Assignment No. 3

Abhijit Somnath Shendage
ID:123103499

March 16, 2024

MSc Computing Science
School of Computer Science & Information Technology
University College, Cork

Contents

1 Overview

The tools developed encompass a framework for generating and visualizing data sequences based on randomly generated distributions. These utilities make it easier to create datasets based on the specifications given in configuration files. After these configurations are parsed, the generator module creates sequences that follow the given distributions. These sequences are subsequently written to CSV files for further analysis and are optionally plotted using the plotter module. The developed tool encompasses three main components: the Driver, Generator, and Plotter.

1.1 Driver

1. It walks through the current directory and finds all the files whose extension is `.cfg`
2. Each file in this list passes the name to the generator function to parse and get the data in a usable format, i.e., a dictionary.
3. Again, it passes this parsed data to the generator function to generate various sequences or distributions, as mentioned in the configuration file.
4. If everything before went right, it retrieves the configuration file name without extension and adds CSV as an extension.
5. Then, it passes this filename along with generated data to the generator to write this data into the CSV file as per distributions.
6. Lastly, if the optional argument `--plot` is passed in the command line, the plotter is used to plot various graphs and write these into the CSV files. To do this, it makes use of the `argparse`.

1.2 Generator

The generator module serves as the engine that drives the generation of sequences or distributions based on the parsed configuration data. Its functionalities include:

1. It gets the filename from the driver, then looks for this file in the current directory and reads that file to parse that data.
2. A `configparser` is used to parse the data, and all the data is converted into a dictionary, which can be easily used afterward. To do this, it iterates over every section of the file and gets separate data for each distribution.
3. It also fixes the datatypes, like, while reading the file, everything is, by default, a string. Hence, it converts that data as keys to strings, like distribution type or parameter names, and the values to int or float, depending on how they are supposed to be. It is a manual function named

`convert_to_float` in the code. It also makes everything lowercase so that the requirement of case insensitivity can be fulfilled.

4. Then, it seeds the random function seed value parsed from the configuration file.

5. After this, it iterates over all the items, that is, distributions like gamma, exponential, daggum, gaussian, geometric, poission, skellam in the configuration file, to generate the sequences. For dagum and skellam, it uses the custom functions named `distribution_dagum` and `distribution_skellam`, and for all other functions, it uses `NumPy` library functions.

   Below are the descriptions and formulas for the sampling functions for the Dagum and Skellam distributions:

1.2.1 Dagum Distribution

The probability density function (PDF) of the Dagum distribution is given by:

$$f(x;\alpha,\beta,p) = \frac{\alpha}{\beta}\left(1-\left(1-u^{-1/\alpha}\right)^{1/p}\right)$$

Where:

- α: shape parameter
- β: scale parameter
- p: shape parameter
- u: random variable sampled from a uniform distribution on the interval $[0,1]$

The function returns an array of random samples from the Skellam distribution.

1.2.2 Skellam Distribution

The Skellam distribution represents the difference of two independent Poisson-distributed random variables X_1 and X_2. The probability mass function (PMF) of the Skellam distribution is given by:

$$f(x;\mu_1,\mu_2) = e^{-(\mu_1+\mu_2)}\left(\frac{\mu_1}{\mu_2}\right)^{\frac{x}{2}} I_x\left(2\sqrt{\mu_1\mu_2}\right)$$

Where:

- μ_1: mean parameter of the first Poisson distribution

- μ_2: mean parameter of the second Poisson distribution
- I_x is the modified Bessel function of the first kind

The function returns the difference between random samples drawn from two Poisson distributions with means μ_1 and μ_2.

6. It then writes this generated data into the CSV file using the built-in CSV module and adds a row.
7. Meaning, for each configuration file, there will be a CSV file. Every distribution in the configuration file will have a row in that CSV file with the generated sequence.

1.3 Plotter

Completing the tool's functionality, the Plotter module facilitates the visual representation and analysis of the generated distributions through graphical plots. Its functionalities include:

1. It takes the data and filename from the driver function.
2. It plots two graphs for each distribution, the CDF and boxplot, using the subplot function.
3. It then writes this in the pdf file, which has names like the configuration file name, distribution name, and the CDF-box plot type in the end.

2 Technologies Used

The developed tool has been implemented using the following technologies:

- Python 3.7.4
- Matplotlib 3.1.2
- NumPy 1.17.4

3 Output: Generated Sequences and Plots

The tool has been developed to generate the following sequences:

3.1 Continuous Distributions

- Gamma
- Exponential
- Gaussian
- Dagum

3.2 Discrete Distributions

- Geometric
- Poisson
- Skellam

For each of these sequences, the tool generates both boxplots and Cumulative Distribution Function (CDF) plots. Below are the screenshots of the generated plots:

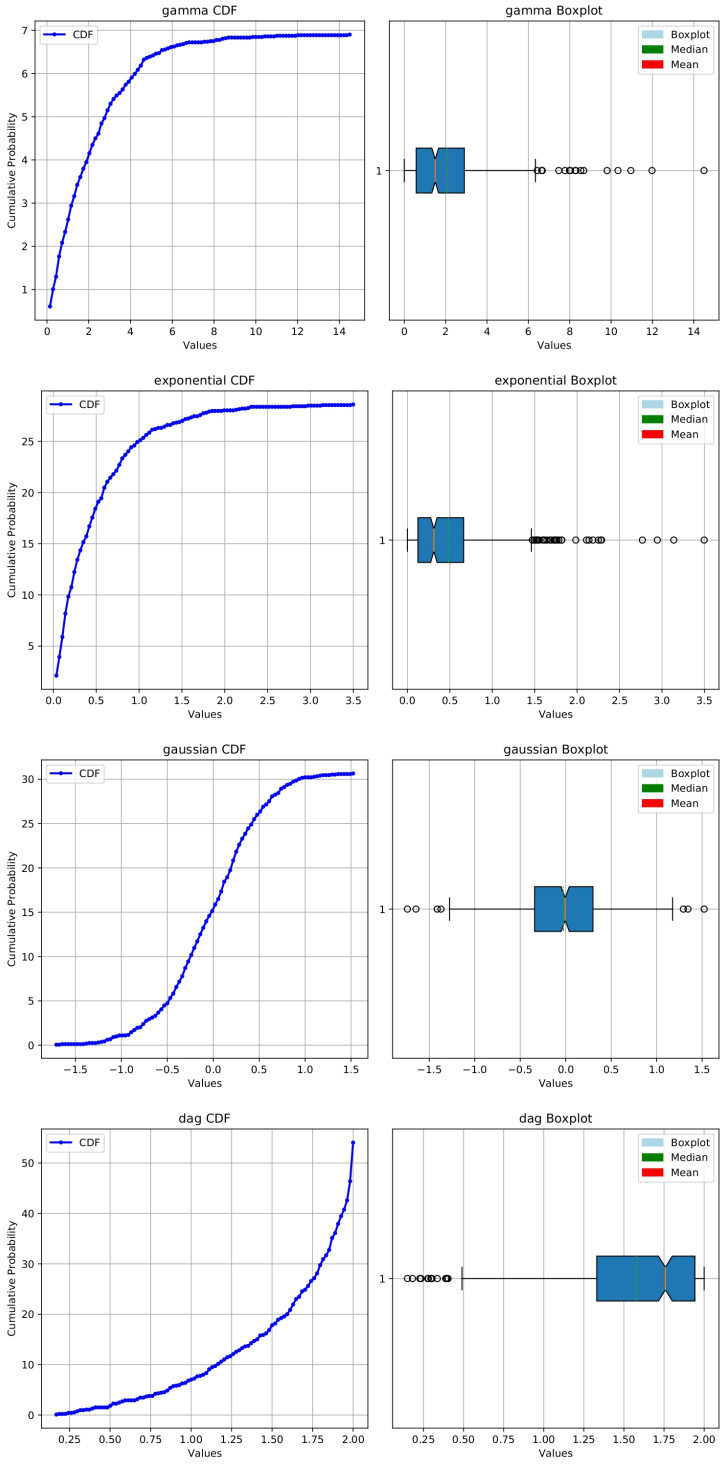

Figure 1: Boxplots and CDF plots for Continuous Distributions

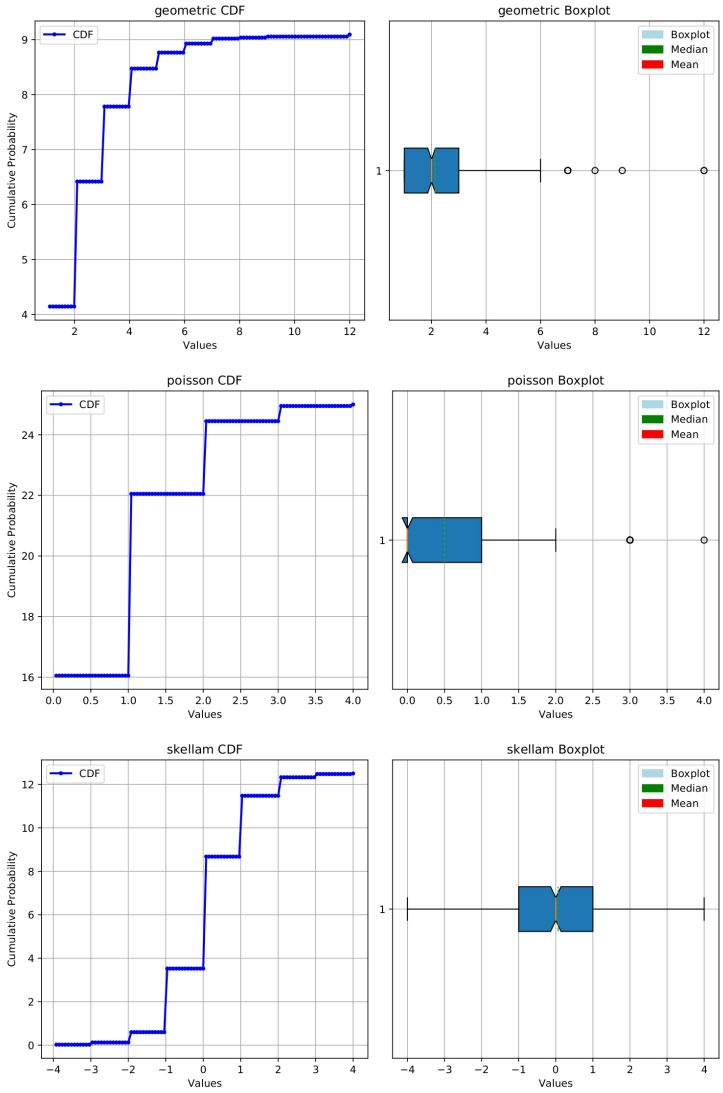

Figure 2: Boxplots and CDF plots for Discrete Distributions

These figures were taken from the configuration as mentioned in the file `configuration-1.cfg` Here is the GitHub link for all the code: `https://github.com/sabhi303/PlotGraphsA3.git`
It also contains the output in the output folder, i.e., CSVs and PDFs of generated graphs.

3.3 Additional Notes and Configuration Details

The tool incorporates several additional features and configuration details above the basic requirements:

3.3.1 Use of configparser

Instead of reading the configuration file line by line, I have utilized the `configparser` library to parse the configuration file. This allows for a more structured approach to reading and extracting data from the configuration, ensuring adherence to a specific format. More details about `configparser` can be found at: `https://docs.python.org/3/library/configparser.html`

3.3.2 Configuration File Format

The format of the configuration files used by the tool may differ from the sample configuration file provided. Additionally, the distribution names in the configuration files are written in full rather than using abbreviations (First three letters).

3.3.3 Argparse for Graph Creation

The `argparse` library is employed to enable optional graph creation functionality. Users can request the generation of graphical plots alongside the generated sequences by specifying the `--plot` flag in the command line.

4 Conclusion

In conclusion, the developed tool represents a comprehensive solution for generating and visualizing various probability distributions specified in configuration files. The tool offers flexibility, efficiency, and usability by integrating Python libraries such as `NumPy` and `Matplotlib`.

The tool's functionality spans three main components: the Driver, Generator, and Plotter. The Driver serves as the central module, coordinating the operations of the Generator and Plotter. The Generator module handles the parsing of configuration files and the generation of sequences or distributions. The Plotter facilitates the visual representation and analysis of the generated distributions through graphical plots.