Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

INPUT DISTRIBUTION MODELING USING THE KOTLIN SIMULATION LIBRARY

Manuel D. Rossetti[1], Farid Hashemain[1], Maryam Aghamohammadghasem[1],
Danh Phan[1], and Nasim Sadat Mousavi[1]

[1]Department of Industrial Engineering, University of Arkansas, Fayetteville, AR, USA

ABSTRACT

Input distribution modeling is an important aspect of stochastic simulation models. This paper provides an overview of the functionality of the discrete and continuous distribution modeling capabilities provided by the Kotlin Simulation Library (KSL). The library provides an API framework for defining distribution selection metrics, combining the metrics into an overall score, and recommending a distribution based on the scoring models. In addition, the library provides capabilities for performing goodness of fit statistical tests and other model fit diagnosis analyses. The paper provides context for how users can use existing capabilities and extend those capabilities to their needs. Examples are provided to illustrate the library.

1 INTRODUCTION

Based on various programming language rankings, Kotlin has consistently held a position within the top 15 languages over the past five years. According to Loftus (2023), Kotlin currently occupies the 11th rank and has been steadily gaining popularity. Kotlin boasts several compelling language features, including clear and concise functional and object-oriented specifications, static compilation, robust type safety, explicit handling of null values, and support for asynchronous programming via co-routines. Kotlin also offers support for Jupyter notebook computations. These attributes make Kotlin an attractive choice for teaching, research, and practice.

The Kotlin Simulation Library (KSL) is an open-source library written for Kotlin that facilitates Monte Carlo and discrete-event simulation modeling. A full description of the capabilities of the KSL is provided in Rossetti (2023a), with a summary overview provided in Rossetti (2023d). The KSL provides interfaces and classes that extend the capabilities of Kotlin to include generating pseudo-random numbers, random variate generation, probability modeling, distribution fitting, statistical analysis, and discrete-event system simulation model building via both the event view and the process view. The probability modeling and distribution fitting functionality are the focus of this paper.

Input modeling for stochastic simulation experiments remains an important topic within the application of discrete-event simulation models for both researchers and practitioners. Because of the importance of this topic, the KSL provides classes and interfaces that can be used to develop new methods for the fitting of probability models and implements functionality that is on par with commercial packages that focus on the univariate distribution fitting problem, commonly found in practice and commonly supported by discrete-event simulation packages. Because of Kotlin's support for Jupyter notebooks, the incorporation of the KSL's distribution fitting functionality is easily used within a notebook without learning all the details of the Kotlin programming language. Thus, the discussed functionality could be used instead of proprietary software within an educational setting. In addition, a Kotlin-based simulation course could be supported by Rossetti (2023a).

Because distribution fitting is typically not discussed in introductory probability and statistics courses, this leaves simulation courses to cover this important topic. The theory and methods of distribution modeling have been and continue to be an important topic in many simulation textbooks. For example, Chapter 4 of Kelton et al. (2010a), Chapter 9 of Leemis and Park (2006), Chapter 9 of Banks et al. (2005),

Chapter 4 of Kelton et al. (2010b), and Chapter 6 of Rossetti (2015) all cover the distribution input modeling task. While proprietary software is available to perform the distribution fitting task, except for a couple of exceptions noted in the literature review, there remain limited options for high-quality open-source distribution fitting software. The purpose of the KSL distribution modeling functionality is to supply an open-source and extendable framework for educators who cannot afford to or do not want to get locked into proprietary software. This paper will overview the capabilities of the KSL for distribution modeling and illustrate the application of the functionality with an example and an illustrative research question.

The paper is organized as follows. The next section presents background on selected topics in input distribution modeling based on a review of the literature. The literature review will identify a few emerging topics of interest. In addition, the section will briefly summarize other libraries or tools that users may find useful for input distribution modeling. Then, section 3 of the paper presents the KSL's distribution modeling functionality. Unfortunately, due to paper length limitations, this will necessarily only present key capabilities so that the reader can better understand the illustrative examples in the subsequent sections. Finally, the summary provides some conclusions about input modeling and suggests avenues for future research and the extension of the KSL's capabilities in this area.

2 LITERATURE REVIEW

A review of the literature indicates that the teaching of stochastic concepts, especially that surrounding the understanding of distributions remains of interest to educators. For example, Erjongmanee (2019) investigates a novel approach to teaching probability and statistics to engineering students, which offers a practical, data-driven methodology based on more than two years of empirical teaching experience (2016–2017). The method includes two steps: one, sample data collection: using a specially designed website that generates random numbers, students gather and examine sample data. This gives students a concrete way to see and comprehend distributions and randomness. Second: sample origin narration and parameter estimation: students can connect abstract numbers and actual occurrences by hearing stories about how samples are obtained from experiments. The survey findings revealed that students felt the sample analysis tasks significantly improved their comprehension of probability and their capacity to use statistical ideas.

Fergusson and Pfannkuch (2020) address the critical problem of how to informally test the fit of a probability distribution model in a way that is educationally desirable for senior secondary school students, as the current method lacks clear criteria and does not account for sample size, despite the need for a conceptually sound approach before introducing formal methods. Marasinghe et al. (1996) highlighted the limitations of traditional data analysis assignments, which often fail to provide students with a deep understanding of statistical concepts due to their static and procedural nature. Their project at Iowa State University designed instructional modules that utilize statistical software, simulations, and interactive graphics, providing a more comprehensive understanding of statistical methods. The authors propose integrating these modules into undergraduate statistics curricula to enhance student learning outcomes and facilitate understanding of complex statistical theories.

Jamie (2002) provides a comprehensive review of the application of computer simulation methods (CSMs) to enhance student understanding of statistical ideas through interactive and visual learning experiences, aligning with active learning and constructivist teaching methods. Key statistical concepts, including the Central Limit Theorem, t-distribution, Confidence Intervals, Binomial Distribution, hypothesis testing, regression analysis, sampling distributions, and survey sampling, are identified as areas where CSMs can be incredibly effective. Despite the promising recommendations from educators, Jamie (2002) identifies a significant gap in empirical research validating the effectiveness of CSMs compared to traditional teaching methods.

In the following, we present existing packages that can be used to fit distributions within a course setting. The interactive software package, UNIFIT, introduced by Law and Vincent (1985) greatly simplified the process of fitting probability distributions to observed data. With UNIFIT, statistical methods are combined with graphical representations in a three-step process: first, potential distributions are hypothesized using heuristic techniques; next, parameters for these distributions are estimated from the

data; and last, the best-fit distribution is identified using goodness-of-fit tests and heuristic techniques. In addition to heuristic techniques, UNIFIT provides analysts with access to goodness-of-fit tests such as the Anderson-Darling, Kolmogorov-Smirnov, and chi-square tests. The authors list some of the main benefits of utilizing their method, such as a notable decrease in both cost and time, a significant improvement in the amount of thoroughness that can be performed, and a reduction in the likelihood of analysis errors, tackling the problem of errors discovered in many books and software programs.

Delignette-Muller and Dutang (2015) concentrate on fitting univariate distributions to different types of data (continuous censored/non-censored, discrete) through various estimate methods, highlighting the difficulty in the statistical practice of choosing suitable distributions and estimating parameters. The *fitdistrplus* package in R enhances distribution fitting by providing various estimating methods tools for comparing fits and managing bootstrap parameter estimations. The authors present functions in the *fitdistrplus* package that facilitate distribution fitting using various methods such as maximum likelihood estimation, moment matching, and quantile matching. The package also includes tools for evaluating and comparing the accuracy of fits, but it does not automatically recommend a distribution. Possible areas for improvement include increasing the package's ability for multi-modal distributions, fitting distributions to high-dimensional data, or incorporating more complex statistical learning techniques.

In Python, there is a Reliability library (Reid 2020) that discusses the process of fitting all available distributions given a dataset. The documentation explains the automatic selection of distributions based on the BIC metric, which works with the likelihood function, number of data points, and number of parameters to be estimated. The next section introduces the distribution fitting capabilities of the KSL.

3 THE KSL DISTRIBUTION FITTING PACKAGE

Input distribution modeling involves understanding the process of generating the data, developing a plan for collecting the data, graphical and statistical analysis of the data, hypothesizing distributions, estimating the parameters of the distributions, and checking the goodness of fit for the hypothesized models. The KSL provides the modeler with tools for the graphical and statistical analysis of the data, estimating the parameters of the distributions, and checking the goodness of fit. In addition, as part of the distribution fitting framework, the KSL automates the parameter estimation process and the evaluation of a set of candidate distributions in order to provide a recommended distribution according to user-defined scoring models. This section provides an overview of this functionality.

3.1 Graphical and Statistical Analysis

To develop an understanding of the data and to generate possible candidate probability models, the key step in the input modeling process is to analyze the data graphically. The plotting capabilities of the KSL are leveraged on the *lets-plot* Kotlin open-source library. The lets-plot library is a rendition of the capabilities found within the popular *ggplot2* library that is well-known to R enthusiasts. There are implementations of lets-plot for Python and the JVM ecosystem, including Kotlin. The KSL provides pre-defined classes that wrap the capabilities of the lets-plot library to implement a set of easy-to-use plots that are most relevant to simulation practitioners. The plotting capabilities of the KSL are found in the *ksl.utilities.io.plotting* package. This section only mentions the plots relevant to input modeling. Due to space limitations, a subsequent section will illustrate a few of these plots. The KSL provides the following classes that facilitate this analysis.

- *Histogram, HistogramPlot* – The KSL *Histogram* class allows the definition of interval breakpoints and tabulates the frequencies and proportions associated with the defined bins. Summary sample statistics are also tabulated including but not limited to (average, variance, standard deviation, coefficient of variation, skewness, kurtosis, minimum, and maximum). The Histogram class provides automated procedures for determining suitable breakpoints for both the visualization task

and the goodness of fit task. The *HistogramPlot* class provides the visual representation of the histogram data within a browser window or for export to a file.

- *IntegerFrequency*, *IntegerFrequencyPlot* – These two classes parallel the capabilities of the histogram classes and are intended for use when analyzing situations involving discrete random variables.
- *Statistic* – The *Statistic* class tabulates summary statistics and provides functions for computing the order statistics, quantiles, lag correlations, central moments, Von Neumann lag-1 test statistic, AIC criterion, Anderson Darling test statistic, BIC criterion, Cramer Von Mises test statistic, empirical CDF, g-test statistic, and Watson test statistic.
- *ObservationPlot* – The observation plot makes a time-sequenced plot of observations for diagnosing time dependence and patterns.
- *ACFPlot* – The autocorrelation plot visually displays the estimated autocorrelation function for a set of lags. This is used to diagnosis whether the data can be considered as independent.
- *PPPlot*, *QQPlot* – These two plots provide the probability-probability and the quantile-quantile plots for comparing the empirical probabilities and quantiles to the hypothesized values.

The KSL also provides a PMF comparison plot, a CDF difference plot (as described in Chapter 6 of Law (2007), a density overlay of the histogram, and an empirical versus theoretical CDF plot. The most useful for input modelers is the *FitDistPlot* class, which combines the P-P plot, Q-Q plot, density plot, and empirical CDF plot into a single plot similar to the functionality found in R's *fitdistrplus* package (see Delignette-Muller and Dutang, 2015). After statistically and graphically analyzing the data, the next step in the input modeling process is to hypothesize candidate distributions. Once the candidate distributions are specified, their parameters need to be estimated.

3.2 Estimation of Distribution Parameters

The KSL provides a framework of classes and interfaces to implement procedures for estimating the parameters of distributions. Rossetti (2023d) describes the distributions available within the KSL each of which have corresponding implementations for estimating its parameters. The following estimation methods are available: generalized beta (MOM), binomial (MOM), exponential (MLE), gamma (MLE or MOM), lognormal (MLE), negative binomial (MOM), normal (MLE), Pearson Type 5 (MLE), Poisson (MLE), triangular, uniform, and Weibull (MLE and percentile).

The most important concept for using the API is that a user must provide the *estimateParameters()* function of the *ParameterEstimatorIfc* interface and that it returns an instance of the *EstimationResult* class. These interfaces define a flexible approach to the estimation process: an input array of data and an output array of parameter estimates. The results of the estimation process are summarized in the *EstimationResult* class, which provides the estimated parameters, statistics, data for testing, and most importantly an indication of the success or failure of the estimation process. In essence, this permits the application of any parameter estimation algorithm on a given dataset regardless of whether the distribution is appropriate or regardless of the convergence of the algorithm. If the algorithm reports success, then the estimated parameters are included in the evaluation of the distribution.

If modelers need to fit a distribution that is not modeled within the KSL, then the distribution's parameter estimation procedure must be implemented. In addition, the modeler will need to implement a class representing the distribution. Depending on the type of distribution, this may involve implementing the requirements of the *ContinuousDistributionIfc* interface or the *DiscreteDistributionIfc* interface for the underlying probability model. The KSL also automatically provides the ability to compute confidence intervals on the estimated parameters. This is achieved through the KSL's extensive bootstrapping capabilities provided within the *ksl.utilities.statistic* package.

To setup the KSL's distribution recommendation functionality, the user supplies a set of objects that implement the *ParameterEstimatorIfc* interface. This set of estimators constitute the set of possible distributions evaluated for quality of fit. The default set includes the uniform, triangular, normal,

generalized beta, exponential, lognormal, gamma, Weibull, and Pearson type 5 distributions. The following section describes the recommendation methodology.

3.3 Distribution Recommendation Functionality

Once the parameters have been estimated, the quality of the distributional fit can be assessed, and a distribution recommended. Via the KSL's *PDFModeler* class, the parameters of candidate distributions can be automatically estimated and scored across a number of pre-defined or user developed metrics. The metric must define its domain (or set of legal values) and its direction. Direction means whether bigger scores are better than smaller scores or vice versa. This information is used when a set of scores are combined into an overall score. To combine the scores, the KSL uses a form of multi-objective decision analysis (MODA). The basics of the MODA methodology are fairly standardized as laid out in (Parnell, et al. 2013). The following scoring models are illustrated in this paper.

- Squared error criterion - The squared error is defined as the sum over the intervals of the squared difference between the relative frequency and the theoretical probability for each interval.
- Chi-Squared criterion - The chi-squared criterion is the chi-squared test statistic value that compares the observed counts to the expected counts over the intervals.
- Cramer Von Mises criterion - The Cramer Von Mises criterion is a distance measure used to compare the goodness of fit of a cumulative distribution function to the empirical distribution.
- Anderson Darling criterion - The Anderson-Darling criterion is similar in spirit to the Cramer Von Mises test statistic except that it is designed to detect discrepancies in the tails of the distribution.
- Kolmogorov-Smirnov criterion - The Kolmogorov-Smirnov criterion represents the largest vertical distance between the hypothesized distribution and the empirical distribution over the range of the distribution.

As part of the MODA methodology, the KSL recommendation framework permits users to define their own scores, specify the value measure mapping functions (linear by default), specify the weights of importance amongst the metrics (equal by default), and the form of the value aggregation model (additive by default). This method permits an overall score to be computed, and the distribution with the highest score is recommended.

One of the key challenges of automating the scoring process is that some criteria may be sensitive to the specification of histogram breakpoints. The squared error and chi-squared criteria are two such metrics. The KSL attempts to define the breakpoints for the intervals such that each interval has the same probability of occurrence. This also ensures that the expected number of observations within each interval is approximately the same. This approach is outlined in Chapter 6 of the Law (2007). The main issue is determining the number of intervals. Williams (1950) recommended choosing the class limits for testing U (0,1) data such that the expected number in the interval was n/k, where n is the number of observations and k is the number of class intervals. With $z_{1-\alpha}$ representing the standard normal ordinate for confidence level $1-\alpha$, Williams (1950) recommended that the number of class intervals be:

$$k = \left\lfloor 4\sqrt[5]{\frac{2(n-2)^2}{z_{1-\alpha}}} \right\rfloor$$

This approach produces equally distant breakpoints over (0,1). Define the breakpoints as u_i for $i = 1,2,\cdots,k$. Then, the approach defines breakpoints for the distribution, F(x), as $b_i = F^{-1}(u_i)$, which results in non-equally spaced breakpoints for F(x), but the probability associated with each interval will be approximately the same (equal). As recommended in Law (2007), this approach mitigates the sensitivity

of the metrics to the specification of the breakpoints and has theoretical underpinnings that attempt to have sufficient observations within each interval.

While the KSL's recommendation framework is based on scoring models with no distribution testing, the framework also facilitates the statistical tests that may be associated with some of the metrics. Thus, statistical tests of the recommended distribution can still easily be performed, including the chi-squared test, the Kolmogorov-Smirnov test, the Anderson-Darling test, and the Cramer-Von Mises test.

The KSL focuses on the fitting and evaluation of continuous distributions; however, the *PMFModeler* class provides the ability to estimate the parameters of discrete distributions and perform statistical tests on the fitted distribution. Due to space limitations, we do not discuss these capabilities in this paper. The design of the *PDFModeler* class and the flexibility of the estimation framework permit the modeler to step through a series of commands to produce pieces of the analysis or to call a single command and fit all the default distributions, evaluate the scores, recommend a top distribution, then perform statistical analysis. This will be illustrated in the next section.

4 EXAMPLE USAGE OF THE LIBRARY

This section presents illustrative examples of using the library for both distribution fitting and investigating interesting research questions. The first example illustrates how to use the library to recommend a distribution and its parameters to a data set.

4.1 Illustrative Distribution Modeling

The data associated with this example can be found here. This data represents the time to perform a task within a pharmacy setting measured in seconds rounded to two decimal places. To analyze this situation, we will work through a KSL Kotlin script. The first step is to import the data, perform some statistical analysis, and make some plots. These tasks can be performed with the following code.

```
val myFile = KSLFileUtil.chooseFile()
if (myFile != null){
    val data = KSLFileUtil.scanToArray(myFile.toPath())
    val d = PDFModeler(data)
    println(d.histogram)
    println()
    val hPlot = d.histogram.histogramPlot()
    hPlot.showInBrowser()
    val op = ObservationsPlot(data)
    op.showInBrowser()
    val acf = ACFPlot(data)
    acf.showInBrowser()
```

This code opens a file chooser dialog to allow the user to select the file. The file is a text file with the data in a single column with an observation on each line of the file. The data is scanned into an array and supplied to create the instance of the *PDFModeler* class. The *PDFModeler* class uses a KSL histogram to summarize the statistics of the data and prepare a histogram plot. In addition, the KSL's *ObservationsPlot* and *ACFPlot* class are used to show the observations ordered by time and an autocorrelation plot. Due to space limitations, the plots are not shown here. The histogram output is informative in that it clearly shows that the task time has a minimum that is much larger than zero. Thus, estimating a shift parameter will be necessary for this situation.

```
Histogram:
Bin Range        Count CumTot Frac  CumFrac
```

```
  1 [36.00,161.00) = 60 60.0 0.600000 0.600000
  2 [161.00,286.00) = 22 82.0 0.220000 0.820000
  3 [286.00,411.00) = 10 92.0 0.100000 0.920000
  4 [411.00,536.00) = 6 98.0 0.060000 0.980000
  5 [536.00,661.00) = 1 99.0 0.010000 0.990000
  6 [661.00,786.00) = 1 100.0 0.010000 1.000000
Number 100.0
Average 182.77779999999996
Standard Deviation 141.61119713746893
Minimum 36.84    Maximum 782.22
```

The KSL *PDFModeler* class can automatically test for and prepare the data for distribution fitting when a shift parameter is warranted. The following code illustrates the fitting and evaluation portion of the KSL script.

```
val results  = d.estimateAndEvaluateScores()
println("PDF Estimation Results for each Distribution:")
results.sortedScoringResults.forEach(::println)
val topResult = results.sortedScoringResults.first()
val scores = results.evaluationModel.alternativeScoresAsDataFrame("Distributions")
val values = results.evaluationModel.alternativeValuesAsDataFrame("Distributions")
val distPlot = topResult.distributionFitPlot()
distPlot.showInBrowser("Recommended Distribution ${topResult.name}")
println("** Recommended Distribution** ${topResult.name}")
val gof = ContinuousCDFGoodnessOfFit(topResult.estimationResult.testData,
    topResult.distribution,
    numEstimatedParameters = topResult.numberOfParameters)
println(gof)
```

The instance of the *PDFModeler* class is used to estimate and evaluate the fit via the function (*estimateAndEvaluateScores*()). From the results, the user can access the estimated parameters and results for every distribution that was evaluated. In addition, the scoring model results allow the user to review the scores and find the recommended result. Due to space limitations, only a portion of the output is presented in Figure 1, which illustrates the scoring results and distribution fits.

Distributions	Chi-Squared	K-S	Squared-Error	Anderson-Darling	Cramer-von-Mises	Overall Value
36.83628655364795 + Exponential(mean=...	0.962437	0.927242	0.962437	0.996240	0.997991	0.969269
36.83628655364795 + Weibull(shape=1.0...	0.958681	0.912628	0.958681	0.995842	0.997172	0.964601
36.83628655364795 + Gamma(shape=0.983...	0.942716	0.921876	0.942716	0.995954	0.997854	0.960223
GeneralizedBeta(min=29.31090909090909...	0.927691	0.795922	0.927691	0.972583	0.982603	0.921298
36.83628655364795 + Lognormal(mean=25...	0.935204	0.797447	0.935204	0.959276	0.970068	0.919440
Normal(mean=182.77779999999996, varia...	0.799037	0.696777	0.799037	0.926534	0.932761	0.830829
Triangular(minimum=29.310909090909092...	0.866651	0.444919	0.866651	0.721535	0.699584	0.719868
Uniform(minimum=29.310909090909092, m...	0.644090	0.103027	0.644090	0.110693	0.110888	0.322557
36.83628655364795 + PearsonType5(shap...	0.106937	0.150190	0.106937	0.528995	0.473141	0.273240

Figure 1: Sorted standardized MODA value scores for each distribution.

The scoring results show that the recommended distribution is 36.84 + Exponential(mean=145.94). The KSL also facilitates statistical testing of the results as shown in the following output.

```
Number of estimated parameters = 1
Number of intervals = 20
Degrees of Freedom = 18
Chi-Squared Test Statistic = 16.00078444607034
P-value = 0.5924925929222205
Hypothesis test at 0.05 level:
The p-value = 0.5924925929222205 is >= 0.05: Do not reject hypothesis.

Goodness of Fit Test Results:
K-S test statistic = 0.0399213095618332
K-S test p-value = 0.9954253138552503
Anderson-Darling test statistic = 0.24859556182926212
Anderson-Darling test p-value = 0.9710833730000956
Cramer-Von-Mises test statistic = 0.02528354392365822
Cramer-Von-Mises test p-value = 0.9893094979248238
```

Based on the goodness of fit tests and reported P-Values, we would not reject the hypothesis that the data comes from the recommended exponential distribution. Figure 2 presents the *FitDistPlot* plot containing the density overlay, empirical CDF, Q-Q, and P-P plots and clearly indicates a good fit to the data.

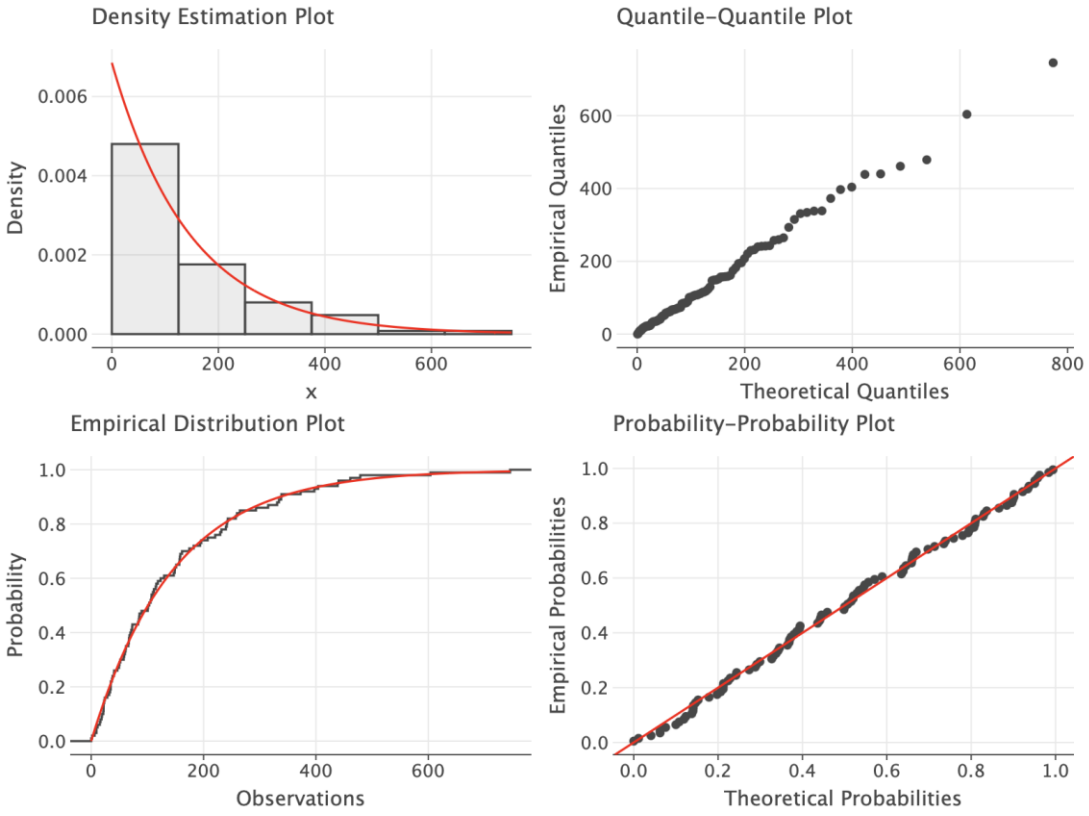

Figure 2: Histogram, Q-Q, ECDF, and P-P FitDistPlot charts.

4.2 How Much Data Do You Need to Fit a Gamma Distribution?

This section illustrates how the KSL's distribution fitting API can be utilized to study research questions of interest. The results presented in this section were generated and analyzed within the context of a unit on distribution modeling within a graduate level course on simulation. The API of the library allows the flexibility of executing experiments involving distribution fitting and evaluation. The API facilitates an analysis of the quality of different estimation algorithms, the study of metrics and their concordance with selecting the correct distribution, the tuning of evaluation/recommendation models, and understanding the effects of sample size on the recommendation process. In this section, we attempt to answer the question of, how much data is needed to adequately fit a gamma distribution. Our effort in this section presents preliminary work towards addressing the same question for any distribution.

The experimental setup is as follows. For a set of test cases involving the gamma distribution, we will generate datasets for sample sizes $n = 10, 20, \cdots, 2000$. This process will be repeated for multiple replications ($r = 1000$) to measure the chance that the correct distribution family (in this case the gamma distribution) is selected by the scoring and evaluation process.

Since Kotlin is a general-purpose programming language users have access to a wide variety of libraries and solutions. Using the IO utilities, statistics and random packages of the KSL, the experiments were set up to capture a wide variety of statistical responses including the (1^{st} = win, 2^{nd} = place, 3^{rd} = show) place winners for each sample size across the 1000 replications. From this, we can estimate the chance that the gamma distribution is correctly recommended. We also capture the estimated parameters (shape and scale) so that error measures can be computed with respect to the true known parameters. The set of candidate distributions included are exponential, Weibull, gamma, generalized beta, lognormal, normal, triangular, uniform, and Pearson Type V. The scoring criteria presented in section 3.3 were used in the evaluation process and with default equal weights across the scoring models were used in the MODA evaluation. In other words, the default setting of the KSL *PDFModeler* class was used. However, since the data was known to be generated without a shift parameter, the automated shift parameter estimation procedures were turned off for the experiments.

The results of Figure 3 concern the accuracy of the estimation fitting process. As anticipated, the results indicate that as the sample size grows, the probability of the shape or scale parameter being within 10% of the true value also increases.

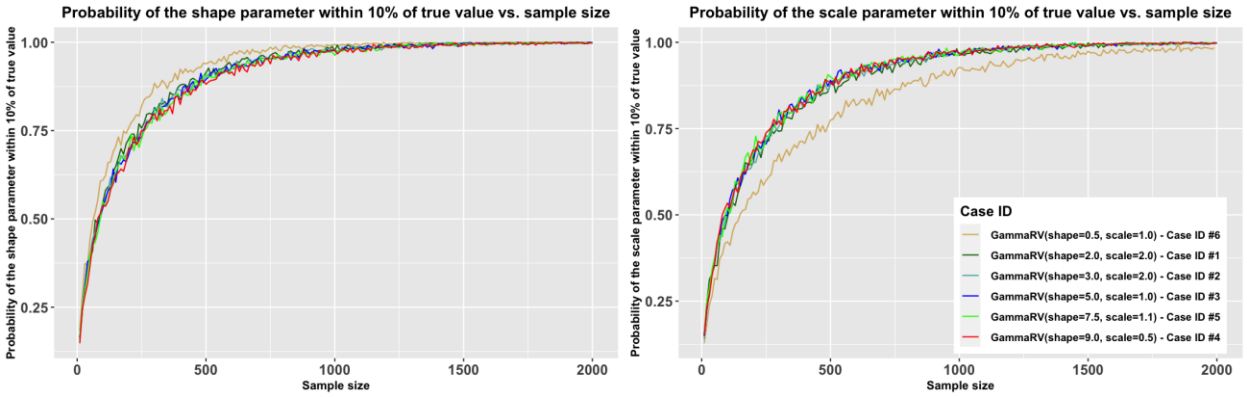

Figure 3: Probability of the shape and scale being within 10% of true value over different sample sizes.

Table 1 shows the minimum sample size needed to identify the distribution family with a predetermined probability for each case. The sample sizes in the table represent the initial sample size where the probability of correctly selecting the distribution family is associated. Based on Table 1, scenario 6 can accurately identify the Gamma distribution with a probability exceeding 0.975 by using just 100 samples. However, for cases 4 and 5, it takes a sample size of over 2000 to meet this probability.

Table 1: Gamma distribution cases and number of samples needed to identify the distribution family.

							P(Correct Selection)					
ID	**shape**	**scale**	**mean**	**variance**	**cv**	**0.5**	**0.75**	**0.8**	**0.85**	**0.9**	**0.95**	**0.975**
1	2	2	4.0	8.0	2.0	50	130	160	240	310	580	840
2	3	2	6.0	12.0	2.0	70	190	290	350	530	800	1170
3	5	1	5.0	5.0	1.0	100	290	410	520	720	1170	1690
4	9	0.5	4.5	2.25	0.5	160	450	610	800	1120	1770	>2000
5	7.5	1.1	8.25	9.075	1.1	188	440	500	700	1000	1630	>2000
6	0.5	1	0.50	0.500	1.0	20	30	40	40	50	80	100

Noting that the column (0.5) in Table 1 represents the sample size where the correct distribution family (gamma) became the majority of the recommendations, we get insights into the minimum amount of data necessary to find the distribution family. Clearly, as shape gets smaller, the sample size needed decreases. The reason that the shape of 0.5 does so well is that the resulting distribution is very different in shape (form) than many common distributions. Also, as the shape increases, the sample size needed tends to increase. Looking at the actual distributional forms indicates that more data is needed because the cases look very similar to other distribution shapes (e.g., normal etc.). The recommendation method needs more data when the distributions look similar. It should be clear from these initial results that the amount of data needed depends on the distributional form, and, in general, has a magnitude in the hundreds.

Figure 4 visually presents the relationship between the sample size and the probability of choosing the correct distribution family for each case. Based on Figure 4, some cases can detect the distribution family faster than others, such as case 6 (brown line in the figure). The probability of detecting the family of distribution is significantly influenced by the shape and scale parameter of the Gamma distribution.

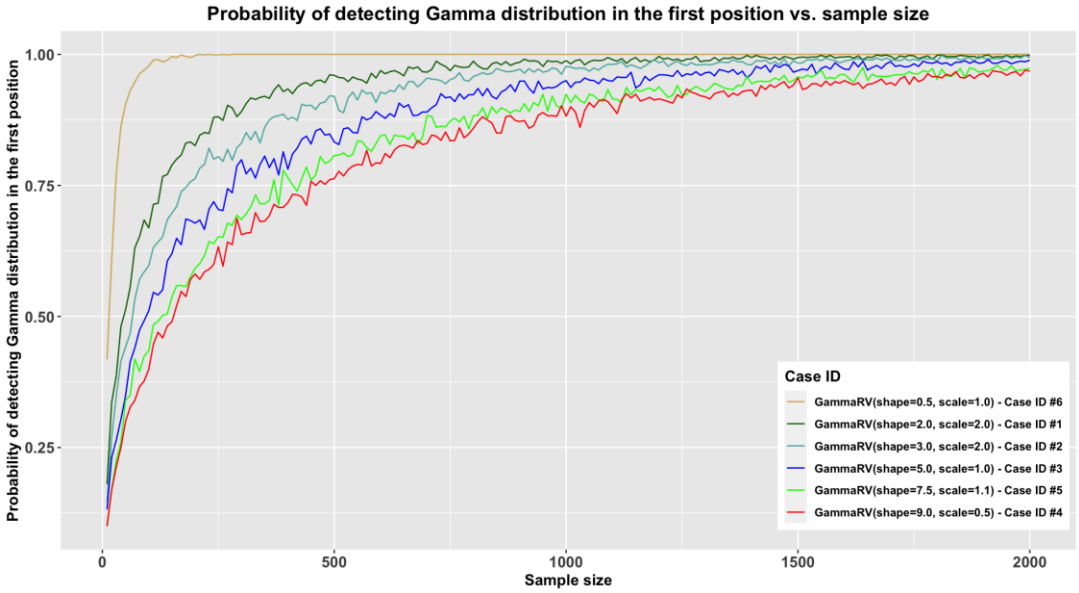

Figure 4: Probability of detecting Gamma distribution for each case over different sample sizes.

5 SUMMARY AND FUTURE WORK

This paper introduced a new distribution and fitting API for the Kotlin Simulation Library (KSL). The API consists of packages (classes and interfaces) designed to facilitate the estimation of distribution parameters and the evaluation of the resulting distributions for their quality of fit. The key innovative features include

a standardized API for the estimation process, which allows any estimation routine to be used for distribution parameter estimation. This also includes the bootstrap estimation of confidence intervals for the estimated parameters. The KSL provides a wide variety of diagnostic plots for analyzing the data before the fitting process (e.g., time series plot, ACF, histograms, frequency plots, etc.) as well as diagnostic plots such as density overlay, empirical CDF, Q-Q, and P-P plots. The KSL also provides a multi-objective distribution scoring and recommendation engine based on the inclusion of sets of user defined distribution fitting metrics. The API will also estimate shift parameters and perform goodness of fit statistical tests.

The main area for future work includes the implementation of additional distributions such as Pearson Type IV, Laplace, LogLogistic, Logistic, Johnson, and MetaLog (Keelin, 2016) and implementations for their parameter estimation. In addition, work is in progress to investigate and tune the distribution recommendation engine and develop recommendations for sample size requirements for various distributions. In the case of tuning the recommendation engine, a method for computing basic statistics for the sample and then setting the weights for the scoring may allow for more accurate recommendations. For example, if you suspect that the data comes from a particular distributional family, how much data should you have to reliably recommend the distribution.

REFERENCES

Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. 2005. *Discrete-Event System Simulation*. 4th ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Delignette-Muller, M. L., & Dutang, C. 2015. "fitdistrplus: An R Package for Fitting Distributions". *Journal of Statistical Software* 64(4):1–34.

Erjongmanee, S. 2019. "Alternative Approach to Teach Probability and Statistics for College Engineering Students". In *2019 IEEE Global Engineering Education Conference (EDUCON)*, April 8th-11th, Dubai, UAE, 931-936.

Fergusson, A., Pfannkuch, M. 2020. "Development of an Informal Test for the Fit of a Probability Distribution Model for Teaching", *Journal of Statistics Education* 28(3):344-357.

Jamie, D.M., 2002. "Using Computer Simulation Methods to Teach Statistics: A Review of the Literature". *Journal of Statistics Education* 10(1).

Keelin, T. W., 2016. "The Metalog Distributions". *Decision Analysis* 13(4):243-277.

Kelton, W. D., Sadowski, R. P., and Swets, N. B. 2009. *Simulation with Arena*. 5th ed. McGraw-Hill.

Kelton, W. D., Smith, J. S., Sturrock, D. T., Verbraeek, A. 2010. *Simio & Simulation Modeling, Analysis, Applications*. Boston, Massachusetts: McGraw-Hill.

Law, A. 2007. *Simulation Modeling and Analysis*. 4th ed. McGraw-Hill.

Law, A.M. and Vincent, S.G., 1985. "UNIFIT: An Interactive Computer Package for Fitting Probability Distributions to Observed Data". In *1985 Winter Simulation Conference (WSC)*, 59 https://doi.org/10.1145/21850.253074.

Leemis, L. M. and Park, S. K. 2006. *Discrete-Event Simulation: A First Course*. 1st ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Loftus, R. 2022. "The 15 Most Popular Programming Languages of 2023". *Hacker Rank*, accessed 5th April 2023, https://www.hackerrank.com/blog/most-popular-languages-2023/.

Marasinghe, M. G., Meeker, W. Q., Cook, D., Shin, T. S. 1996. "Using Graphics and Simulation to Teach Statistical Concepts". *The American Statistician* 50(4), 342-351.

Parnell, G., Bresnick, T., Tani, S., and Johnson, E. 2013. *Handbook of Decision Analysis*. 1st ed. Hoboken, New Jersey: Wiley.

Reid, M. 2020. Reliability - A Python library for reliability engineering. https://reliability.readthedocs.io/en/latest/, accessed 19th June 2024.

Rossetti, M. D. 2015. *Simulation Modeling and Aren*. 2nd ed. Hoboken, New Jersey: John Wiley & Sons.

Rossetti, M. D. 2023. Simulation Modeling using the Kotlin Simulation Library (KSL), Open Text Edition. Retrieved from https://rossetti.github.io/KSLBook/ licensed under the Creative Commons Attribution-Noncommercial-No Derivatives 4.0 International License.

Rossetti, M. D. 2023. "KSL Open-Source Repository". https://rossetti.github.io/KSLDocs/-k-s-l-core/ksl.utilities/index.html, Accessed 5th April 2024.

Rossetti, M. D. 2023. "KSL API Documentation". https://rossetti.github.io/KSLDocs/index.html, accessed 5th April 2024.

Rossetti, M. D. "Introducing the Kotlin Simulation Library (KSL)". In *2023 Winter Simulation Conference (WSC)*, 3311-3322 https://dl.acm.org/doi/10.5555/3643142.3643418.

Williams, C. A. 1950. "The Choice of the Number and Width of Classes for the Chi-Square Test of Goodness of Fit". *Journal of the American Statistical Association* 45 (249):77–86.

AUTHOR BIOGRAPHIES

MANUEL D. ROSSETTI is a University Professor of Industrial Engineering and the Director of the Data Science Program at the University of Arkansas. He received his Ph.D. in Industrial and Systems Engineering from The Ohio State University. Dr. Rossetti has published two textbooks and over 125 refereed journal and conference articles in the areas of simulation, logistics/inventory, and healthcare and has been the PI or Co-PI on funded research projects totaling over 6.5 million dollars. In 2013, Rossetti received the Charles and Nadine Baum Teaching Award, the highest teaching honor bestowed at the UofA, and was elected to the UofA's Teaching Academy. Dr. Rossetti was elected as a Fellow for IISE (Industrial and Systems Engineers) in 2012, in 2021 received the IISE Innovation in Education competition award, and in 2023 he received the Albert G. Holzman Distinguished Educator Award. He serves as an Associate Editor for the International Journal of Modeling and Simulation and is active in IISE, INFORMS, and ASEE. He served as co-editor for the WSC 2004 and 2009 conference, the Publicity Chair for the WSC 2013 Conference, 2015 WSC Program Chair, and is the General Chair for the WSC 2024. He can be contacted at rossetti@uark.edu and https://rossetti.uark.edu/.

FARID HASHEMIAN received his bachelor's and master's degree in Industrial Engineering in 2018 and 2020. Currently, he is a Ph.D. student in Industrial Engineering at the University of Arkansas. He is passionate about data-driven decision-making and real-world problem-solving using Machine Learning, Data Analysis, and Optimization. He can be contacted at sfhashem@uark.edu and https://tinyurl.com/2kuy5sr8.

MARYAM AGHAMOHAMMADGHASEM is a Ph.D. student in the Department of Industrial Engineering at the University of Arkansas. She earned her M.S. in Industrial Engineering from Sharif University of Technology in 2016. Her primary research focus includes Deep Learning, Deep Reinforcement Learning, and Simulation methodologies. She can be reached via email at maghamoh@uark.edu.

NGUYEN D. PHAN is a Ph.D. student in Industrial Engineering at the University of Arkansas. He obtained his M.S. in Operations Management in 2019 and M.S. in Industrial Engineering degree in 2022. His research interests include using advanced mathematical and optimization techniques to improve decision-making. Before his graduate studies at the University of Arkansas, he was working in an industry with high competition and homogeneous products. He can be contacted at ndphan@uark.edu.

NASIM SADAT MOUSAVI is a Ph.D. student in the Department of Industrial Engineering at the University of Arkansas. She earned her bachelor's and master's degrees in industrial engineering and supply chain and Operation Management, respectively, from Sharif University of Technology in 2015 and 2018. Her primary research focus includes using Deep Learning, feature engineering, and optimization techniques to improve decision-making in the healthcare sector. She can be reached via email at nmousavi@uark.edu.