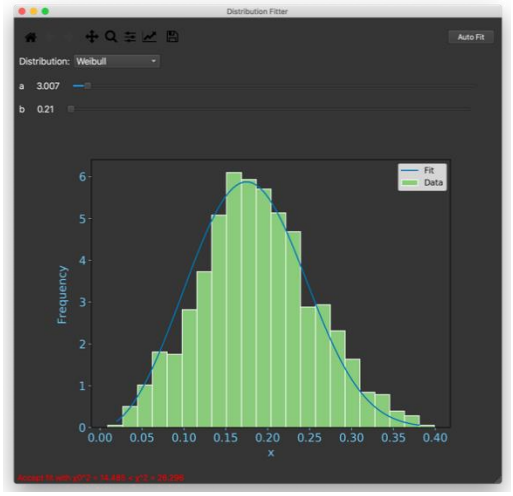


Figure 2 Example Fitter Window

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.

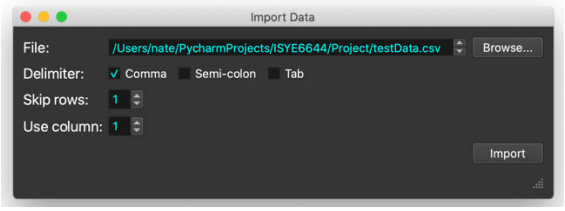


Figure 3 Example Import Data Dialog Box

You can now try to fit your data to one of the selectable distributions (Bernoulli, Binomial, Normal, etc.) either manually using the sliders, or by pressing the Auto Fit button.

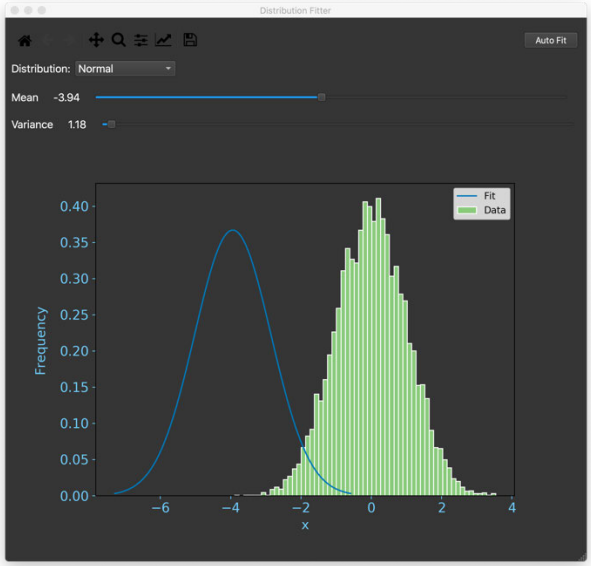


Figure 4 Example Distribution Fitting in GUI

Example 3: Fitting data (from an unknown distribution) loaded from a CSV file (alternative method)

```
import simufit as sf
from simufit import DistributionType as dt
import Display as dp

x = sf.Distribution()
dp.run_fitter(distribution=x)
```

Follow the steps in Example 2 for importing/fitting the data. When the GUI is closed, the imported data will be stored in the x object.

4 PERFORMANCE

4.1 Evaluation Approach

A Tests.py file was created to hold tests that measure the accuracy of the Simufit distribution identifications: a set of random samples is generated according to a particular distribution, and a test is run to see if it can be correctly identified. Each test is defined as a method in the “Tests” class and there

is one per distribution. Each distribution undergoes 300 executes, each with different parameters randomly generated using Numpy (Numpy, 2020). Accuracy scoring is calculated by dividing the number of correct identifications by the total number of tests performed. Simulation of the random variables (RVs) is performed by the generateSamples method in dist_generator.py. The number of observations is randomly determined for each test ranging from a minimum collection of 20 and maxing out at 400 to try and evaluate accuracy levels for a wide range of use cases. We also tested for a larger number of samples, with a minimum of 1000 samples and a maximum of 10000 samples.

4.2 Results

The test results for each distribution consisting of 300 trials each for a total of 2400 tests are shown in Table 1. As can be seen, the distributions were able to perform at an accuracy better than chance with a minimum of 0.64 (64% Accurate) and a maximum of 0.99 (99% Accurate) and an overall combined accuracy of 0.77 (77%) accuracy rate. These results correspond to a minimum of 20 samples and a maximum of 400 samples.

Distribution	***Trials***	***Accuracy***
Weibull	300	0.69
Bernoulli	300	0.99
Binomial	300	0.66
Geometric	300	0.75
Uniform	300	0.81
Normal	300	0.88
Exponential	300	0.64
Gamma	300	0.69
Overall	2400	0.76

Table 1 Performance Results for Distribution Identification; samples sizes ranging from 20 to 400.

When the number of samples is increased to between 1000 and 10000, the accuracy greatly improves, as shown in Table 2.

Distribution	*Trials*	*Accuracy*
Weibull	300	0.95
Bernoulli	300	1.00
Binomial	300	0.75
Geometric	300	0.91
Uniform	300	0.94
Normal	300	0.95
Exponential	300	0.55
Gamma	300	0.94
Overall	2400	0.86

Table 2 Performance Results for Distribution Identification; samples sizes ranging from 1000 to 10000.

Some distributions experienced lower accuracy rates than others, and this is believed to be caused by a few possible reasons. First, the testing procedure randomly determines the distribution parameters used for the RV generation; if this step produces a distribution that is highly skewed, experiencing strong kurtosis, experiences unintended scaling effects, etc… the distribution being tested has the potential to fail the test. In other cases, the software does not achieve a high enough score to identify the distribution, in these cases they are counted as an automatic failure even if the most likely distribution displayed to the user is correct. Additionally, when the number of samples is small, due to the method implemented to merge bins (see the mergeBins function in Helpers.py), it is not always possible to achieve 5 samples per bin, and a goodness of fit score cannot be calculated. This can also happen with distributions that are strongly peaked and die off rapidly. If after the merging of bins, the number of degrees of freedom is less than 1, a GOF test will not be attempted. One possible solution is to instead implement equal probability bins and ensure an appropriate minimum number of samples. Finally, correctly identifying exponentially distributed samples is challenging because a gamma distribution with a=1 and a Weibull distribution with a=1 reduce to an exponential distribution. Therefore, exponentially distributed samples can be identified as being exponential, gamma, or Weibull distributed. In a sense, our software has not actually made a

mistake here, but we currently count the identification of exponentially distributed RVs as being gamma or Weibull distributed as an error, even if the MLE parameter $\hat{a} \approx 1$.

Since the overall identification performance is much higher than chance, the team considers this both a successful project and tool with room for future improvement.

5 CONCLUSION

5.1 Summary

Overall the project was a success as it was able to accurately identify all distributions at a rate higher than chance. Additionally, the application met all of the necessary requirements to satisfy the needs to identify a distribution. While some requirements were not met, they were stretch goals and are not consequential to this project.

The application took full advantage of the available tools and resources to provide a packaged product for installation and testing. Included with the project is a README.md file which includes a set of instructions for how to the use the application. The software was designed with a sound and extendable architecture. All methods have been verified to contain sufficient comments for future maintenance and development.

5.2 Challenges

No project goes without its challenges and this one is no different. Key challenges while developing Simufit included:

- Team members were 14 hours apart (Central US & Japan)
- Multiple operating systems
- Attempting to build a WHL file
- General Learning Curve

As a team we were able to overcome these challenges. First, we engaged in communication using Slack and Zoom which provided both asynchronous and synchronous conversations at times (Slack, 2020; Zoom, 2020). To help adjust to this schedule, we learned the time difference and scheduled meetings that

coincided with one person's morning and the other's evening. Regarding multiple operating systems, we were able to find common ground by using WSL2 and Boot Camp to install and run different combinations of the supported systems. We did experience trouble when building the WHL file, this took more time than originally anticipated, but now that it is part of the continuous integration (CI) it is completely automated. Finally, it should be noted that testing and troubleshooting were more difficult than an average application due to the complexity level of the content, improvement in this area remained a work in progress throughout the project.

5.3 Future

Though this may conclude the Simufit project for ISYE-6644 - Simulation and Modeling for Engineering and Science, the practicality of this package goes beyond just the class. The goal is to first continue working on the application until all of the stretch goal requirements are met and work to improve the accuracy some. Once this has been accomplished, we will release the package to the PyPi server as an open source project for general use by the public.

6 REFERENCES

Law, A. M. (2015). Simulation modeling and analysis (5th Edition). New York: Mcgraw-Hill.

Azure DevOps Services | Microsoft Azure. (2020). Retrieved 1 December 2020, from https://azure.microsoft.com/en-us/services/devops/

Slack, Where work happens. (2020). Retrieved 1 December 2020, from https://slack.com/

Zoom Phone for Global Unified Communications | Zoom. (2020). Retrieved 1 December 2020, from https://zoom.us/docs/en-us/home-abm-a.html?utm_source=demandbase&utm_medium=redirect&utm_campaign=zp_massmarket

NumPy. (2020). Retrieved 1 December 2020, from https://numpy.org/

SciPy.org — SciPy.org. (2020). Retrieved 1 December 2020, from https://www.scipy.org/

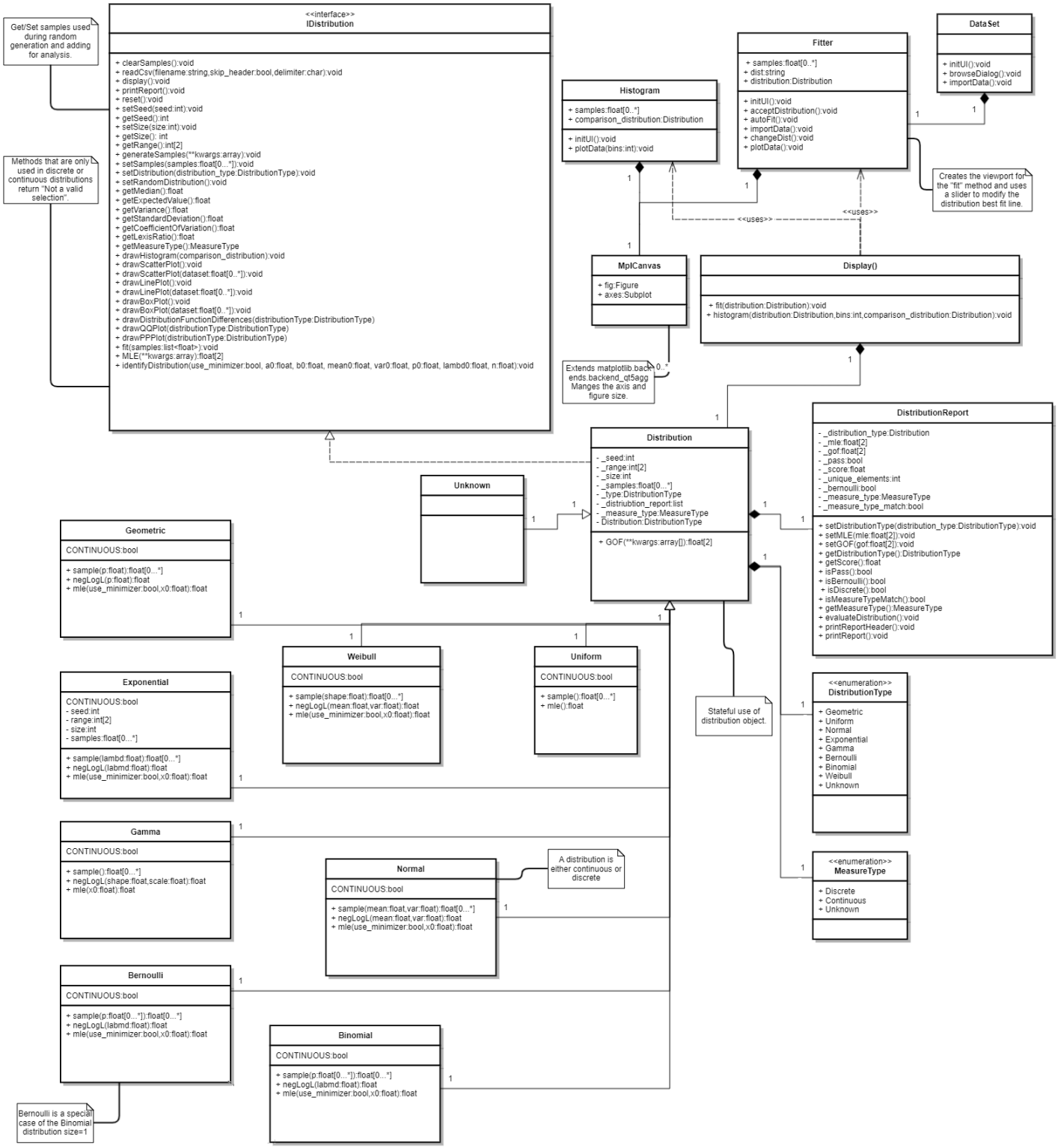
Matplotlib: Python plotting — Matplotlib. (2020). Retrieved 1 December 2020, from https://matplotlib.org/

Riverbank Computing | Introduction. (2020). Retrieved 1 December 2020, from https://www.riverbankcomputing.com/software/pyqt/

7 APPENDIX

7.1 Architecture Diagram

The software architecture diagram was designed using Gliffy (gliffy.com).



7.2 Software Requirements

The following software requirements were loaded into Azure DevOps as User Stories. All requirements that were completed are marked as "Met" below, those that were not completed are marked as "Not Met", other status are denoted as needed in the table. All requirements specific to the success of the project's primary goal (denoted by an asterisk *) were successfully met leaving only requirements for future beyond coursework development.

1	**Package**	-	
1.1	*Software shall be multi-platform using a Conda environment*	Met	*
1.2	*Software shall have a YML requirements file to set up Conda environment*	Met	*
1.3	*Software shall be packaged into a .whl file for distribution*	Met	*
1.4	*Software shall have public classes for API use*	Met	*
2	**User Interface**	-	
2.1	*A user shall interact with the software using a command line interface*	Met	*
2.2	*A user shall use menus to work engage in activities within the software*	Met	
2.3	*A user shall be able to generate a histogram of a loaded dataset*	Met	
2.4	*A user shall be able to generate a scatter plot of a loaded dataset*	Not Met	
2.5	*A user shall be able to generate a line plot of a loaded dataset*	Not Met	
2.6	*A user shall be able to generate a box plot of a loaded dataset*	Not Met	
2.7	*A user shall be able to generate a side by side histogram of a loaded dataset and a specific distribution*	Met	
~~2.8~~	~~*A user shall be able to generate a side by side histogram of a loaded dataset and a specific distribution*~~	Removed (Duplicate)	
2.9	*A user shall be able to generate a side by side scatter plot of a loaded dataset and a specific distribution*	Not Met	
2.1	*A user shall be able to generate a side by side line plot of a loaded dataset and a specific distribution*	Not Met	
2.11	*A user shall be able to generate a side by side box plot of a loaded dataset and a specific distribution*	Not Met	
2.12	*A user shall be able to generate a distribution-function-differences plot for a loaded dataset and a specific distribution*	Not Met	
2.12	*A user shall be able to generate a Q-Q plot for a loaded dataset and a specific distribution*	Not Met	
2.13	*A user shall be able to generate a P-P plot for a loaded dataset and a specific distribution*	Not Met	
3	**Import**	-	
3.1	*A user shall be able to import sample values from CSV formatted files*	Met	*
3.2	*A user shall be able to load sample values from an array*	Met	*
3.3	*A user shall have an option in the menu to load a dataset*	Met	
4	**Generate Distributions**	-	
4.1	*A user shall be able to generate values from a normal distribution*	Met	*
4.2	*A user shall be able to generate values from a uniform distribution*	Met	*
4.3	*A user shall be able to generate values from a Bernoulli distribution*	Met	*
4.4	*A user shall be able to generate values from a Binomial distribution*	Met	*

4.5	*A user shall be able to generate values from a geometric distribution*	Met	*
4.6	*A user shall be able to generate values from an exponential distribution*	Met	*
4.7	*A user shall be able to generate values from a gamma distribution*	Met	*
4.8	*A user shall be able to generate values from a Weibull distribution*	Met	*
4.9	*A user shall be able to generate a random dataset from supported distributions*	Met	
5	**Summary Statistics**	-	
5.1	*A user shall be able retrieve the range of any sample set*	Met	
5.2	*A user shall be able retrieve the median value from any sample set*	Met	
5.3	*A user shall be able retrieve the expected value from an identified distribution*	Met	
5.4	*A user shall be able retrieve the variance of any sample set*	Met	
5.5	*A user shall be able retrieve the standard deviation of any sample set*	Met	
5.6	*A user shall be able retrieve the coefficient of variation value for any continuous sample set*	Not Met	
5.7	*A user shall be able retrieve the Lexis ratio value for any discrete sample set*	Not Met	
5.8	*A user shall be able retrieve the skewness value for any sample set*	Not Met	
5.9	*A user shall be able retrieve the maximum likelihood estimation value from a sample set*	Met	*
6	**Distribution Identification**	-	
6.1	*A user shall be able submit a loaded dataset for analysis to identify and order distribution fit according to a score based on goodness of fit*	Met	*
6.2	*The software shall evaluate a loaded dataset against all supported distributions*	Met	*
6.3	*The software shall provide a report consisting of the all distributions as defined by a calculated score based on the goodness of fit value and distribution measure type (e.g. Continuous or Discrete)*	Met	*
6.4	*Each reported distribution shall provide the results of a goodness-of-fit test (if applicable)*	Met	*
6.5	*Each reported distribution shall provide the results of a Kolmogorov-Smirnov test*	Not Met	
7	**Documentation**	-	
7.1	*The software shall have a user guide that describes use via the command line interface*	Met	*
7.2	*The software shall have API documentation that describes use as an imported package*	Met	*

Table 2 Simufit Project Features and Requiements

7.3 OTS Components

The following off-the-shelf components are used by the Simufit package, versions included are pinned. Descriptions are as shown are provided by the developers of each respective package. All packages noted here are listed in the references section.

Name	Numpy
Version	1.19.2
License	OSI Approved (BSD)
Author	Travis E. Oliphant et al.
URL	https://numpy.org/
Description	NumPy is the fundamental package needed for scientific computing with Python. This package contains: • a powerful N-dimensional array object • sophisticated (broadcasting) functions • basic linear algebra functions • basic Fourier transforms • sophisticated random number capabilities • tools for integrating Fortran code • tools for integrating C/C++ code Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is a successor for two earlier scientific Python libraries: Numeric and Numarray.

Name	SciPy
Version	1.5.2
License	BSD License (BSD)
Author	SciPy Developers
URL	https://www.scipy.org/
Description	SciPy (pronounced “Sigh Pie”) is open-source software for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world’s leading scientists and engineers. If you need to manipulate numbers on a computer and display or publish the results, give SciPy a try!

Name	matplotlib
Version	3.3.2

License	Python Software Foundation License (PSF)
Author	John D. Hunter, Michael Droettboom
URL	https://matplotlib.org/
Description	Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

Name	PyQt5
Version	5.15.1
License	GPL v3
Author	Riverbank Computing Limited
URL	https://www.riverbankcomputing.com/software/pyqt/
Description	Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development. PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android. PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications.

Table 3 OTS Components Included in Simufit