İSTANBUL TECHNICAL UNIVERSITY
Department of Computer Engineering
BLG443E – Discrete Event Simulation
2019 Spring Assignment 1 – Basic Monte Carlo + Food Inventory

ASSIGNMENT 1

Scenario

Read this article: http://www.businessinsider.com/whole-foods-employees-reveal-why-stores-are-facing-a-crisis-of-food-shortages-2018-1

Part 1

In the first part of this assignment you will gain familiarity with Jupyter Notebooks and Monte Carlo methods for simulation by solving a series of straightforward problems.

The problems are as follows:

- Generate samples from a discrete Bernoulli distribution over gender.
- Use that function to implement a Binomial distribution for number of females.
- Generate samples from a normal distribution over height.
- Use Monte Carlo Integration to calculate the expected value of this normal distribution over height.
- Generate samples from a joint distribution consisting of Bernoulli+normal distribution over height.
- Use Monte Carlo Integration to calculate the expected value of this Bernoulli+normal distribution.

Part 2

In the second part of this assignment you will attempt to model a single shop in a small-town chain like that described in the link. But it will be a simplification. **If you can think of ways of making the simulation more realistic we would look forward to seeing you do it!** (add it to the end of your submitted notebook)

You will examine only one product: cabbage. The shop keeps a small stock of cabbage on the shelf. Actually there is room for up to 40 cabbages! However, cabbages go rotten within 7-12 days so it would be expensive to keep so many cabbages stocked and to keep throwing them out. The shop's current scheme is to re-order cabbages whenever a cabbage is used. The current problem is that once a cabbage is ordered it takes 1-15 days to get it delivered. Frequently stock is empty and people coming for a cabbage (shoppers needing cabbage come every 0-3 days) need to be refused!

You may assume that all random variables with only ranges specified are uniformly distributed between those ranges.

Your task is to model this scenario using either Python or C++ (and you will submit your model in a working Jupyter Notebook), to run some simulations, and determine a number of cabbages to keep in stock such that:

- The expected number of people refused cabbages is less than 3 in 1000.
- The number of cabbages that expire is minimised.

Assignment Description

1. You will model the above scenario in either Python 3 or Cling/C++17 in a Jupyter Notebook. Step-by-step suggestions are given below.

2. For Part 1 you will write the code for and show the outputs in your notebook (including graphics) to solve the problems given.

3. For Part 2 you will:

- Run simulations for different numbers of cabbages kept in stock (each one of these is a configuration). For each configuration tried, run 10 simulations.

- Document your progress in the Jupyter notebook and you will present results of running the simulation, and make a recommendation for the number of cabbages to be kept in stock. Each configuration should have an associated histogram over the quantities of interest.

- At the end of the notebook, make also any suggestions for how the ordering process might be improved (you do not need to write a simulation for any improved processes).

Continue reading below for more details.

Submission Procedure

- Only electronic submissions through *Ninova* will be accepted.
- Late submissions or those submitted otherwise than according to instructions will not be accepted.
- Submit:
  - A **`ipynb`** file containing the final program (Python 3 or Cling/C++17) and its documentation.
- Submit in English.
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.

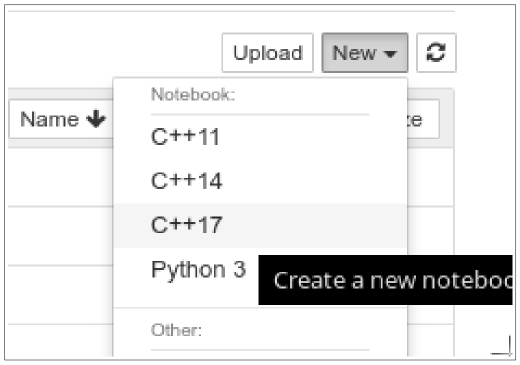
Step-by-step suggestions

If you are confident enough you may skip the below step-by-step suggestions and do your own thing but if you do that, do read these instructions anyway to ensure you have not missed something.

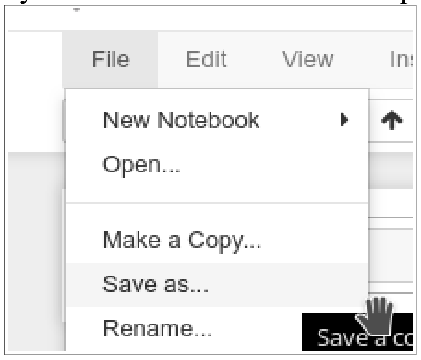
Setup

1. It is possible to use Jupyter Notebooks entirely in the cloud (e.g. AWS, Google Collab, Azure, Binder, CoCalc, Paperspace… none of which has been tested for this assignment). However, you may wish to install them on your own computer (this has been tested on (K/L/X)Ubuntu variants 16.04 and 18.04). Installing Jupyter Notebooks is best done with Miniconda (note that a full Anaconda install is not recommended because of some incompatibilities with the experimental C++ interpreter) on all platforms (including Linux and Windows – support may only be available for Linux). The below commands install Jupyter Notebooks on Ubuntu variants 16.04 and 18.04 with both Python 3 and C++17 kernels, assuming that Miniconda is already installed and in the system path:
```
conda create -n blg443 # create a conda environment for our course
source activate blg443 # activate that
# The next two lines install the C++ interpreter for Jupyter
conda install xeus-cling xplot -c QuantStack -c conda-forge
conda install widgetsnbextension bqplot -c conda-forge
# The next line installs Jupyter with comes with a Python 3 kernel
conda install jupyter matplotlib
```

2. Once you have installed Jupyter you can start a notebook (at least on Linux) using the command:
```
jupyter notebook
```
You can start a notebook in the language of your choice by clicking on New and selecting the appropriate language (either Python 3 or C++17 for this course):



3. You can save your notebook at any time for submission using the File.. Save As.. functionality (if you are working in the cloud you may need to use the Download option):



4. For more information and example Jupyter notebooks for **Python** see the following examples (see the next step for C++):
   New to Python?
   - https://www.learnpython.org/

   Want to see the basics of working with Jupyter and Python? See:
   - https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb
   - http://opentechschool.github.io/python-data-intro/core/notebook.html

   Want to see what Jupyter+Python can offer in diagrams?
   - https://nbviewer.jupyter.org/github/ipython/ipython/blob/1.x/examples/notebooks/Part%203%20-%20Plotting%20with%20Matplotlib.ipynb
   - https://nbviewer.jupyter.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb

   Adding documentation to Jupyter notebooks:
   - https://nbviewer.jupyter.org/github/ipython/ipython/blob/1.x/examples/notebooks/Part%204%20-%20Markdown%20Cells.ipynb

5. For more information and example Jupyter notebooks for **C++** see the following examples (see the previous step for Python):
   Want to see the basics of working with Jupyter and C++? See:
   - https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb
   - https://mybinder.org/v2/gh/QuantStack/xeus-cling/stable?filepath=notebooks/xcpp.ipynb

   Want to see what Jupyter and C++ can offer in diagrams? See in particular the examples here:
   - https://mybinder.org/v2/gh/QuantStack/xplot/stable?filepath=notebooks

   Adding documentation to Jupyter notebooks:

- https://nbviewer.jupyter.org/github/ipython/ipython/blob/1.x/examples/notebooks/Part%204%20-%20Markdown%20Cells.ipynb

⚡ **Warning**: The Jupyter integration of this C++ interpreter (Cling) are experimental.

🎓 **Opportunity**: If you can find out how to do things that noone else knows and share them with the class via Ninova in good time (for example, how to inspect variables using Jupyter, how to encapsulate plotting in a function, etc.) you can be in to earn some extra marks – ensure you link to the relevant discussion in your submitted notebook.

Part 1

1. Write a function called *gender* that takes no arguments and randomly samples a gender from a Bernoulli distribution with a probability of 0.5 for Male and 0.5 for Female and returns either "Male" or "Female" accordingly (or some representative number, object, enum, etc.). You can either use the standard random number generation routines (Python: https://docs.python.org/3/library/random.html C++: https://en.cppreference.com/w/cpp/numeric/random/rand or https://en.cppreference.com/w/cpp/numeric/random) or special purpose routines from boost/random (for C++) or scipy.stats.random (for Python). Apart from these, use of external libraries is not allowed.

The "coin flip" Jupyter notebooks shared from class should contain sufficient examples to help you with this, including charting capabilities; there is a Python and C++17 version.

2. Write a function that wraps the first function, takes as an argument number of trials as *n* and returns the number of Females. Call it *num_females*. So the signature of the function will be *num_females(n)* → *int.* Note: the new function now implements a Binomial distribution with a probability of 0.5. Test the function by calling it 20 times and making a bar chart of frequency vs gender. You will have to calculate the number of males as *n* minus the number of females.

3. Write a function called *female_height* that samples from a normal distribution with mean 158.9 and standard deviation 0.6 and returns a single floating point number representing the height of a randomly chosen female in Turkey. So its signature will be *female_height()→ float*. Similarly, write a function called *male_height* sampling from a normal distribution average 174.1 and standard deviation 0.7 with signature *male_height()→ float*. See for Python the function **random.gauss** and for C++17 the function **normal_distribution** (note that the C++ version is slightly more complex in that one needs to instantiate both the distribution and the random number generator before one can generate random numbers). Test these by calling each of your functions 20 times and making two histograms of frequency vs. height (one for each gender).

4. Write a function called *expected_height_by_gender* that takes as a parameter the gender, a number of samples parameter *n* (e.g. *n*=20), and runs the functions written in the previous step *n* times to calculate an estimate of the expected height of that gender (as the average of the sampled heights). Therefore the function signature will be *expected_expected_height_by_gender(gender, n)→ float*. Congratulations! You have now done Monte Carlo integration! Compare the histogram of running this function 20 times for the female gender vs. the histogram of running the function *female_height* 20 times. Is there a difference in the two histograms? Should there be theoretically? What if you reran

this experiment or increased the number of times the function was run?

5. Write a function called *expected_height* which takes as a parameter a number of samples as a parameter (e.g. *n*=20), calls the function *gender n* times, and for each call, calls either *male_height* or *female_height*, to calculate a height of a person (depending on what the function *gender* returned). The function should return the average height over all these calls so that it returns an estimate of the expected height of a person in Turkey regardless of gender. Therefore the function signature will be *expected_height(n)→ float*. Additionally, plot the histogram of frequency vs. height by calling this function 10000 times with *n* set to 1. What happens to this histogram if n is set to 2000?

Part 2

1. Write a function that calculates and returns the inter-arrival time of shoppers as per the scenario (note that the scenario states that customers come every 0-3 days and that random variables with only ranges specified should be treated as uniformly distributed). It should be able to return not just whole numbers but in-between numbers (e.g. floats).

2. Check that the more samples you collect from this function the more resulting histogram of inter-arrival times looks like a uniform distribution (it should converge on the generating distribution as the number of samples gets high). Ensure that the histogram is visible in your submitted notebook.

3. Write, test, and visualise similar functions for delivery time and time-to-rot of cabbages.

4. Create a state variables to keep track of the number of cabbages in storage and write a simulation of the scenario, simulating cabbages arriving but without simulating the tendency of cabbages to go rotten and without simulating customers arriving.

5. Now make the simulation more complex by simulating customers buying the the cabbages, with the simplifying assumption that reordered cabbages arrive immediately and cabbages never go rotten. **[leave the previous simplified simulation in your notebook to document your process]**

6. Create a state variable to keep track of a list of when reordered cabbages should arrive and augment the simulation you previously created to simulate the ordering of cabbages. Note that although in future assignments you will have to write this in a general way, in this assignment you only need to attempt to solve the problem in front of you. **[leave the previous simplified simulation in your notebook to document your process]**

7. Create a state variable to keep track of a list of when cabbages are going to go rotten and augment the simulation you previously created to simulate cabbages going rotten. **[leave the previous simplified simulation in your notebook to document your process]**

8. Rewrite the simulation as a *function* and try running it a few times. **[leave the previous simplified simulation in your notebook to document your process]**

9. Make your simulation function return *number of expired cabbage*, and *number of people refused cabbage* (in modern C++ you may use tuple and tie functionality for returning multiple values if you wish – with Python returning a tuple is customary – but you may also prefer tailored structs/classes). Run your simulation for a number of trials for each of a particular number of cabbages on the shelf.

For each number of cabbages show a histogram over number of expired cabbages and number of people refused cabbage. Try to make a conclusion about the task specified in the first page of this assignment – the optimal number of cabbages to be kept in stock according to the given criteria. **[leave the previous simplified simulation in your notebook to document your process]**

10. Offer some discussion, ensure your notebook is well documented and everything you do justified. It will help the marker to understand and grade your work if you put question numbers in the document also.

11. **BONUS PART** (you do not need to attempt this to achieve 100% in this assignment): Write code to automatically answer the key task as specified in the first page of this assignment – the optimal number of cabbages to be kept in stock according to the given criteria. The code will call your simulation function.

12. **BONUS PART** (you do not need to attempt this to achieve 100% in this assignment): Now you are going to repeat your experiments but using a more realistic distribution of customer arrival times. First we will examine the relevant probability distribution. Considering that the uniform distribution assumed above over inter-arrival times of customers (0-3 days), the average customer arrival time is 1.5 days. You are going to use the **exponential distribution** to generate inter-arrival times with an average of 1.5 days (the exponential distribution is the best distribution to use for inter-arrival times without having more knowledge). The relevant examples are here:
    - Python: `random.expovariate` here: https://docs.python.org/3/library/random.html#real-valued-distributions
    - C++: https://en.cppreference.com/w/cpp/numeric/random/exponential_distribution
      (if you wish you may use boost/random (C++) or the scipy stats random (Python) for more advanced capabilities, but you will have to install the related libraries)
      Now, plot histograms of samples generated from this distribution as the number of samples increases from e.g. 10 to e.g. 100000. Does it seem to converge on the theoretical exponential distribution probability density function? Is the average of the samples converging to 1.5?

    Now rerun your simulations with the customer inter-arrival time governed by the exponential distribution and see if your conclusions must change.

13. **BONUS PART** (you do not need to attempt this to achieve 100% in this assignment): If you would like to do extra things such as trying a simulation framework, running a genetic algorithm, etc. do this at the end of the notebook under a heading “Advanced Explorations”. If necessary repeat your experiments with the new functionality.

Continue reading below for more details.

Grading Policy

Bonus points may be obtained by doing additional interesting modelling or simulation, but do point out your contributions in the notebook at the end under the heading “advanced explorations”.

Code (70%):

- Correctness of answers to part 1 problems.
- Correctness of simulation logic ✕ Completeness of simulation logic.
- Good engineering practice (extensible, modular, etc.)

Results and documentation (30%):

- Correctness of statistics and figures (histogram & report).
    - Don’t forget to label axes etc.
- Conclusions appropriate to data & relevant suggestions.