CME 270
L. Lyman
Fall 2021

Assignment 1

Due: **Thursday, October 21**, by 11:59 pm PDT
Submitted in class, on Canvas, 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, through the Canvas site, or by email. (ii) If emailing your assignment, please to do to `lymanla@stanford.edu` with `CME 270` in the subject line. (iii) 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.). (iv) Export your answers (plots, numerical values, etc.), along with all code used to generate those answers, as a PDF. (v) Use best coding practices when possible. Extract repeated code with functions, use descriptive variable names, and add informative comments. (vi) You are encouraged to work with others, but each person is responsible for submitting an individual writeup. (vii) You are required to do:

- Exercises 1 and 6
- **Any Two** of Exercises 2 - 5

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 ($N \leq M$)
   - Discrete time steps $t_0, t_1, \ldots$ of identical length

   We can picture the road via Figure 1. Further, we assume the following.

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

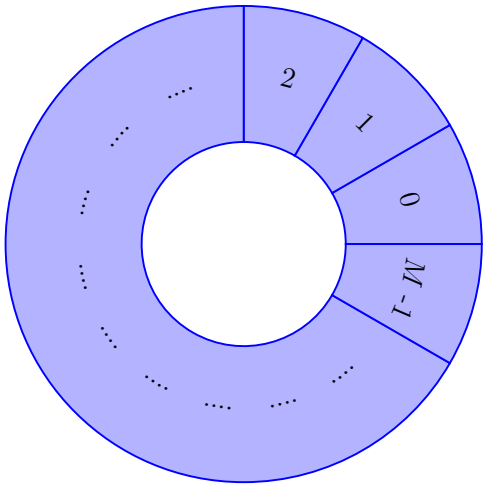

Figure 1: A circular, one-way, single lane road with M spots, labeled $0, \ldots, M-1$.

Algorithm 1: NS-update

input

- The position $x \in \{0, \ldots, M-1\}$ and velocity v of a 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** $v_i \leftarrow \min(v_i + 1, v_{\max})$ (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** $v_i \leftarrow \min(d - 1, v_{\max})$ (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

Of course, if you prefer, this routine (Algorithm 1) could 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.

As was pointed out in class, steps (1) - (3) happen *instantaneously*. That is, we assume a driver does the mental calculations for these velocity updates and then perfectly transitions to the new speed — all in one time step.

Possible pseudocode for the whole procedure could look like the following (Algorithm 2) 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 2: Nagel-Schreckenberg

input
- M, N, p

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 $d_i = |x_{i+1 \pmod N} - x_i|$ // 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 $(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{v})$

(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 for your figure in (b).

(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.

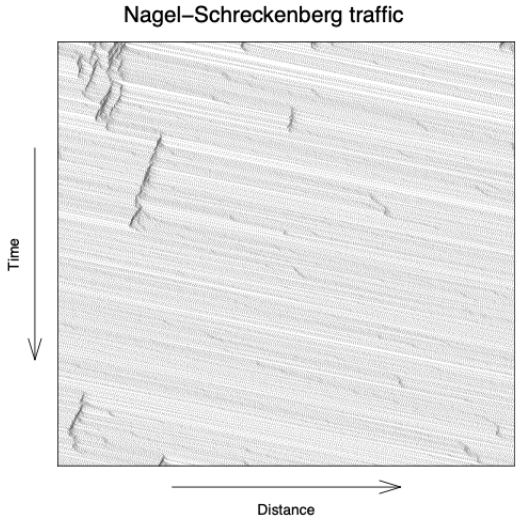

Figure 2: Flow trace image for NS traffic modeling.

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) Derek asked in class about what happens if you permute any of the velocity-updating steps. That is, does the order of these steps matter? Try rearranging (1) - (3) from Algorithm 1 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) Adam 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. 2), 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. (15 pts) **Hey there, cutie pie! Estimating π with Monte Carlo.**

   Estimate π by computing a Monte Carlo integral. If you're stuck, go through the video of the third lecture.

   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. (15 pts) **They're not....symmetric?! Losing confidence in confidence intervals.**

   During the fourth lecture, we examined the plot shown in Figure 3. The code for generating it is available here. Your supervisors (Tara et. al) are worried that the confidence interval in light green doesn't seem symmetric. It sure doesn't look symmetric! An especially troublesome portion is emphasized in red. What gives?

   Take a look at the code for generating this figure. If there is a mistake, explain. If not, then what is happening in Figure 3? Be specific. For example, if you think there is no mistake, try to say more than, *"It's a visual illusion."* You can draw pictures, write code, do calculations....whatever you need to convince your superiors of your point.

4. (15 pts) **Orthogonal Polynomials and Gaussian Quadrature.**
   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 $k = 0, 1, 2, \ldots$

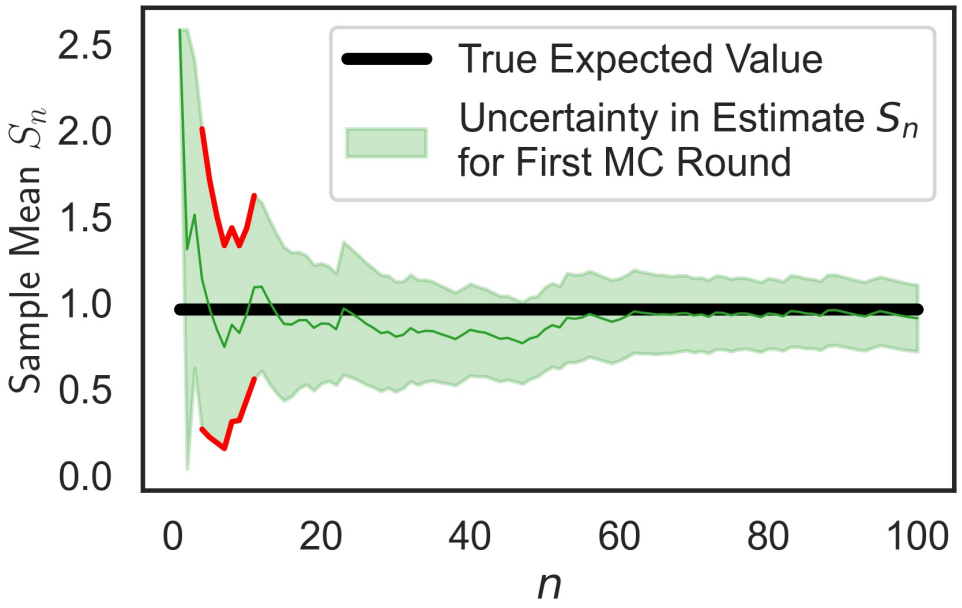

Figure 3: Confidence interval for S_n converging to the true mean $\mathbb{E}(f(X))$ for $f(x) = (\cos(50x) + \sin(20x))^2$ and $X \sim U(0,1)$.

(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 confirmed that

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

(a) (3 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 saw 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) (12 pts) Leah asked, "How do we know the roots of π_n are unique?" Prove it! **Instructor note**: I might send out a hint/template for how this proof goes if people are stuck.

5. (15 pts) **Newton-Cotes Quadrature.**
Suppose we are integrating the function $f(x) = \cos(3x)$ over the interval $x \in [-1, 1]$.

(a) (10 pts) Write code to integrate the function numerically via a Newton-Cotes rule with $n = 3$ quadrature points. Plot the result.

(b) (5 pts) What happens when we increase n? Test out $n = 10$ and then $n = 100$. Why is the approximation not improving? This is one of the limitations of Newton-Cotes.

6. (20 pts) **Gaussian Quadrature.**
Gaussian quadrature to the rescue! It can integrate polynomials exactly up to degree $2n+1$ — and no other quadrature rule with n points can do so. Let's try the integration $\int_{-1}^{1} f(x)dx$ for $f(x) = \cos(3x)$ from the previous problem with this technique.

(a) (10 pts) Implement your own Gaussian quadrature code. Compare your estimate of the definite integral with $n = 10$ nodes to that of a built-in integration tool in your programming language of choice.

(b) Now let's explore some limitations of Gaussian quadrature.

(i) (5 pts) Every time you increase the number of quadrature points, you get completely different nodes. This means that you cannot reuse the function evaluations you have seen so far. To verify this, plot the selected quadrature points on the function $f(x)$ to verify that they change for $n = 3, 4, 5$ with different colors for each n value. Why would we expect the nodes to be different for each n value? (A one sentence answer is sufficient.)

(ii) (5 pts) We mentioned in lecture that polynomial approximation methods work best for smooth functions. Let's see how Gaussian quadrature struggles when we have a non-smooth function. Instead let $f(x)$ be a step function, and use your code to try to integrate it. What do you get?