Guidelines: This is to be individual work. You may consult with others in this class, but subject to the *empty-hands* policy as discussed in the syllabus — your work must be your own. In your write-up, clearly document/cite:

- Any persons with whom you consulted.
- Any sources (other than course slides or the APIs for Python and R) you consulted in completion of your work.

All solutions, with the exception of source code and data files, must be typeset and submitted as a single self-contained PDF. Include appropriate figures, data tables, etc. to support your discussion and conclusions. Strive to create a well-written, self-contained, research-article-like solution set. Include all source code with your solution set.

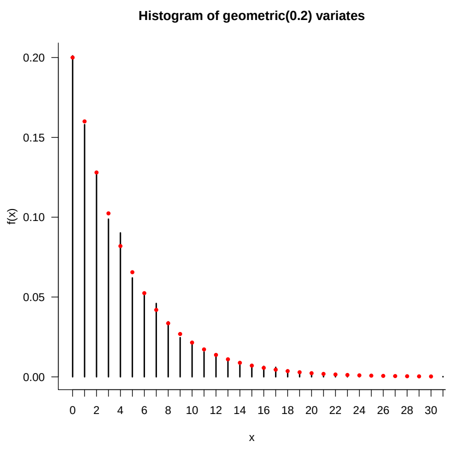
Problem 1: Complete your Python class implementation of the RNG class that acts as a wrapper library around numpy's MT19937 Mersenne Twister generator, providing for streams capability. Specifically, you must include class-level methods for:

- `random(cls, which_stream: Stream) -> numpy.float64`
- `randint(cls, a: int, b: int, which_stream: Stream) -> numpy.int64`
- `uniform(cls, a: float, b: float, which_stream: Stream) -> numpy.float64`
- `exponential(cls, mu: float, which_stream: Stream) -> numpy.float64`
- `geometric(cls, p: float, which_stream: Stream) -> numpy.int64`
- `gamma(cls, shape: float, scale: float, which_stream: Stream) -> numpy.float64`

For each method, make sure to initialize the streams of generators if not already initialized. For each method, make sure to also include a thorough Python docstring for your method clearly describing the purpose, each parameter, and the returned value. Use Google's docstring style guidelines as a guide.

For each method, benchmark its output against the corresponding variate generator from R. To do so, generate *many* variates from your generator, saving to a file. Then in R, read in those data, and create a histogram (using `hist` for continuous distributions, and a combination of `table` and `plot` for discrete distributions) using your variates. Then superimpose the theoretical values as indicated by R, using the so-called "d*" functions. Below is an example for `geometric`, as we developed and discussed in class.

```
% python rng.py > geom.txt
% R
...
> values = scan("geom.txt")
Read 10000 items
> geom_hist = table(values) / length(values)
> pdf("geom_hist.pdf")
> plot(geom_hist, xlim = c(0,30), type = "h",
    xlab = "x", ylab = "f(x)", bty = "l", las = 1)
> points(0:30, dgeom(0:30, 0.2), pch = 20, col = "red")
> title("Histogram of geometric(0.2) variates")
> dev.off()
```



Problem 2: Use event-driven (i.e., next-event) simulation to simulate the performance of a single-server service node with a FIFO queue discipline and a *finite* service node capacity (i.e., finite queue length). For this work, you must extend your `ssq.py` model developed using simulus, and you must use the wrapper library developed in Problem 1. Use a different stream for each of the three stochastic components below (which means you will need to modify the `Stream` class in your RNG library).

Assume that:

- the arrival process has *exponential*(λ) interarrival times with arrival rate $\lambda = 0.5$;
- the service process[1] (per customer) consists of a number of service tasks equal to 1 + η, where η is a *geometric*(p) variate[2] with $p = 0.1$; and and
- the time for each of the service tasks within a service process is, independently for each task, a *uniform*(0.1, 0.2) random variate.

Your experimentation and corresponding presentation/discussion should address the following questions:

(a) Based on approximately 1 000 000 processed customers, construct a table of the estimated steady-state probability of rejection (i.e., a customer arriving to find the system at capacity) and the estimated average sojourn time for service node capacities of 1, 2, 3, 4, 5, and 6. (Note that a service node capacity of 1 corresponds to a server only with no queue; capacity 2 corresponds to a server with maximum queue length of 1; etc.)

(b) Construct a similar table if the time per service task is changed to be *uniform*(0.1, 0.3).

(c) Using R (i.e., independent of your `ssq.py` simulation), provide appropriate histograms to meaningfully compare the distribution of service times generated by the two different service models. Discuss the two models and comment on their appropriateness for a single-server queuing system.

(d) Comment on how the probability of rejection and the average sojourn time depends on the underlying service process.

(e) Discuss what you did to convince yourself that your results are correct. Provide as many consistency checks as you can think of. (There are plenty available here, e.g., $\bar{n} = \bar{q} + \bar{x}$, where those type-averaged statistics are collected and computed independently.) In addition to the more obvious ones, what should you expect the probability of rejection to be when the capacity is 1? 2?)

Problem 3: A test is compiled by selecting 12 different questions, at random and without replacement, from a well-publicized list of 120 questions. After studying this list you are able to classify all 120 questions into two classes, I and II. Class I questions are those about which you feel confident; the remaining questions define class II. Assume that your grade probability, conditioned on the class of the problems, is

	A	B	C	D	F
class I	0.6	0.3	0.1	0.0	0.0
class II	0.0	0.1	0.4	0.4	0.1

In other words, if a question is class I, you have probability 0.6 of earning an A, probability 0.3 of earning a B, and so forth. Each test question is graded on an A = 4, B = 3, C = 2, D = 1, F = 0 scale and a score of 36 or better is required to pass the test. (a) If there are 90 class I questions in the list, use Monte Carlo simulation to generate a histogram of scores. (b) Based on this histogram what is the probability that you will pass the test? (c) Provide a sense of the uncertainty of your result.

Submitting: Submit your write-up (PDF), all source code, and any additional supporting files on Lyceum.

[1] See supplementary §3.1 reading from Leemis and Park, specifically Example 3.1.4, provided on Lyceum as a PDF.

[2] The mean value of η should be 9.