STAT 494
L. Lyman
Spring 2023

Assignment 1

Due: **Wednesday, February 15**, by 11:59 pm CST
Submitted in class, or emailed to the instructor

> *A turkey is fed for 354 days — and every day reports to its statistics department that the human race cares about its welfare with "increased statistical significance." On the 365th day, the turkey has a surprise.* — Nassim Taleb [paraphrased]

Instructions (Please read carefully before beginning)

(i) You may submit your assignment in class or via email to llyman@macalester.edu. (ii) The default programming language for this course is Python with Jupyter Notebooks. However, you are free to use the language of your choice (MATLAB, C++, etc.). (iii) Export your answers (plots, numerical values, etc.), along with all code used to generate those answers, as a PDF. (iv) Use best coding practices when possible. Extract repeated code with functions, use descriptive variable names, and add informative comments. (v) You are **encouraged to work with others**, but each person is responsible for submitting an individual writeup. (vi) You are required to do:

- Exercise 1
- Either Exercise 2 or 3.

If you complete all exercises, you earn the highly-coveted title of **UQ Master**.

---

1. (50 pts) **Nagel-Schreckenberg traffic modeling.**

   Recall that our traffic flow model has the following:

   - A single lane
   - M chunks of road, labeled $0, \ldots, M-1$ (with $M > 1$)
   - N total vehicles ($N \le M$)
   - Discrete time steps $t_0, t_1, \ldots$ of identical length

   We can picture the road via Figure 1.

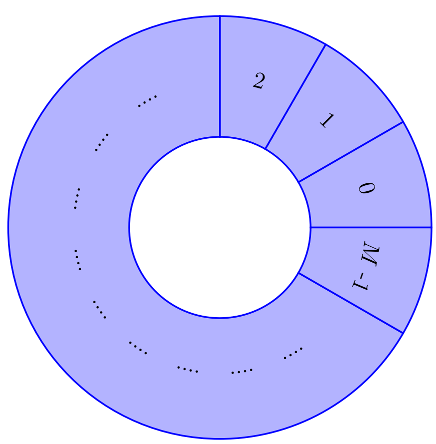

   Figure 1: A circular, one-way, single lane road with M spots, labeled $0, \ldots, M-1$.

   Further, we assume the following.

- Each vehicle occupies exactly one chunk at a given time step t
- A car with velocity $v \geq 0$ moves v spots forward in one time step
- There is some speed limit $v_{\max} > 0$

Possible pseudocode for the whole procedure could look like the following (Algorithm 1) for a *single* time step. For indexing convenience, it is likely a good idea to label your cars such that car $i+1$ is in front of car i for all $i \in \{0, \ldots, N-2\}$.

Algorithm 1: Nagel-Schreckenberg

input

- M, N, fixed probability $0 < p < 1$

input (or initialize)

- $\boldsymbol{x}$ `//` $N \times 1$ `vector of car positions`
- $\boldsymbol{v}$ `//` $N \times 1$ `vector of car velocities`
- $\boldsymbol{d}$ `//` $N \times 1$ `vector of distances between cars, where` $d_i = |x_{i+1 \,(\mathrm{mod}\, N)} - x_i|$
  `// for` $i = 0, \ldots, N-1$

---

for $i = 1, \ldots, N$ **do**
| $(x_i, v_i) \leftarrow$ **NS-update**(x_i, v_i, d_i, p)
end
update $\boldsymbol{d}$ based on new $\boldsymbol{x}, \boldsymbol{v}$
return $(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{v})$

The following helper routine (Algorithm 2) updates the ith car. If you prefer, Algorithm 2 could of course be modified to update all of the cars at once by taking in vectors $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{d}$ of positions, velocities, and distances.

Algorithm 2: NS-update

input

- The position $x \in \{0, \ldots, M-1\}$ and velocity v of the vehicle
- The distance d between this vehicle and the car immediately in front of it
- Probability $0 < p < 1$ of randomly slowing down

`// Drivers are eager to move forward, so you speed up by 1 unit if you can. If you're already speeding, you (begrudgingly) slow down to the speed limit`

- **set** $v \leftarrow \min(v+1, v_{\max})$ (1)

`// You're worried about what would happen if the car in front of you suddenly stopped. If you'll crash into the car in front within one time step (i.e. if` $v \geq d$`), then slow down by 1 unit. Otherwise, continue at your current speed`

- **set** $v \leftarrow \min(d-1, v_{\max})$ (2)

`// This car is a metal death trap! With probability` p`, slow down by 1 unit (if you're not already stopped)`

- With probability p, **set** $v \leftarrow \max(0, v-1)$ (3)

`// Based on your current speed, update the position`

- **set** $x \leftarrow x + v \mod M$ (4)

return x, v

(a) (12 pts) Create a Monte Carlo simulation of the Nagel-Schreckenberg traffic model. Let the number of spaces $M = 1,000$ and the number of cars $N = 50$. Take $v_{\max} = 35$ and $p = \frac{1}{3}$. Initialize your simulation via the following.

- The cars have initial positions x_i determined by sampling N values from $\{0, \ldots, M-1\}$ uniformly without replacement.
- The N cars have initial velocity $v_i = 0$.
- There is a "burn-in" period of 2,500 time steps. That is, let your simulation run for 2,500 time steps before "starting the clock" to produce your figure for (b).

  Note. This terminology is **different** than what I described in class, in which the "burn-in" period was the time for which the simulation is run.

(b) (13 pts) Make a flow trace image analogous to Figure 2 for 1,000 time steps after the burn-in period. You should get something quite similar. Note that you might prefer to make $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{d}$ into matrices to keep track of your data per time step.

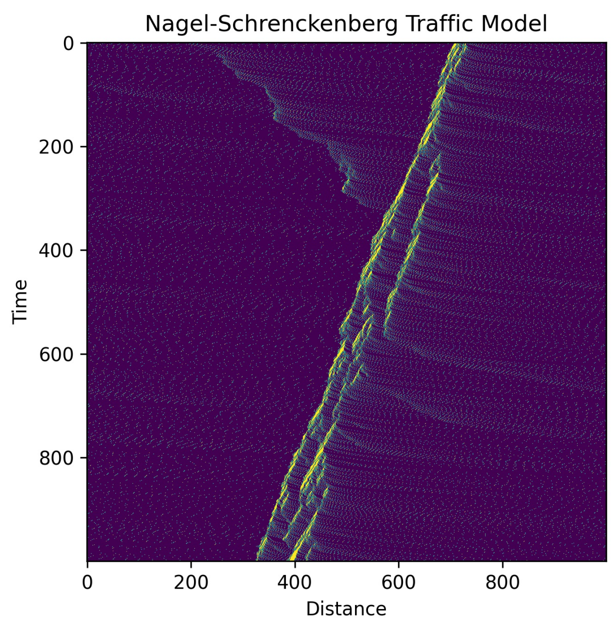

Figure 2: Flow trace image for NS traffic modeling.

The dark bands represent the traffic jams.

(c) (2 pts) Since the road is one-way, any update formulas must preserve that all $v_i \geq 0$. Add a line in your code to verify this is the case.

(d) (5 pts) Ben initiated an important discussion in class about whether certain velocity-updating steps should be permuted. That is, even if we imagine the velocity-updating steps occurring *simultaneously*, there will ultimately be an order of execution in the code itself. Does the order of these steps matter? Try rearranging (1) - (3) from Algorithm 2 in your code and state your observations.

(e) (8 pts) I claimed in class that the slowing with probability p in step (3) is what *causes* the jams to appear. Let's verify that in our simulation. Take out step (c) and run your model for some time. Do you observe any jams?

(f) (10 pts) Sarah pointed out that for step (2), we might not crash into the car in front of us even if $v \geq d$, because the car ahead is also traveling forward by some velocity $v \geq 0$. That is, we might be *too* cautious by worrying about whether the car ahead will randomly stop. (How likely is that, anyway?) Instead, take out step (2). When re-computing $\boldsymbol{d}$ (e.g. after the `for` loop in Alg. 1), only force a car to slow down if it is actually going to collide with someone. What effect, if any, does this have on the traffic jams?

(g) **Bonus**. (Title of Ultimate UQ Star) Animate your graphic in time. Include your code, save your animation (e.g., as a GIF), and include a link to the animation in your submission.

2. (50 pts) **Cutie π.**

Estimate π by computing a Monte Carlo integral. If you didn't take notes in class that day, get them from a friend or ask me for them.

This one is a classic problem for getting accustomed to integrating with Monte Carlo. You can likely find the answer on any data science blog online, but **try** to do this on your own. It is a good reality-check for whether you are internalizing the relevant concepts.

3. (50 pts) **Orthogonal Polynomials and Gaussian Quadrature.**
Note. You might need to wait until after Monday's lecture (on 2/6) to complete this problem, depending on how much we cover in class on Friday.

Let $\{\pi_k(s)\}_{k=0}^{\infty}$ be a collection of polynomials defined on domain $\mathcal{D} \subseteq \mathbb{R}$ that are orthonormal with respect to a weight function $w(s) : \mathcal{D} \to [0, \infty)$. That is, $\langle \pi_i, \pi_j \rangle_w = \delta_{ij}$, where δ_{ij} is the Kronecker delta. We require w to satisfy the assumptions given in class, namely
(a) $\int_{\mathcal{D}} s^k w(s)\, ds < \infty$ for all $k = 0, 1, 2, \ldots$
(b) $\int_{\mathcal{D}} w(s)\, ds = 1$.
These allow for w to be interpreted as a probability density of a random variable that has all finite moments.

It is known that $\{\pi_k(s)\}_{k=0}^{\infty}$ satisfy a three-term recurrence relation

$$s\pi_k(s) = \beta_k \pi_{k-1}(s) + \alpha_k \pi_k(s) + \beta_{k+1}\pi_{k+1}(s) \tag{1}$$

where we set $\pi_{-1}(s) = 0$ and $\pi_0(s) = 1$. Let $\boldsymbol{\pi}(s) = [\pi_0(s), \ldots, \pi_{n-1}(s)]^T$ be a vector of functions. Let $\boldsymbol{e}_n$ denote the nth standard basis vector in $\mathbb{R}^n$, and

$$J_n = \begin{bmatrix} \alpha_0 & \beta_1 & & & \\ \beta_1 & \alpha_1 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-2} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_{n-1} \end{bmatrix}.$$

In class, we will confirm that

$$s\boldsymbol{\pi}(s) = J_n \boldsymbol{\pi}(s) + \beta_n \pi_n(s) \boldsymbol{e}_n.$$

(a) (25 pts) Let $\mathrm{id} : \mathcal{D} \to \mathcal{D}$ be the identity map (i.e. $\mathrm{id}(s) = s$). We used Eq. 1 to show that $\alpha_i = \langle \mathrm{id}, \pi_i^2 \rangle_w$. Prove that $\beta_i = \langle \mathrm{id}\, \pi_{i-1}, \pi_i \rangle_w$.

We will see in class that the zeros $\{\lambda_i\}_{i=0}^{n-1}$ of the nth orthogonal polynomial π_n are the eigenvalues of J_n with corresponding eigenvectors $\boldsymbol{\pi}(\lambda_i)$. Since J_n is a real symmetric matrix, this means the roots λ_i of π_n are positive and real valued. For a random variable ξ whose PDF is w, these λ_i (the *abscissas*) are precisely the sampled values of ξ that we use for estimating the expected value of the response to an uncertainty propagation problem.

(b) (25 pts) How do we know the roots of π_n are unique? Prove it! **Note.** I might send out a hint/template for how this proof goes if people are stuck.