Teaching Modeling, Simulation, and Performance Evaluation Course Online with Jupyter Notebook: Course Development and Lessons Learned

Farag M. Sallabi
Computer and Network Engineering Dept.
United Arab Emirates University
Al Ain, UAE
f.sallabi@uaeu.ac.ae

Sanja Lazarova-Molnar
Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
slmo@mmmi.sdu.dk

***Abstract*—This innovative practice full paper presents a case of teaching modeling, simulation, and performance evaluation course online during the COVID-19 pandemic for graduate students. The course includes theoretical and practical sessions with varying complexity. Students should have a good background in math, statistics, and probability. Students must also have good experience in one of the computer programming languages to solve the homework and work on the term project. The challenge is how to teach these topics online and engage the students in the course as they learn in face-to-face classes. Delivering the course using PowerPoint slides and a whiteboard is not suitable for teaching the class online. Therefore, there must be an alternative way to deliver the course online. We noticed a growing interest in using Jupyter Notebook in teaching, which motivated us to apply it to the mentioned course with some innovations. Jupyter Notebook is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations, and narrative text. We want to share our experience teaching this course online using Jupyter Notebook in this innovative practice. We will share the course development plan, delivery mode, lessons learned, and student feedback. I will also highlight maximizing the benefits of the Jupyter Notebook using add-ins and tools useful for teaching, such as converting the Jupyter Notebook to a slide show. Developing courses in Jupyter Notebook could be time-consuming and frustrating, especially if there are a lot of math equations, tables, and drawings. This effort pays off in terms of the quality of the instruction and learning, and it gives students a tool to help them practice and engage with the course material.**

Keywords—Jupyter Notebook, online teaching, simulation, modeling, course development

I. INTRODUCTION

Teaching modeling, simulation, and performance evaluation course for graduate students is a mix between theory and practice. The course catalog description includes computer simulation concepts and modeling theory, probability distributions and queuing theory, random number generation, probability distribution generation, data collection, and input/output analysis. Students should have a solid foundation in probability, statistics, and mathematics. Students must also have good experience in one of the computer programming languages to solve the homework, work on the term project, and work on the coding examples in the lecture. The course conduct includes theoretical and practical sessions with varying complexity as students should apply what they learn in this course in simulation. The challenge is teaching these topics online and engaging students in the course as they learn in face-to-face classes. Delivering the course using PowerPoint slides and a whiteboard is not suitable for teaching the course online. Therefore, there must be a way to facilitate delivering this course online. During the COVID-19 pandemic, we moved to online teaching. After extensive research on online teaching tools, we decided to use the Jupyter Notebook.

Jupyter Notebook is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations, and narrative text [1]. There is a growing interest in using Jupyter Notebook in teaching, which motivated us to apply it to the modeling, simulation, and performance evaluation course with some innovations. The interest in using Jupyter Notebook has increased during the COVID-19 pandemic as it was very convenient for interactive online teaching. The interest is not only for STEM (Science, Technology, Engineering, and Mathematics) courses, but other non-STEM subjects, such as social sciences, used Jupyter.

We first taught the course in fall 2020 and then in spring 2022. The experience from the first time was extraordinary and beneficial to the students, and student feedback was positive. Students used Jupyter Notebook or Google Colab (Collaboratory) to apply what they saw in class and work on their homework and project. Some students use the same approach in their thesis research and other courses.

Teaching and learning can use any available online resources that improve teachers' and students' productivity and give them flexibility and support for collaborative work. In engineering courses, Jupyter Notebook provides a programming environment for developing and sharing educational materials, combining different types of resources such as text, images, and code in several programming languages in a single document, accessible through a web browser. This environment is also suitable for providing access to online experiments and explaining how to use them [2]. Jupyter Notebook has been recently used to support teaching and learning in many areas [3]. Alberto et al. [2] used Jupyter Notebook to teach "Data Analysis and Transformation" in the 2nd year of the Informatics Engineering B.Sc. course at the University of Coimbra, Portugal. Andres et al. [4] used it as an interactive tool in a virtual laboratory to teach the subject of mechanical engineering for optimizing manufacturing processes.

The Jupyter Notebook provides a suitable programming environment to teach engineering labs [4]. Willis et al. [5] report that while using Jupyter Notebook to develop the students' coding skills, they investigated how the cell structure of Jupyter Notebook can be exploited to improve the students' understanding of how to structure a report on a data investigation. Jupyter Notebook has also been used to accelerate learning and curriculum development. The authors

in [6] used the technical features of Jupyter Notebook and Docker to develop and deliver three Geo-Computation modules to Geography undergraduates, with some progressing to data science and analytics roles. The authors in [7] used Jupyter Notebook to teach the "Theory of Structures and Industrial Constructions" course module that involves matrix structural analysis, which requires intensive matrix calculus and computational methods that cannot be solved with pen and paper alone. Jupyter Notebook can be used to teach elementary engineering undergraduate courses such as linear algebra [8]. The authors introduce a new form of computer-aided linear algebra course by applying a set of Python computer programs implemented on the Jupyter platform. They design some visual examples demonstrating the practical applications of linear algebra to help students understand the intrinsic physical meanings of some concepts. The authors aim to improve the development of innovative teaching methodologies.

Masanori [9] describes how to use Python to teach topics in a microeconomic theory course at the undergraduate level. Specifically, the author describes how to use Python to solve optimization problems. They provide a Program code for every example to encourage replication and experimentation. Dominguez et al. [10] used Jupyter Notebook to support the teaching of "Introduction to Chemical Engineering" courses to 1st-year B.Sc. students in Chemical Engineering, 2nd-year B.Sc. students in Chemistry, and 3rd-year B.Sc. students in Biochemistry at the Universidad Complutense de Madrid (UCM). Thomas et al. [11] used Jupyter to provide rich user interfaces to interact with a supercomputer. The interfaces are easy and more productive, which attracts new kinds of users and helps to expand the application of supercomputing to new science domains. Jupyter Notebook has been widely accepted as interactive data science and scientific computing tool among researchers [12]. Analysts can use Jupyter Notebook to write documents that contain software code, computational output, formatted text, and data visualizations [13]. Muller et al. [14] developed an open-source educational material for a "Preparation Course for Python (PCP)" that use signal processing as an application for practicing the programming concepts. The interactive documents in the PCP notebooks include executable code, textbook-style justifications, mathematical formulas, graphs, pictures, and audio examples.

The remainder of this paper is organized as follows: Section II presents the course structure. The proposed course development and implementation are explained in Section III. Section IV presents the discussion and best practices. Finally, Section V presents the conclusions.

II. COURSE STRUCTURE

Modeling, Simulation & Performance Evaluation is a graduate course offered to the PhD students in the College of Information Technology, United Arab Emirates University. The course contents include; Computer simulation concepts and modeling theory, probability distributions and queuing theory, random number generation, probability distribution generation, data collection and input analysis, discrete modeling and simulation concepts, and Monte Carlo Simulation. Upon completion of this course, students should be able to achieve the following course learning outcomes (CLOs):

- Describe modeling and simulation, and performance evaluation techniques.
- Apply simulation and performance evaluation techniques to develop and evaluate the relative merits of alternative system design models.
- Analyze simulation and performance evaluation results clearly and coherently.
- Apply appropriate modeling techniques to real-world problems and data sets.

The first outcome, based on the revised Bloom's Taxonomy [15], requires the students to understand the main components of modeling and simulation and be able to discuss them. The students should also identify the performance measures techniques and describe them. In the second outcome, the students should apply what they learned in the first outcome. The students will apply these techniques through homework assignments and a project. The students should be able to analyze simulation results, make informed decisions about their simulation, and compare the results with other alternative systems. Students explore different real-world problems during the term project and collect related data. They formulate, design, model, and implement the project based on the tools learned in this class. To realize all of these tasks, Table 1 presents the course outline. The outline is based on a 15 weeks semester. The course content follows a well-known textbook [16][17].

TABLE I. COURSE TOPICAL OUTLINE

Week	Topic(s)	CLO1	CLO2	CLO3	CLO4	Course Activities
Week 1	Introduction to Simulation and Modeling	√				Lecture/Coding/Class Discussion
Week 2	Basics of Discrete-Event Simulation	√				Lecture/Coding/Class Discussion/Homework1
Week 3	Introduction to SimPy: A Discrete-Event Simulation Package based on Python Simulation Examples	√				Lecture/Coding/Class Discussion
Week 4	Statistical Models in Simulation		√			Lecture/Coding/Class Discussion/Project Discussions
Week 5	Probability Distributions		√			Lecture/Coding/Class Discussion
Weeks 6, 7	Queuing Models		√			Lecture/Coding/Class Discussion/Homework2
Week 8	Random Number Generation and Random Variate Generation	√				Lecture/Coding/Class Discussion
Week 9	Input Modeling		√	√		Lecture/Coding/Class Discussion
Week 10	Verification, Calibration, and Validation of Simulation Models	√	√	√	√	Lecture/Coding/Class Discussion/Project Discussions

Week 11	Performance measurement in simulation-based environments				√	Lecture/Coding/Class Discussion/Homework3
Week 12	Monte Carlo Simulation	√				Lecture/Coding/Class Discussion/Homework4
Weeks 13, 14	Projects Presentations and Discussion	√	√	√	√	Project Presentations and Discussions

The course includes theoretical and practical activities. The course instructor develops a Jupyter Notebook for each week (lecture) and makes them available to the students through the Blackboard system. Students then download a copy of the Jupyter Notebook for each lesson in which they perform various practical in-class and off-class tasks. The course assessment tools include theoretical and programming assignments, term projects, and final exams. Students are also asked to do lecture scribing, in which they revise the lesson Notebook, suggest improvements, and add examples and formatting. The term project consists of a real-world problem, where the students select any problem of their choice and work on it after agreement with the course instructor. The project activities should involve all tasks taught in class, starting from formulating the problem to the output analysis.

III. COURSE DEVELOPMENT AND IMPLEMENTATION

All lectures are developed and implemented by Jupyter Notebook. In the first week of the course, the instructor gives an orientation about the course and the used tools. Lectures are distributed to the students in a Jupyter Notebook format to use in the class and perform practical tasks. Students are instructed to install and configure Jupyter Notebook on their computers. The best way to install and configure Jupyter Notebook is through Anaconda distribution, which is free. Students may also use Google Colaboratory (Colab). All lectures start with the relevant references used in the Notebook. References are textbooks, references, tutorials, videos, Python libraries, etc. Then a list of the lecture objectives is stated. The Jupyter Notebook is organized into cells. Each concept/task is explained in one or more cells, followed by an interactive case study or example, as shown in Fig. 1. The following subsections discuss the development and implementation details of each lecture.

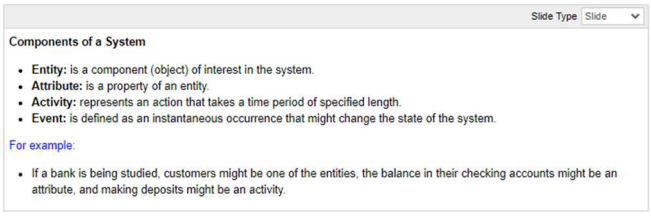

Fig. 1. A Jupyter Notebook cell shows the components of a system with an example

A. Week 1: Introduction to Simulation and Modeling

This lecture is an introduction to simulation and modeling. It discusses the general concepts of simulation and when to use simulation. It explores the concepts of a system and system model. Then discusses the advantages and disadvantages of the simulation. It also states the steps in building and using a system simulation model. The main objectives of the lecture are as follows:

- Define a system and its characteristics.
- Define simulation and how it is used to imitate a system.
- Identify discrete and continuous systems.
- Outline the steps of a simulation study.

The explanations of these objectives follow the textbook [16] and the references. This lecture doesn't involve coding but includes some tables and figures. Tables can be written in Markdown language. For example, the following table shows some examples of systems and their components written in Markdown and the rendered output.

```
| System | Entities | Attributes | Activities | Events | State Variables |
|:----------|:----------|:----------|:----------|:----------|:----------|
| Banking | Customers | Balance | Deposits | Arrival | Busy tellers|
|Rapid rail |Riders| Origin| Traveling| Arrival| Riders waiting|
```

System	Entities	Attributes	Activities	Events	State Variables
Banking	Customers	Balance	Deposits	Arrival	Busy tellers
Rapid rail	Riders	Origin	Traveling	Arrival	Riders waiting

Figures can be inserted in the Jupyter Notebook as an image. For example, to insert the image of types of system models, we use the following syntax:*![Types of Models](models.png)*. This lecture also gives brief tutorials about the tools used in the course. These tutorials include; Jupyter Notebook, Markdown, Latex, and Python. Then provide links to some other useful tutorials.

B. Week 2: Basics of Discrete-Event Simulation

In this lecture, we introduce the general principles of discrete-event simulation. Topics include event scheduling and time advance algorithm. The lecture objectives are:

- Discuss discrete-event system simulation.
- Identify the components and organization of a discrete-event simulation model.
- Explain time-advance mechanisms.
- Run discrete-event examples.

After discussing the theoretical part of discrete-event system simulation, we develop and implement the first simulation program. The simulation program consists of the main program and supporting routines. The challenging part of this lecture is the time-advance mechanism and how to keep track of the time while running the program. To explain the time-advance mechanism, we run a Python program in Jupyter Notebook and monitor the time advance. The example program simulates a single-server queueing system [17]. The example shows how to simulate a single-server queueing system such as a barbershop, call center, or bank system with one teller. Fig. 2 shows the system model to be simulated.

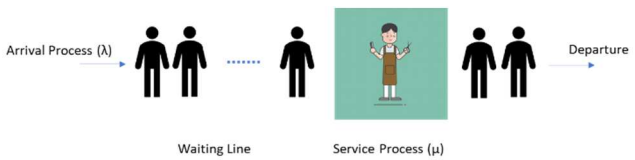

Fig. 2. Barbershop system model.

The barbershop can be simulated as a Python class that includes all instance variables and methods to model the system. Fig. 3 presents a snapshot of the Barbershop class. The class cell is followed by a Test cell to check the validity of the methods in the class. For example, to test the statistical counters, we may run the Test cell shown in fig. 4. The arrivals and the service rate follow an exponential distribution, which is also implemented in Python. To explain the exponential distribution to the students, we create a cell and run the exponential distribution as depicted in fig. 5 with the cell output followed by a discussion cell.

```
In [ ]:                                                    Slide Type -
# Barbershop system model implementation.

class Barbershop:
    def __init__(self, interarrival_time, service_rate):
        self.interarrival_time = interarrival_time
        self.service_rate = service_rate

        self.num_in_system = 0
        self.clock = 0

        #Statistical counters
        self.num_arrivals = 0
        self.t_depart = float('inf')
        self.num_depart = 0
        self.total_wait = 0

        self.t_arrival = self.generate_interarrival()

    def advance_time(self):
        t_event = min(self.t_arrival, self.t_depart)

        self.total_wait += self.num_in_system * (t_event - self.clock)

        # Advance the simulation clock
        self.clock = t_event

        if self.t_arrival <= self.t_depart:
            self.handle_arrival_event()
        else:
            self.handle_depart_event()
```

Fig. 3. Barbershop class implementation snapshot

```
In [ ]:                                                    Slide Type -
# Test Cell

time_history = []
num_arrivals_history = []

interarrival_time = 1/3
service_rate = 1/4

sim = Barbershop(interarrival_time, service_rate)

for i in range(100):
    sim.advance_time()

print(f"Total wait: {sim.total_wait}")
print(f"Number of Arrivals: {sim.num_arrivals}")
print(f"Number of Departures: {sim.num_depart}")
print(f"Number in system: {sim.num_in_system}")
```

Fig. 4. Test cell for statistical counters

```
In [1]:                                                    Slide Type -
# Inter-arrival time modeled by Exponential Distribution
# The exponential distribution is the probability distribution
# of the time between events.

import numpy as np # We need this for the exponential RV generation method.
import matplotlib.pyplot as plt # We need this to plot the distribution.

# Set the number of events (arrivals) to 3; 1/3 is the time between events.
nums = np.random.exponential(1/3, 10000)

# Draw the histogram; the x-axis shows the inter-arrival time
# the y-axis shows the frequency of occurance.
# The total area of the histogram should be 1.
plt.hist(nums, bins = 100, density = True)
plt.show()
```

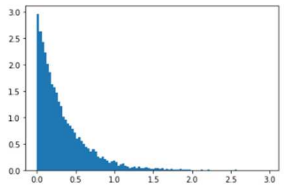

Fig. 5. Cell for generating exponential random variable and drawing the distribution

C. *Week 3: Introduction to SimPy*

Simulation code can be implemented from scratch, where we keep track of events and time, or we can use a simulation framework that organizes events and time advance. In this course, we use the SimPy framework [18]. This lecture has four objectives to understand how SimPy works.

- Explain SimPy framework
- Define Python generator function.
- Discuss SimPy shared resources, containers, and stores.
- Implement SimPy working examples.

We start with an overview of the SimPy package and define the generator function with examples. To use SimPy, we should install it as it doesn't come with the Python distribution. We refer the reader to the SimPy tutorial web page to learn more about SimPy [18]. Fig. 6 presents a code using SimPy. The code implements the timeout event and function generator. To conclude this lecture, we run the same Barbershop simulation program but with SimPy. The code is written by Paul T. Grogan, Stevens Institute of Technology, and is available on YouTube [19].

```
In [ ]:                                                    Slide Type -
# Example to explain the Timeout event and the generator function.
def car(env):
    while True:
        print('Start parking at {}'.format(env.now))
        parking_duration = 5
        yield env.timeout(parking_duration)

        print('Start driving at {}'.format(env.now))
        trip_duration = 2
        yield env.timeout(trip_duration)

# Setup the environment and run the program.
env = simpy.Environment()
env.process(car(env))
env.run(until=15)
```

Fig. 6. Timeout event and the function generator

D. *Week 4: Statistical Models in Simulation*

Most real-world modeling systems are stochastic rather than deterministic. Therefore, a prerequisite for this course is that students have good knowledge of statistics and probability distributions. The lack of this knowledge is covered in this lecture to ensure that students efficiently use statistics and probability distributions in the simulation to model the input and analyze the output. This lecture gives the necessary theoretical work supported by many Python examples. The objectives of this lecture are as follows:

- Define discrete and continuous random variables.
- Define cumulative distribution function.
- Explain expectation, variance, and standard deviation.
- Identify useful statistical models.

We used the built-in random module in Python, the random functions in NumPy, and the SciPy library to generate random variables and run random experiments. For example, the cell in Fig. 7 generates a random variable between (0, 10). Python is rich in statistics and probability distributions, and you can find almost every function you need. Students are very excited to learn and implement these functions, and they stated that these readily available functions make coding easy for them. The students also indicated that running these functions helped them understand the theory behind it.

```
In [4]:                                                    Slide Type -
# Example: Generate a randome number
# imports random module
import random
import numpy as np

# Generates a random number between a given positive range
r1 = random.randint(0, 10)
r2= np.random.randint(0, 10)
print("Random number between 0 and 10 is %s using Random module" %(r1))
print("Random number between 0 and 10 is %s using NumPy library" %(r2))
```

```
Random number between 0 and 10 is 7 using Random module
Random number between 0 and 10 is 2 using NumPy library
```

Fig. 7. Random number generation.

We discussed several statistical models for well-known systems. The probability distributions are discussed in more

detail in week 5. This lecture also presented some useful statistical models needed in real-world simulations.

E. Week 5: Probability Distributions

This lecture discusses some selected probability distributions that describe various probabilistic events. It contains several mathematical equations that are written in Latex. This appears to be time-consuming as writing LaTeX equations in Jupyter Notebook is not easy and prone to mistakes, but it is worth doing. For example, Fig. 8 shows the Binomial distribution written in LaTeX and the rendered output.

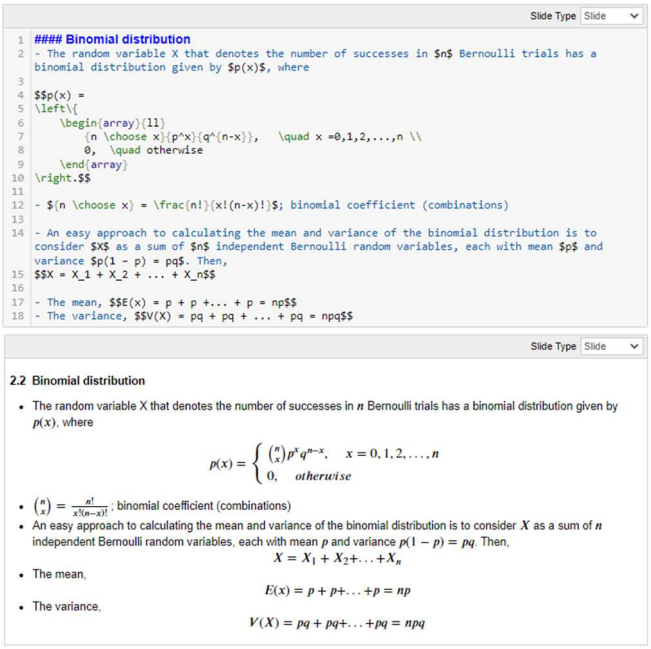

Fig. 8. Binomial distribution in LaTeX and the rendered output

Based on feedback from the students, this lecture exhibited the powerfulness of the Jupyter Notebook and its flexibility in using Python code, Markdown, LaTex, and HTML. All discussed probability distributions are implemented in Python active cells, and the students run the code on their machines with different values. Students gained a great knowledge of using these distributions and gave them insight into how to use them in their graduate research. The students also had a chance to implement all examples in this lecture in Python and discuss the results. Fig. 9 shows how to implement Weibull distribution in Python.

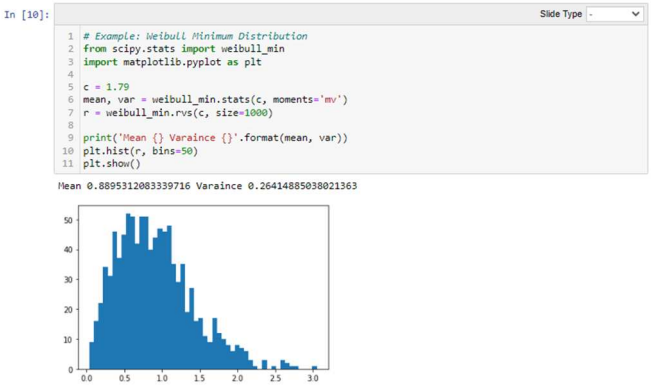

Fig. 9. Weibull distribution

F. Week 6 and 7: Queuing Models

Most real-world systems such as service facilities, production systems, repair and maintenance facilities, communications and computer systems, and transport and material-handling systems are modeled as queueing systems. Therefore, analyzing the queueing models is essential in any simulation and modeling subject. Queueing models provide the analyst with a powerful tool for designing and evaluating the performance of queueing systems. Queueing models can be solved mathematically or analyzed through simulation. In this lecture, we discuss the mathematical and simulation solutions of the queueing models. Jupyter Notebook is an excellent tool to present this lecture. The following are the main objectives addressed in this lecture:

- Discuss the characteristics of queueing systems.
- Identify queueing systems notations.
- Analyze long-run measures of performance of queueing systems.
- Study the steady-state behavior of infinite-population Markovian models.

The Jupyter Notebook is divided into cells. Some cells present the theory of the queueing systems, followed by discussion cells that ask the students to engage in groups to discuss a certain topic. The lecture contains several equations, which are implemented in LaTeX. The main point of this lecture is to teach how to develop the mathematical model and compare it with the simulation results, i.e., the steady-state behavior of the system versus the long-run measures. Therefore, we dedicated some cells to run simulation code for some queueing models. The lecture is supported with homework to implement most of the concepts of the lecture and discuss the results. The students had a chance to verify the two important theorems of "Law of Large Numbers" and "Central Limit Theorem" through simulations.

G. Week 8: Random Number Generation and Random Variate Generation

Generating random numbers that follow a specified distribution is essential in simulating almost all real-world systems. Fortunately, Python is rich in functions that generate random variables and variates. Despite the readily available functions, the theory behind generating the random variables and random variates is important. Therefore, the main objectives of this lecture are:

- Discuss random number generation.
- Discuss random variate generation.
- Provide example code for generating random variables and random variates for some probability distributions.

Fig. 10 shows a Python code to generate a random variable, while Fig. 11 shows the Python code for random variate from an exponential distribution.

```
In [1]:                                                    Slide Type -
# Generate random numbers between [0, 1]
import random
import matplotlib.pyplot as plt

random_list = []
for i in range(0, 1000000):
    n = random.random()
    random_list.append(n)

plt.hist(random_list, bins=100)
plt.show()
```

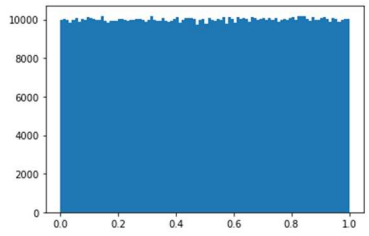

Fig. 10. Random variable generation.

```
In [2]:                                                    Slide Type
# Random variate generation from exponential function.
import numpy as np
import matplotlib.pyplot as plt

random_list_exponential = []
arrival_rate = 3

for i in range(0,1000000):
    n = np.random.exponential(1/arrival_rate)
    random_list_exponential.append(n)

plt.hist(random_list_exponential, bins=100)
plt.show()
```

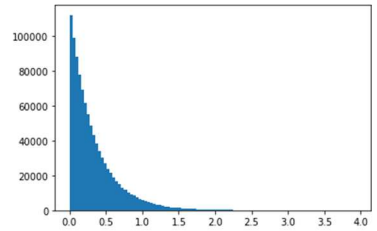

Fig. 11. Random variate generation from the exponential function

H. *Week 9: Input Modeling*

Input modeling is the driving force for any simulation. Input modeling involves collecting data, identifying a probability distribution, choosing parameters of the distribution, and evaluating the distribution with the associated parameters for the goodness-of-fit. Choosing appropriate distributions for input data is a major task from the standpoint of time and resource requirements. Erroneous input data models lead to output data whose interpretation gives misleading recommendations. The objectives of the lecture are:

- Define input modeling.
- Identify the four steps for developing input models.
- Apply goodness-of-fit tests.

Even though this lecture is mainly theoretical, most of the goodness-of-fit tests are readily available in Python, and we used them to run some examples. For example, the code in Fig. 12 presents the Kolmogorov-Smirnov test for two sample datasets.

```
In [1]:
# Kolmogorov-Smirnov test for two sample dataset.

import numpy as np

from scipy.stats import ks_2samp

#generate two datasets
data1 = np.random.randn(100) #Standard Normal distribution.
data2 = np.random.lognormal(3, 1, 100) # Lognormal distribution.

ks_2samp(data1, data2)
```

```
Out[1]: Ks_2sampResult(statistic=1.0, pvalue=2.2087606931995054e-59)
```

Fig. 12. Kolmogorov-Smirnov test for two datasets

I. *Week 10: Verification, Calibration, and Validation of Simulation Models*

Any simulation work is not complete without verification and validation. Validation ensures that the model accurately represents the actual system behavior concerning simulation goals and increases the model's credibility. Validation is concerned with building the correct model, while verification concerns building the model correctly. Verification guarantees that the model is implemented correctly in the simulation software and that the input parameters and the logical structure of the model are represented accurately. The main objective of this lecture is to describe the methods used in the verification and validation process. We found that Jupyter Notebook is a good tool for working on the verification and validation processes. We divide the code into cells in a modular format and test each cell separately for verification. Therefore, before moving to the next module, we confirm that the previous model is correct and supplies accurate results. While developing the code and applying the verification process, we create cells to discuss the input parameters and output results and validate the model, especially when the model is stochastic and produces different output. Jupyter Notebook provides powerful functions to plot the input and output data and make informative analyses and decisions.

J. *Week 11: Performance measurement in simulation-based environments*

The output data from a simulation exhibits random variability when random-number generators are used to produce the values of the input variables. Two different streams or sequences of random numbers produce two sets of different outputs. Therefore, there is a need for statistical output analysis based on the observation. As we stated, Jupyter Notebook is a collection of libraries and functions that produce results in different formats, making it easy to analyze and make decisions about the input parameters and the model. Several examples from the textbook [16] were given in this lecture. Then the students coded these examples in Python and produced output results in different formats. The students then used discussion cells to compare the results and validate the model. One important and confusing task, what is a confidence interval? How do we find it? There is one example in the lecture that calculates the analytical value of the confidence interval. The example uses the percentage points of the t distribution with v degrees of freedom, as shown in Table A.5 in the textbook [16]. To make this example more concrete and clearer, we implemented the same example in Python and found the confidence interval very close to the analytical one. Fig. 13 shows the code to find the confidence interval of the example given in the class with a true mean of 2.64 and variance of 2.28.

Fig. 13. Confidence interval

K. Week 12: Monte Carlo Simulation

Finally, we cannot leave this course without touching Monte Carlo simulation as some students might use them in their research studies. Monte Carlo simulation is a scheme that employs random numbers, which is used to solve certain stochastic and deterministic problems. The main objectives of this lecture are:

- Define Monte Carlo Simulation.
- Evaluate integrals with Monte Carlo simulation.
- Apply Monte Carlo simulation to deterministic problems.
- Apply Monte Carlo simulation to stochastic problems.

Three analytical examples and one in-class exercise were given in this lecture. Then the students simulated the examples in the exercise in Python using Jupyter Notebook. After running the simulation code and seeing the results, the students defined and absorbed the concepts. For example, to evaluate the integral of ***sin x*** in the interval **[0, π]**, which is **2** in elementary calculus, with Monte Carlo simulation, we run the following code shown in Fig. 14.

Fig. 14. Integral of ***sin x*** using Monte Carlo simulation

The student had a chance to implement the code on their computers and change the number of samples to get closer results. As Mote Carlo simulation involves running a large number of samples to converge to the analytical result, the code will take a long time to finish. Therefore, I introduced the topic of the Spark Programming Model [20]. The Spark programming model will exploit the processor's multi-cores and run the program faster. We executed one of the examples without Spark and with Spark and noticed the difference in execution time. This feature was seamlessly implemented in Jupyter Notebook and exploited. Fig. 15 shows the code for using Monte Carlo simulation to find the value of **π**.

Fig. 15. Program to use Monte Carlo to find the value of **π**

IV. DISCUSSION AND BEST PRACTICE

After teaching the course twice with Jupyter Notebook, my impression and experiences are fascinating. This experience encouraged me to write this paper and share my experience. Jupyter Notebook is a great tool for teaching engineering courses online and offline. The capabilities of Jupyter augmented with the add-ins provide an excellent environment for teaching and learning. As Jupyter is still under investigation and adoption for teaching and learning, different communities need to report their experiences and pitfalls of Jupyter to consider in the updated versions. For example, the authors in [21] stated that as Jupyter notebooks have grown, so has criticism of the notebook, and they stated some of the reasons. Another group developed useful tools that can be used with the Jupyter Notebook, such as automatic grading [22][23]. Pimentel et al. [24] thoroughly analyzed the quality and reproducibility of Jupyter Notebooks and identified some issues. Then they suggested a Jupyter Lab extension that identifies potential issues in notebooks and suggests modifications that improve their reproducibility.

The literature on using Jupyter Notebook is growing, and most are recent. Our thoughts in using Jupyter Notebook for the first time is that it needs careful preparation and competency in some tools such as programming, LaTeX, Markdown, HTML, etc. For people in engineering, this competency, to some extent, is available. Still, people from other specializations might need the help of competent people. At first, even for us, the task of developing the course in Jupyter Notebook was time-consuming and sometimes frustrating, especially when you develop while you teach. We, however, have to admit the results were compelling and rewarding.

The most time-consuming part is to program in LaTeX and format the Jupyter Notebook to deliver it seamlessly. The add-ins in the "nbextensions" are useful, like RISE for a slideshow, Table of Contents, Equation Auto Numbering, Spell Checker, etc. We used most of the extensions that made our presentation very effective. The feedback from the students was very positive and encouraging. We asked the students to use Jupyter Notebook for all class work, homework, and the term project. We admit that students faced some problems at the beginning of the course as they should learn how to use Jupyter and the associated tools. But they picked them quickly and became experts in them to the extent that they scribed the notebooks, improved their format, and added more code examples.

We believe that courses like this, which at first seem difficult to prepare, open the doors to more challenging and

novel experiences. The hands-on aspect that could be replicated and performed at a self-determined pace invites and allows students to have a more thorough experience. In a time like this, where universities are open to hybrid and online teaching, Jupyter has proven to be a handy tool.

V. CONCLUSIONS

In this innovative research practice, we presented our work of developing the modeling, simulation, and performance evaluation course that we teach to graduate students online. This move was due to the pandemic of COVID-19. But this experience could be further developed to continue in normal situations. The course may include more exercises with auto-grading. The format and appearance of the notebooks can be improved. The experience we gained and the feedback from the students will encourage us to continue developing the course. In future work, we plan to make the notebooks available on GitHub for the different communities to benefit from and receive more feedback and improvements.

REFERENCES

[1] T. Kluyver *et al.*, "Jupyter Notebooks-a publishing format for reproducible computational workflows.," in *In: Loizides F, Schmidt B, editors. Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016, vol. 2016, pp. 87–90.

[2] A. Cardoso, J. Leitão, and C. Teixeira, "Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses," in *Advances in Intelligent Systems and Computing*, 2019, vol. 917, pp. 227–236. doi: 10.1007/978-3-030-11935-5_22.

[3] V. Liubchenko and H. Parkhomenko, "The Involvement of Jupyter Notebooks as an Educational Tools: A Case Study," in *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, 2021, vol. 2, pp. 147–150. doi: 10.1109/CSIT52700.2021.9648674.

[4] A. Zúñiga-López and C. Avilés-Cruz, "Digital signal processing course on Jupyter–Python Notebook for electronics undergraduates," *Comput. Appl. Eng. Educ.*, vol. 28, no. 5, pp. 1045–1057, Sep. 2020, doi: 10.1002/cae.22277.

[5] A. Willis, P. Charlton, and T. Hirst, "Developing students' written communication skills with Jupyter notebooks," in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Feb. 2020, pp. 1089–1095. doi: 10.1145/3328778.3366927.

[6] J. Reades, "Teaching on Jupyter – using notebooks to accelerate learning and curriculum development," *Region*, vol. 7, no. 1, pp. 21–34, 2020, doi: 10.18335/region.v7i1.282.

[7] A. Suárez-García, E. Arce-Fariña, M. Álvarez Hernández, and M. Fernández-Gavilanes, "Teaching structural analysis theory with Jupyter Notebooks," *Comput. Appl. Eng. Educ.*, vol. 29, no. 5, pp. 1257–1266, Sep. 2021, doi: 10.1002/cae.22383.

[8] C. Tang, "Computer-aided Linear Algebra Course on Jupyter-Python Notebook for Engineering Undergraduates," in *Journal of Physics: Conference Series*, Feb. 2021, vol. 1815, no. 1. doi: 10.1088/1742-6596/1815/1/012004.

[9] M. Kuroki, "Using Python and Google Colab to teach undergraduate microeconomic theory," *Int. Rev. Econ. Educ.*, vol. 38, Nov. 2021, doi: 10.1016/j.iree.2021.100225.

[10] J. C. Domínguez *et al.*, "Teaching chemical engineering using Jupyter notebook: Problem generators and lecturing tools," *Educ. Chem. Eng.*, vol. 37, pp. 1–10, Oct. 2021, doi: 10.1016/j.ece.2021.06.004.

[11] R. Thomas and S. Cholia, "Interactive Supercomputing with Jupyter," *Comput. Sci. Eng.*, vol. 23, no. 2, pp. 93–98, Mar. 2021, doi: 10.1109/MCSE.2021.3059037.

[12] H. Fangohr, T. Kluyver, and M. Dipierro, "Jupyter in Computational Science," *Computing in Science and Engineering*, vol. 23, no. 2. IEEE Computer Society, pp. 5–6, Mar. 01, 2021. doi: 10.1109/MCSE.2021.3059494.

[13] J. Piazentin Ono, J. Freire, and C. T. Silva, "Interactive data visualization in Jupyter notebooks," *Comput. Sci. Eng.*, vol. 23, no. 2, pp. 99–106, Mar. 2021, doi: 10.1109/MCSE.2021.3052619.

[14] M. Müller and S. Rosenzweig, "PCP Notebooks: A Preparation Course for Python with a Focus on Signal Processing," *J. Open Source Educ.*, vol. 5, no. 47, p. 148, Jan. 2022, doi: 10.21105/jose.00148.

[15] T. Practice and R. Bloom, "A Revision of Bloom ' s Taxonomy : An Overview David R . Krathwohl," *ReVision*, vol. 41, no. 4, pp. 212–218, 2008.

[16] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation: Pearson New International Edition PDF eBook*. Pearson Higher Ed, 2013.

[17] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*, vol. 3. Mcgraw-hill New York, 2007.

[18] SimPy, "Discrete event simulation for Python." https://simpy.readthedocs.io/en/latest/ (accessed Apr. 21, 2022).

[19] P. Grogan, "Queuing System Discrete Event Simulation in Python." https://www.youtube.com/watch?v=oJyf8Q0KLRY (accessed Apr. 22, 2022).

[20] Apache Spark, "Spark Programming Model." https://spark.apache.org/ (accessed Apr. 22, 2022).

[21] J. W. Johnson, "Benefits and Pitfalls of Jupyter Notebooks in the Classroom," in *SIGITE 2020 - Proceedings of the 21st Annual Conference on Information Technology Education*, Oct. 2020, pp. 32–37. doi: 10.1145/3368308.3415397.

[22] C. D. González-Carrillo, F. Restrepo-Calle, J. J. Ramírez-Echeverry, and F. A. González, "Automatic grading tool for jupyter notebooks in artificial intelligence courses," *Sustain.*, vol. 13, no. 21, Nov. 2021, doi: 10.3390/su132112050.

[23] H. Manzoor, A. Naik, C. A. Shaffer, C. North, and S. H. Edwards, "Auto-grading Jupyter notebooks," in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Feb. 2020, pp. 1139–1144. doi: 10.1145/3328778.3366947.

[24] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "Understanding and improving the quality and reproducibility of Jupyter notebooks," *Empir. Softw. Eng.*, vol. 26, no. 4, Jul. 2021, doi: 10.1007/s10664-021-09961-9.