ISyE 6644 — Fall 2020 — Projects!

Timeline:

- **Su 9/20/20**: Suggested topics and ground rules posted.
- **M 9/28/20**: Select topic and group members (handled as an assignment in Canvas)
- **Th 10/22/20**: Progress report due (rubric TBA).
- **Th 12/3/20**: Project due (rubric TBA).
- **Th 12/10/20**: Peer evaluations due (each student with be assigned 2 evaluations, rubric TBA).

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; of course, Arena is recommended for simulation modeling tasks.
- Good luck, and have fun!

Some Suggested Projects for ISyE 6644:

Applications-Oriented Problems

1. (2 group members) The African Tanker Problem. If you like tricky little problems, then this is the one for you! This famous exercise concerns the simulationof a somewhat complicated port, and is described in Law (2015) (Chapter 2 exercises). You should model this and conduct a little more output analysis than is asked for in the text of the problem.

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) Go to some intersections and model a couple of *traffic lights*. Collect some real data. Try to answer as many interesting questions as you can think of. For instance, what would the effect of synchronization be? Or, how do cell phones affect driver behavior? Justify everything you conclude by conducting careful output analysis. This could be a great project!

5. (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.

6. (2 members) Model, simulate, and analyze a reasonably large-scale real-life *warehouse*. Let's see how creative you can be.

7. (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.

8. (2 members) *Pandemic Flu Spread.* Consider a classroom of 21 elementary school kids. 20 of the kids are healthy (and susceptible to flu) on Day 1. Tommy (the 21st kid) walks in with the flu and starts interacting with his potential victims. To keep things simple, let's suppose that Tommy comes to school every day (whether or not he's sick) and will be infectious for 3 days. Thus, there are 3 chances for Tommy to infect the other kids — Days 1, 2, and 3. Suppose that the probability that he infects any individual susceptible kid on any of the three days is $p = 0.02$;

and suppose that all kids and days are independent (so that you have i.i.d. Bern(p) trials).

If a kid gets infected by Tommy, he will then become infectious for 3 days as well, starting on the next day.

(a) What is the distribution of the number of kids that Tommy infects on Day 1?

(b) What is the expected number of kids that Tommy infects on Day 1?

(c) What is the expected number of kids that are infected by Day 2 (you can count Tommy if you want)?

(d) Simulate the number of kids that are infected on Days 1,2,.... Do this many times. What are the (estimated) expected numbers of kids that are infected by Day i, $i = 1, 2, \ldots$? Produce a histogram detailing how long the "epidemic" will last.

Language- and Modeling-Oriented Problems

9. (3 members) Learn and implement the self-contained simulation language SIMLIB from Chapter 2 of the Law (2015) text. Although SIMLIB is not the most-efficient simulation language around, it will really teach you how an actual simulation language is written. In your project write-up, give your comments and opinions on the language, provide an easy user's guide, and demonstrate how SIMLIB works on one or two example models. Note: I'll allow you to replicate the functionality of SIMLIB using any language such as C++, Python, etc.

10. (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.

11. (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.

12. (2 members) A different modeling paradigm. Read and comment on Lee Schruben's fundamental paper, "Simulation Modeling with Event Graphs," *Communications of the ACM*, Volume 26, pp. 957–963 (1983). Then do the same for the Sargent (1988)

and Som and Sargent (1989) articles cited in the References section of the Law (2015) text. (The latter two papers extend and refine the event graphs technique, and they provide lots of nice examples.)

13. (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

14. (1 member) Let's *gamble*! Implement a blackjack simulation; in fact, simulate a few reasonable strategies. Statistically analyze these strategies by playing lots of independent blackjack games. Then determine which choice maximizes your profit. (Assume that the House plays by the usual rules.) Include all code and commentary. Be creative!

15. (1–2 members) Let's gamble some more! Same thing as in Problem 14, except for solitaire. (This isn't easy, but it's fun.)

16. (1–2 members) Another game. There are two players, A and B. At the beginning of the game, each starts with 4 coins, and there are 2 coins in the pot. A goes first, then B, then A,.... During a particular player's turn, the player tosses a 6-sided die. If the player rolls a:

    - 1, then the player does nothing.
    - 2: then the player takes all coins in the pot.
    - 3: then the player takes half of the coins in the pot (rounded down).
    - 4,5,6: then the player puts a coin in the pot.

    A player loses (and the game is over) if they are unable to perform the task (i.e., if they have 0 coins and need to place one in the pot). We define a cycle as A and then B completing their turns. The exception is if a player goes out; that is the final cycle (but it still counts as the last cycle). We are trying to determine the expected number (and maybe even the distribution) of cycles the game will last for. I'm guessing that you can use"first-step" analysis to get the expected value. Simulation seems the easiest thing to do to get the entire distribution.

17. (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.

18. (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.

19. (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.

20. (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 fake 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 fake Weibull data, a Weibull fit ought to do better than an exponential fit.

21. (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

22. (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.

23. (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.

24. (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.

25. (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.

26. (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...

27. **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 unteresting.