Project Milestone Progress Report

Project #18 - Group 208 - Hugo Dupouy (GT903738077)

1. Abstract

Based on the Simulation class, we took an interest in the random variate generation routines as they are the bases of a Monte Carlo simulation (Law, 2015). Moreover, law (2015) mentioned that these random variates are generated by hands or by computer with special techniques in many cases. Therefore, this research project aims to build a library where users would be able to produce random variates for discrete and continuous distributions routines in Python. Indeed, we will create a base that will take the form of a program to be run offline.

2. Background and Description of Problem

2.1. Background

Within this class, we understood the importance of distributions in applying simulation to our dataset. We covered the identification of random variate generation, but we were missing a more manual application than using Excel formulas. Indeed, building a dataset of random variates for a specific distribution may take a while. This is why we aim to develop a library of routines for generating random variates. The routines' outcomes will help in having general ideas about the dataset. This project will cover a literature review and the main findings with the development and guidelines on the library program.

2.2. Literature review

This section will discuss the purpose of this project in creating a library of random variate for some significant distributions based on Python through related research conducted before hands.

Created two decades ago, Python's coding language has provided high-level data structures to make interactive, interpreted, and oriented objects (Dhruv, Patel, & Doshi, 2021). The success of Python lies in its uncomplicated syntax while remaining a robust and formidable programming language in the world of data science (Dhruv, Patel, & Doshi, 2021). One of the main objectives of this project is to create an open-source library, meaning to make all the codes available in the hope of "open science" (Daele, Hoey, & Nopens, 2015). Furthermore, it would allow individuals to make future contributions to the original code published by selecting their contribution part, their level of assistance, and the ease of integrating the contributions to the project (Heron, Hanson, & Ricketts, 2013).

Within this python program, we will create a base for some probability and statistical systems of distribution to generate random variates (Goldsman & Goldsman, 2020). Based on Goldsman's class, we will implement the following distributions: Bernoulli, Binomial, Geometric, Negative Binonial, Poisson, Uniform, Exponential, ErlanGamma, Triangular, Beta, Weibull, Cauchy, and Normal (Goldsman & Goldsman, 2020; Law, 2015)(Appendix A). We will be using the generation of random variate for each probability distribution from the package Scipy, which allows statistical functions that combine different methods (The SciPy community, 2022). The graph will be generated based on Seaborn and Matplotlib package (Waskom, 2021; The Matplotlib Development team, 2021).

3. Main Findings

3.1. Development

The code is based on the Python language that will facilitate the creation of the run program, and this language is now widely spread among data analysts. We have used the software Visual Studio code because of the ease of combining it with Git and Github (Microsoft, 2022; Software Freedom Conservancy, Inc., 2022; GitHub, Inc., 2022). This software allows a smooth experience when uploading files to Github, and would resemble a similar experience to a working environment. We have established our run program on the Process Flow Chart to have a simple but straightforward loop for the user to have a seamless experience (Appendix B). The code will be provided in a separate file called “project_group_208.py”. Inside that code file, we have attempted to describe the whole process already, but we will now cover the three main parts: the technical part, the distributions, and the graphs and statistics.

Regarding the technical part, we have 3 main parts with welcome(), distribution_choice(), and distribution_call(). The welcome() function display a message and trigger distribution_choice() (Figure 1).

```python
# Welcome message to start the program
def welcome():
    print("Welcome to the Random Variate Distribution Generator\n")
    return distribution_choice()
```

Figure 1: welcome() function code

The distribution_choice() function will display the menu of the distributions. Then, it will ask the user to enter the number of the distribution based on the menu. The entry gets double-checked to have a correct input among the 14 distributions and returns the distribution_call() with the input number (Figure 2). The distribution_call() function will call the proper distribution based on its 'distribution_number' (Figure 3).

```python
# Asking the user to choose the distribution
def distribution_choice():
    print("Here are the availablee Distributions to generate RV:\n",
          "1  - Bernoulli\n",
          "2  - Binomial\n",
          "3  - Geometric\n",
          "4  - Negative Binomial\n",
          "5  - Poisson\n",
          "6  - Uniform\n",
          "7  - Exponential\n",
          "8  - Erlang\n",
          "9  - Gamma\n",
          "10 - Triangual\n",
          "11 - Beta\n",
          "12 - Weibull\n",
          "13 - Cauchy\n",
          "14 - Normal\n"
          )
    # Retrieving the distribution number from user until we have a correct number
    distribution_number = int(input("Please kindly enter the number of the distribution: "))
    while distribution_number not in range(1,15):
        distribution_number = int(input("Please enter an interger number from the list above (ex: 14): "))

    return distribution_call(distribution_number)
```

Figure 2: distribution_choice() function code

```python
# Calling the functions
def distribution_call(distribution_number):
    #double checking that the number is an integer and in-between
    assert isinstance(distribution_number, int)
    assert distribution_number in range(1,15)

    #Launching the distribution based on the number
    if distribution_number == 1:
        return bernoulli()
    elif distribution_number == 2:
        return binomial()
    elif distribution_number == 3:
        return geometric()
    elif distribution_number == 4:
        return negative_binomial()
    elif distribution_number == 5:
        return poisson()
    elif distribution_number == 6:
        return uniform()
    elif distribution_number == 7:
        return exponential()
    elif distribution_number == 8:
        return erlang()
    elif distribution_number == 9:
        return gamma()
    elif distribution_number == 10:
        return triangular()
    elif distribution_number == 11:
        return beta()
    elif distribution_number == 12:
        return weibull()
    elif distribution_number == 13:
        return cauchy()
    elif distribution_number == 14:
        return normal()
    else:
        return distribution_choice()
```

Figure 3: distribution_call() function code

The distribution part is based on the distribution name, asking for users' inputs, the dataset’s creation, and the return part. Below is a screenshot of one of the distributions (Figure 4).

Figure 4: Bernoulli RV distribution code

```python
## Bernoulli
def bernoulli():
    #Message
    x="Bernoulli distribution"
    print(x)

    #Inputs
    p = float(input("Enter your probability (between 0 and 1): "))
    n = int(input("Enter your sample size: "))

    #Generating RV
    from scipy.stats import bernoulli as y
    data = y.rvs(p=p, size=n)
```

We are first displaying a message about which distribution the user picked. Then, we ask the user to enter some inputs for the distribution of random variate. In the above case, we ask for a probability between 0 and 1 under 'p' and the sample size under 'n'. These variables are then used to generate the RVs into a dataset. We have set up 'y' for the distribution name because we will use the distribution Scipy name when calling the stats() from Scipy into the statistics() function. Last but not least, we return three functions: statistics() to get the main statistics with the mean and variance, graph_generator() to generate the graph, and the distribution_end() to continue or stop running the program. The generation of the dataset, statistics(), and graph_generator() are based on the manual from Scipy (The SciPy community, 2022).

Furthermore, the statistics() function retrieves the distribution Scipy name, our distribution name, and the user's inputs (Figure 5). Based on this information, we need to pick the correct distribution and method from Scipy, as each of them combined does require different parameters. In this case, Bernoulli distribution needs only 'p', but Beta needs 'a' and 'b' parameters, which are utterly different in the method stats() from Scipy (The SciPy community, 2022). Below in Figure 5, you will find all the employed RV distribution methods with their respective parameters. It will then display the central moments with the mean, the variance, skewness, and/or the kurtosis.

```
#Statistics
# The y.stats() requires different parameters based on the distribution y
def statistics(y,x,a=None,b=None,c=None,n=None,p=None):
    if a == None and b == None and c == None and n == None and p != None:
        mean, var, skew, kurt = y.stats(p, moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    elif a != None and b != None and c == None and n == None and p == None:
        mean, var, skew, kurt = y.stats(a, b, moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    elif a != None and b == None and c != None and n == None and p == None:
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    elif a == None and b == None and c == None and n != None and p != None:
        mean, var, skew, kurt = y.stats(n, p, moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    elif a != None and b == None and c == None and n == None and p == None:
        mean, var, skew, kurt = y.stats(a, moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    elif a == None and b == None and c != None and n == None and p == None:
        mean, var, skew, kurt = y.stats(c, moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
    else:
        mean, var, skew, kurt = y.stats(moments='mvsk')
        print(x,"/n mean=",mean,"/n the var=",var,"/n the skewness=",skew,"/n and/or kurtosis:",kurt)
```

Figure 5: statistics() function code

In addition, we made a graph_generator() function to generate the graph of each distribution. As shown in Figure 6, this function requires only the RV dataset ('data') and the distribution's name to be put into the graph's title. We use the Seaborn package to define the chart and Matplotlib to show into an interactive output graph (Figure 7).

```python
#Graph
def graph_generator(data,x):
    # import seaborn
    import seaborn as sns
    # settings for seaborn plotting style
    sns.set(color_codes=True)
    # settings for seaborn plot sizes
    sns.set(rc={'figure.figsize':(5,5)})
    #import matplotlib.pyplot as plt
    import matplotlib.pyplot as plt

    # graph
    graph = sns.displot(data,
                        bins='auto',
                        kde=True,
                        color='purple')
    graph.set(xlabel=x, ylabel='Frequency')
    return plt.show()
```

Figure 6: graph_generator() function code

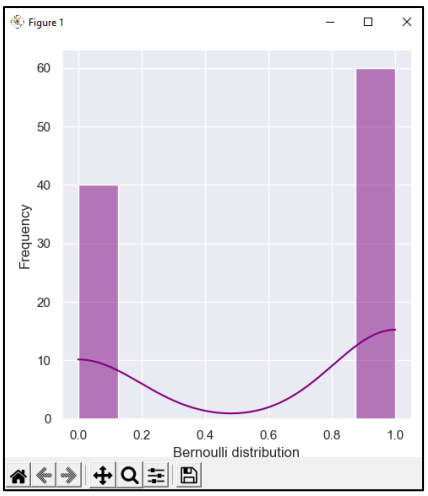


Figure 7: graph output

Last but not least of our distribution function' outcome, we have the distribution_end() function. After closing the graph on VS Code -at least-, the function begins (depending on the editor, it may have started already). This function asks the users if he wants to continue or not and double-checks their inputs. Based on the choices, '1' will return the user to the distribution menu, and '0' will stop running the program.

```python
def distribution_end():
    # Asking to continue or not
    answer = int(input("Do you want to continue with a new distribution or not ?\n Enter 1 for Yes and 0 for No: "))
    while answer not in range(0,2):
        answer = int(input("Please double check your choice\n Enter 1 or 0: "))
    # Asnwer choice
    if answer == 1: return distribution_choice()
    elif answer == 0: return print("Thank you for trying our Library of distributions. Bye-bye!")
```

Figure 8: distribution_end() function code

3.2. Guidelines

We will now go through an example of how the program run. First, we run the python file and check our terminal (Windows machine). Next, we have a welcome message followed by the menu of the distributions and asking the user to enter the number corresponding to the distribution (Figure 9).

```
PS C:\Users\Hugo\Dropbox\GTx\Term 2\ISYE 6644 - Simulation\Project> python -u "c:\Users\Hugo\Dropbox\GTx\Term 2\ISYE 6644 - Simulation\Project\pro
ject_group_208.py"
Welcome to the Random Variate Distribution Generator

Here are the availablee Distributions to generate RV:
 1  - Bernoulli
 2  - Binomial
 3  - Geometric
 4  - Negative Binomial
 5  - Poisson
 6  - Uniform
 7  - Exponential
 8  - Erlang
 9  - Gamma
 10 - Triangual
 11 - Beta
 12 - Weibull
 13 - Cauchy
 14 - Normal

Please kindly enter the number of the distribution:
```

Figure 9: Step 1- Welcome & Distribution Menu

As the user, we will enter ‘1’ for the Bernoulli distribution. The program now displays the distribution name we are about to use and asks us to insert the inputs to generate the RV distribution. We will try with the probability of 0.6 and enter a sample size of 100. Based on Figure 10, the program has generated the four moments and the graph of our RV distribution in Figure 7.

```
Please kindly enter the number of the distribution: 1
Bernoulli distribution
Enter your probability (between 0 and 1): 0.6
Enter your sample size: 100
Bernoulli distribution /n mean= 0.6 /n the var= 0.24 /n the skewness= -0.4082482904638634 /n and/or kurtosis: -1.8333333333333308
```

Figure 10: Step 2 - Pick, adjust, and get the Distribution’s outputs

In VS code, once we close the graph window, the program asks us if we would like to continue with a new distribution or not. If we insert ‘1’ and press enter, the program takes us back to the distribution menu that we saw in the beginning (Figure 11). Appositively, if we would have picked ‘0’ for no, a good-bye message comes out, and the program stops (Figure 12).

“1 for Yes”

```
Do you want to continue with a new distribution or not ?
 Enter 1 for Yes and 0 for No: 1
Here are the availablee Distributions to generate RV:
 1  - Bernoulli
 2  - Binomial
 3  - Geometric
 4  - Negative Binomial
 5  - Poisson
 6  - Uniform
 7  - Exponential
 8  - Erlang
 9  - Gamma
 10 - Triangual
 11 - Beta
 12 - Weibull
 13 - Cauchy
 14 - Normal

Please kindly enter the number of the distribution:
```

Figure 11: Choice to continue

“0 for No”

```
Do you want to continue with a new distribution or not ?
 Enter 1 for Yes and 0 for No: 0
Thank you for trying our Library of distributions. Bye-bye!
```

Figure 12: Choice to stop

4. Conclusion

This project was an outstanding achievement as we now have built a library where users can produce random variates for discrete and continuous distributions routines in Python. Moreover, the code followed a simple yet, robust application of the Python script to meet the requirements with the help of external packages. Yet, the project has been published on Github for the coding community to contribute to its expansion.

Overall, the project has been challenging for a couple of reasons. First, I was a stand-alone person who had to do everything from scratch. Second, I learned Github, Git, and Visual Studio Code to be able to post the code online and track changes, and be closer to an authentic experience in an office team. Third, I review all my previous python classes about loops and functions, especially optimizing the code from 600 to 420 lines. Finally, to keep everything under five pages while being clear and concise.

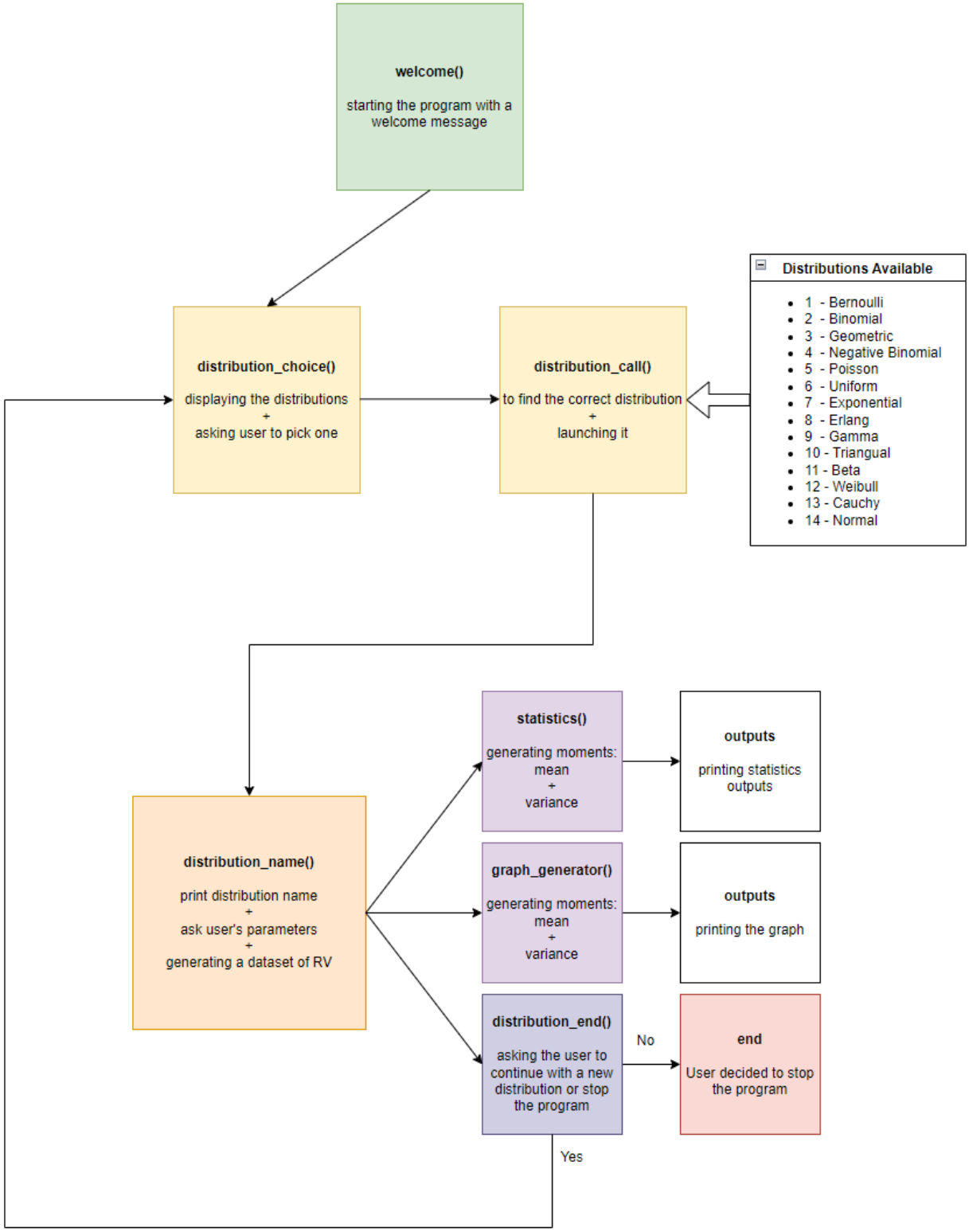
Regarding the future prospect of this library, it is to continue working on it to be available online on a PyPi server. One of the goals would be to import the package, call the distribution directly and add the necessary parameters.

Appendix

Appendix A - Table of Distributions

Distributions	Formula	Graph	Moments	Callable	Running offline	Github
Discrete Distributions						
Bernoulli(p)	✓	✓	✓	✓	✓	✓
Binomial(n,p)	✓	✓	✓	✓	✓	✓
Geometric	✓	✓	✓	✓	✓	✓
Negative binomial(r,p)	✓	✓	✓	✓	✓	✓
Poisson(λ)	✓	✓	✓	✓	✓	✓
Continuous Distributions						
Uniform(a,b)	✓	✓	✓	✓	✓	✓
Exponential(λ)	✓	✓	✓	✓	✓	✓
Erlangk(λ)	✓	✓	✓	✓	✓	✓
Gamma(α, λ)	✓	✓	✓	✓	✓	✓
Triangular(a, b, c)	✓	✓	✓	✓	✓	✓
Beta(a, b)	✓	✓	✓	✓	✓	✓
Weibull(a, b)	✓	✓	✓	✓	✓	✓
Cauchy	✓	✓	✓	✓	✓	✓
Normal (μ, σ^2)	✓	✓	✓	✓	✓	✓

Appendix B - Process Flow Chart



The flow chart was created on diagrams.net from JGraph Ltd. (2022).

References

Bienvenu, J. (2020). *Deep dive: Create and publish your first Python library.* Retrieved from https://towardsdatascience.com/deep-dive-create-and-publish-your-first-python-library-f7f618719e14

Daele, T. V., Hoey, S. V., & Nopens, I. (2015). pyIDEAS: an Open Source Python Package. *Computer Aided Chemical Engineering, 37*(1), pp. 569-574. doi:10.1016/B978-0-444-63578-5.50090-6

Dhruv, A. J., Patel, R., & Doshi, N. (2021). Python: The Most Advanced Programming Language for Computer. *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies*, pp. 292-299. doi:10.5220/0010307902920299

GitHub, Inc. (2022). *GitHub.* Retrieved from https://github.com

Goldsman, D., & Goldsman, P. (2020). *A First Course in Probability and Statistics.* USA. doi:201127.210818

Heron, M., Hanson, V. L., & Ricketts, I. (2013). Open source and accessibility: advantages. *Journal of Interaction Science, 1*(2), pp. 1-10. doi:10.1186/2194-0827-1-2

JGraph Ltd. (2022). *diagrams.net.* Retrieved from https://app.diagrams.net

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

Microsoft. (2022). *Visual Studio Code.* Retrieved from https://code.visualstudio.com

Software Freedom Conservancy, Inc. (2022). *git--local-branching-on-the-cheap.* Retrieved from https://git-scm.com

The Matplotlib Development team. (2021). *Matplotlib: Visualization with Python.* Retrieved from https://matplotlib.org

The SciPy community. (2022). *Statistical functions (scipy.stats).* Retrieved from https://docs.scipy.org/doc/scipy/reference/stats.html

Waskom, M. (2021). *Seaborn: statistical data visualization.* Retrieved from https://seaborn.pydata.org