ISyE 6644 — Spring 2021 — Mini-Project 2!

Timeline:

- **M 3/29/21**: Suggested topics and ground rules posted.
- **F 4/2/21**: Select topic and group members (handled as an assignment in Canvas)
- **M 5/1/21, 11:59 pm**: Turn in Mini-Project 2.

Some ground rules:

- Pick *one* project that you find interesting. The projects are all intended to be a little open-ended, but do not pick a project that will kill you time-wise!
- You are allowed to substitute any other interesting topic of your own choice as long as you run it by us first (scroll to the end for some details).
- You are allowed to work in groups that are reasonably sized, though you will have to form the groups on your own. (Suggested group sizes are given in the text of each problem.)
- The topics range from computer-intensive to more statistically oriented. When computer work is necessary, you are allowed to use whatever computer languages you'd like, e.g., R, Minitab, Matlab, Python, even Excel.
- Good luck, and have fun!

Some Suggested Ideas for Mini-Project 2 for ISyE 6644:

1. You can choose to continue work on your Mini-Project 1 if there is significant content that you can undertake that's above-and-beyond what you accomplished in Mini-Project 1.

Applications-Oriented Problems

2. (2 members) Model, simulate, and analyze a reasonably large-scale real-life *conveyor* system.

3. (3 members) Go to your favorite non-trivial *elevator* system and model it. This is not as easy as you might anticipate, so do your best. In particular, collect some real data, and answer as many interesting questions as you can think of. For instance, what rules govern elevator movement? How long do people typically have to wait? Can you think of elevator movement rules that might reduce the average waiting times? Etc., etc. Justify everything you conclude by conducting careful simulations and output analysis. This could be a fun project!

4. (2 members) Model and simulate a reasonably large-scale *job shop*. This might be a good project if you happen to have a real-life manufacturing job. Let's see how creative you can be.

5. (2 members) Model, simulate, and analyze a reasonably large-scale real-life *warehouse*. Let's see how creative you can be.

6. (2–4 members) Run a simulation having direct applicability for *your job*. Some people in the class have access to interesting "real-world" problems, either through their jobs, or perhaps as an assignment in another course here at Georgia Tech. If you'd like to do a full-scale simulation analysis on a particular nontrivial simulation (no M/M/1 queues, please!), then this project may be the one for you. I would expect some input analysis (i.e., choosing input distributions, planning run-length, etc.), some modeling and programming, and some output analysis (e.g., confidence intervals). This would be a good project for a group with members having diverse interests.

Language- and Modeling-Oriented Problems

7. (2 members) Learn another higher-level simulation language like AutoMod, ProModel, AnyLogic, or any of the available Python- or Java-based freeware systems. In your project write-up, give your comments and opinions on the language, provide an easy user's guide, and demonstrate how the language works on one or two example models.

8. (2 members) Carefully compare some other simulation language to Arena. In your project write-up, give your comments and opinions on the languages, provide an easy user's guide, and demonstrate how the languages work on one or two example models.

9. (2 members) Learn about data encryption using tricks from random number generation. There is a wide literature on this topic ranging from easy stuff to start-of-the-art techniques such as PGP ("pretty good privacy").

Programming-Oriented Problems

10. (2–3 members) *Pandemic Flu Spread.* One of the choices for Mini-Project 1 was a (trivial) simulation of pandemic flu spread in a classroom. For Mini-Project 2, I'd like you to think about a bigger-and-better simulation involving a larger population. Here's a potential scenario (there are many other interesting ones — feel free to be imaginative):

    - Some infectious people enter a population of susceptibles, and some of the susceptibles become infected.
    - There is a short period of a couple of days before a newly-infected person in turn becomes infectious.
    - When a person recovers (or dies), the person is not again susceptible.
    - Infectiousness or death can be mitigated by masking, social distancing, etc.
    - Infectiousness or death can be mitigated by vaccination. Vaccines can be delivered in one or two doses. But there could be supply chain issues.
    - Even if a vaccine requires two doses, the vaccine nevertheless provides partial immunization after even only one dose. Might you immediately give everyone only one dose instead of two, and hope that the supply chain catches up so that you can "eventually" give everyone two doses?

    To determine whether a particular strategy is any good, you probably ought to consider the number of people who eventually get infected (or die), the length of the epidemic, etc.

11. (1 member) Implement the *Tausworthe pseudo-random number generator* (as described in Module 6 of our notes). Perform a decent number of statistical tests on the generator to see that it gives PRN's that are approximately i.i.d. Uniform(0,1). Plot adjacent PRN's (U_i, U_{i+1}), $i = 1, 2, \ldots$, on the unit square to see if there are any patterns. Then, just for the heck of it, generate a few Nor(0,1) deviates (any way you want) using Unif(0,1)'s from your Tausworthe generator.

12. (1–2 members) Test some Unif(0,1) random number generators. There are a lot of uniform random number generators around. How do we know if a particular generator is doing a good job? In this project, you should test various generators to see if they are actually producing numbers that are approximately independent, identically distributed uniforms. The tests you should use are formal hypothesis tests such as χ^2 goodness-of-fit tests, the von Neumann test for independence, runs tests, etc. You should also comment on the run-time efficiency of the generators; i.e., which generators yield the most random numbers per unit time? This could be

a really good project for a group interested in everything from statistics to graphical analysis of data.

13. (1–2 members) Make me a nice library of *random variate generation* routines. You can use your favorite high-level language like C++, Java, Python, Matlab, or even Excel. Include in your library routines for generating random variates from all of the usual discrete and continuous distributions, e.g., Bern(p), Geom(p), Exp(λ), Normal(μ, σ^2), Gamma(α, β), Weibull(α, β), etc., etc. (Just one routine per distribution is fine.) Include in your write-up an easy user's guide, complete source code, and some appropriate examples.

14. (2 members) Make me a nice library of routines for *fitting input distributions*. The input to a fitting routine for a particular distribution should be a bunch of observations; the output should be the maximum likelihood estimate. You can use your favorite high-level language like C++, Java, Python, Matlab, or even Excel. Include in your library maximum likelihood estimation routines for fitting all of the usual discrete and continuous distributions, e.g., Bern(p), Geom(p), Exp(λ), Normal(μ, σ^2), Gamma(α, β), Weibull(α, β), etc., etc. Beware! The Weibull takes a little work (so does the gamma, for that matter). Luckily, everything is outlined very clearly in my notes and/or Law (2015). Include in your write-up an easy user's guide, complete source code, and some appropriate examples. The examples should be along the following illustrative lines: Generate some example data; attempt to fit the data to various distributions; conduct χ^2 goodness-of-fit tests to show how well the fits do. For instance, if you generate some example Weibull data, a Weibull fit ought to do better than an exponential fit.

15. (2 members) Output analysis. Implement the method of *batch means* in your favorite high-level language like C++, Java, Python, Matlab, etc. The input to your program should be a bunch of observations, the number of batches you want, and a confidence level $1-\alpha$; the output should be a $100(1-\alpha)\%$ confidence interval for the steady-state expected value of the observations. Include in your write-up an easy user's guide, complete source code, and some appropriate examples. In particular, generate some waiting time data from an $M/M/1$ queueing system; then report the batch means confidence interval for the steady-state expected waiting time, w_Q. Note that you can actually *look up* w_Q for the $M/M/1$ — so you can tell whether or not your confidence interval covers w_Q. Does it? Now do 100 independent runs to obtain 100 confidence intervals. How often do your confidence intervals cover w_Q? What is the average length of your confidence intervals? What happens if you vary the number of batches (while keeping the run length constant). Be creative!

Theory-Oriented Problems

16. (2 members) Find the exact distributions of the number of *runs* from a sample $U_1, U_2, \ldots, U_n$ of n i.i.d. Unif(0,1) random numbers. This is easy to do for $n = 3$, but then it starts getting nasty, as we discovered in one of our homeworks. Once you get to a certain point, it would be OK for you to use large simulation runs to approximate the distributions. If you have a little extra time, there are different kinds of runs you can look at: runs up-and-down (like we did in class), runs above-and-below-the-mean, etc.

17. (2 members) Explore the efficiency of different random variate generation techniques. Some examples (there are many):

    - Generate a Bin(n, p) from an Inverse Transform c.d.f. table or by adding up Bern(p)'s?
    - Generate a Geom(p) via the simple Inverse Transform equation or by sample Bern(p)'s until you get a success?
    - Generate a Pois(λ) via a table or Acceptance-Rejection?
    - What's the fastest way to generate a Nor(0,1)?
    - Etc., etc., etc.!

18. (1–2 members) Read and comment on Bruce Schmeiser's fundamental *output analysis* paper, "Batch Size Effects in the Analysis of Simulation Output," *Operations Research*, Volume 30, 556–568 (1982). Bruce's paper analytically studies the behavior of the batch means method. Now refer to Chapter 9 of Law (2015), and in particular, the relevant tables there that deal with the *empirical* behavior of the batch means method for output analysis. Compare Bruce's results with Law's.

19. (1–2 members) Another *output analysis* problem (a bit mathematical). Take a look at and comment on the following paper, which calculates the *exact* expected values of certain estimators for the mean of a steady-state simulation output process: Aktaran, et al. (2007), "Exact Expected Values of Variance Estimators for Simulation," *Naval Research Logistics*, Volume 54, 397–410.

20. (2 members) Prepare a tutorial on *variance reduction methods*. You should comment on *at least* the following techniques from Module 10 of our notes: common random numbers, antithetic variates, and control variates. You can also refer to Law (2015) for additional techniques if you have sufficient time. Cook up a few (possibly fake) examples involving Monte Carlo integration or other applications where you might be able to use some of these techniques.

21. (3 members) *Ranking and selection* procedures are designed to help you determine which one of a number of competitors is the "best." For example, we might be interested in selecting that one of k competing queueing systems having the smallest expected time-in-system. If you are interested in such problems, read the second half of Module 10 from our notes and/or Chapter 10 of Law (2015). Give me a nice tutorial on some of the procedures described in those references. Then implement at least one of those procedures in your favorite high-level language like C++, Java, Python, Matlab, etc. Include in your write-up an easy user's guide, complete source code, and appropriate examples. This is very useful stuff!

And...

22. **Any Other Problem That You Choose** — as long as you can convince us. Succinctly describe the problem and why you think it would be cool for this class. Needs to be nontrivial, useful, and interesting.