OpenAI

Experiments 3 - Lake 1

Lake problem describes a situation where inhabitants of a town intend to increase their economic benefits through developing industry and agriculture. These activities will also emit pollution into a lake nearby. Although the lake pollution decreases continuously due to natural removal, once it exceeds a threshold X_{crit}, irreversible lake eutrophication would occur and cause huge losses. This problem requires decision-makers to decide the annual pollution emissions, with the aim of maximizing the economic benefit while avoiding eutrophication. The lake pollution transition function is given by:
\begin{equation}
    X_{(t+1)}=X_t+a_t+\frac{(X_t^q)}{(1+X_t^q)}- bX_t+\epsilon_t
\end{equation}
where X_t is the pollution at time t, and $X_0 = 0$; a_t is the rate of anthropogenic pollution at time t; b is the lake’s natural removal rate; q is the lake's natural recycling rate; ϵ_t is the rate of natural pollution at time t. The natural pollution ϵ_t is modeled as a log normal distribution with mean μ and variance σ^2. The threshold X_{crit} is the pollution level at which the lake’s natural recycling equals its natural removal rate. The total benefit from the pollution is given by
\begin{equation}
    f_{economic} = \sum\limits_{t \in {T}}\alpha a_t \delta^t
\end{equation}
where α is the benefit-to-pollution ratio, and δ is the discount rate.

The lake problem is characterized by both stochastic uncertainty and deep uncertainty. The stochastic uncertainty arises from the natural inflow. Deep uncertainty is presented by uncertainty about the mean μ and standard deviation $sigma$ of the lognormal distribution characterizing the natural inflow, the natural removal rate of the lake β, the natural recycling rate of the lake q, and the discount rate δ.

Please formalize this problem description as a model. Please also specify key components, such as state variables, stochastic variables, constant parameters, uncertainties and transition functions.

Excellent — you’re essentially describing the **formal model formulation** of the *Lake Problem* as it is used in environmental decision analysis, robust decision-making (RDM), and reinforcement learning literature (e.g., Carpenter et al., 1999; Quinn et al., 2017).

Here’s a **structured formalization** of your problem, with clearly identified components.

---

1. Model Overview

The **Lake Problem** is a *stochastic dynamic decision problem* with *deep uncertainty*.
At each time step $t \in \{0, 1, 2, \ldots, T\}$, a decision-maker chooses an **anthropogenic pollution emission rate** a_t to balance:

- **Economic benefit** from pollution (short-term gain), and

- **Environmental risk** of crossing an **irreversible eutrophication threshold** X_{crit}.

2. State and Decision Variables

Symbol	Description	Type
X_t	Pollution level (phosphorus concentration) in the lake at time t	**State variable**
a_t	Anthropogenic (human) pollution emission rate at time t	**Decision variable (control)**

3. Transition Function

The **lake pollution dynamics** evolve according to:

$$X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where:

- $\frac{X_t^q}{1+X_t^q}$: non-linear **phosphorus recycling** term,
- $-bX_t$: **natural removal** term,
- ϵ_t: **stochastic natural inflow** of pollution.

4. Stochastic Variables

Symbol	Description	Distribution	Parameters
ϵ_t	Natural inflow of pollution (e.g., runoff)	Lognormal	Mean μ, variance σ^2

$$\epsilon_t \sim \text{Lognormal}(\mu, \sigma^2)$$

5. Reward (Objective) Function

The **economic benefit** in each period is proportional to anthropogenic emissions:

$$r_t = \alpha a_t$$

The **total discounted economic benefit** over the planning horizon T is:

$$f_{\text{economic}} = \sum_{t=0}^{T} \alpha a_t \delta^t$$

where:

- α: benefit-to-pollution ratio,
- δ: discount factor ($0 < \delta \leq 1$).

6. Constraints

- **Initial condition:**

$$X_0 = 0$$

- **Pollution threshold:**

$$X_t < X_{\text{crit}}, \quad \forall t$$

where X_{crit} is the level beyond which eutrophication becomes irreversible.

- **Control bounds (if any):**

$$0 \leq a_t \leq a_{\max}$$

---

7. Deep Uncertainties

These are parameters whose values are **unknown or highly uncertain**, even in distributional form:

Symbol	Description	Type of Uncertainty
μ	Mean of natural inflow distribution	Deep uncertainty
σ	Standard deviation of natural inflow distribution	Deep uncertainty
b	Lake's natural removal rate	Deep uncertainty
q	Lake's natural recycling rate	Deep uncertainty
δ	Discount rate	Deep uncertainty

Decision-makers may consider **multiple plausible values** or **scenarios** for these parameters when evaluating robustness.

---

8. Critical Threshold Definition

The eutrophication threshold X_{crit} is defined implicitly by the equality of recycling and removal rates:

$$\frac{X_{\text{crit}}^q}{1 + X_{\text{crit}}^q} = bX_{\text{crit}}$$

At this point, the lake's natural ability to recover from pollution collapses.

---

9. Formal Model Structure

The model can be summarized as a **stochastic dynamic programming (SDP)** problem:

$$
\begin{aligned}
\max_{\{a_t\}_{t=0}^{T}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right] \\
\text{s.t.} \quad & X_{t+1} = X_t + a_t + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t, \\
& X_0 = 0, \\
& X_t < X_{\text{crit}}, \\
& a_t \geq 0, \\
& \epsilon_t \sim \text{Lognormal}(\mu, \sigma^2).
\end{aligned}
$$

Under deep uncertainty, this may instead be treated as a **robust decision problem**:

$$
\max_{a_t} \min_{\mu,\sigma,b,q,\delta \in \mathcal{U}} \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right],
$$

where $\mathcal{U}$ represents the space of plausible uncertain parameters.

10. Model Components Summary

Category	Symbols	Description
State Variable	X_t	Pollution concentration
Decision Variable	a_t	Anthropogenic pollution emission
Stochastic Variable	ϵ_t	Natural inflow (lognormal)
Transition Function	$X_{t+1} = X_t + a_t + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$	State evolution
Reward Function	$r_t = \alpha a_t$	Economic benefit
Objective	$\max \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right]$	Maximize discounted benefit
Constraints	$X_t < X_{\text{crit}}, a_t \geq 0, X_0 = 0$	Feasibility
Parameters (known)	$\alpha, T, X_{\text{crit}}$	Fixed inputs
Deep Uncertainties	$\mu, \sigma, b, q, \delta$	Unknown or scenario-based

Would you like me to extend this into a **computational formulation** (e.g., for dynamic programming or reinforcement learning) — showing how to represent the state space, control space, and value function?

X_{crit} is not a constant because it depends on other deeply uncertain parameters. Please refine your model specification.

Excellent catch — you're absolutely right.
In the **Lake Problem**, the eutrophication threshold X_{crit} is **not fixed**; it is **endogenous** to the system because it depends on the nonlinear relationship between the **recycling rate** q and the **natural removal rate** b, both of which are deeply uncertain.

Let's refine the formal model accordingly.

Refined Model Specification (with endogenous X_{crit})

1. Problem Overview

We are optimizing the **anthropogenic pollution emissions** a_t over time to maximize discounted economic benefits, while managing the risk of **irreversible eutrophication** of a lake.

The lake's **pollution stock** X_t evolves stochastically under both:

- **Stochastic uncertainty** (natural inflow ϵ_t), and
- **Deep uncertainty** (structural and parametric uncertainty in $b, q, \mu, \sigma, \delta$).

2. Decision, State, and Stochastic Variables

Symbol	Description	Type
X_t	Lake pollution (phosphorus concentration) at time t	**State variable**
a_t	Anthropogenic pollution emission rate at time t	**Decision variable (control)**
ϵ_t	Natural inflow of pollution	**Stochastic variable**

3. System Dynamics (Transition Function)

$$X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where:

- $\frac{X_t^q}{1+X_t^q}$: nonlinear phosphorus recycling,
- $-bX_t$: natural removal,
- $\epsilon_t \sim \text{Lognormal}(\mu, \sigma^2)$: stochastic natural inflow.

4. Eutrophication Threshold X_{crit}

The **critical pollution threshold** is **endogenous**, defined implicitly by the equilibrium condition between the **recycling** and **removal** processes:

$$\frac{X_{\text{crit}}^q}{1 + X_{\text{crit}}^q} = bX_{\text{crit}}.$$

This yields:

$$b = \frac{X_{\text{crit}}^{q-1}}{1 + X_{\text{crit}}^q} \quad \Longrightarrow \quad X_{\text{crit}} = X_{\text{crit}}(b, q).$$

Thus, X_{crit} is **not constant**, but a function of two deeply uncertain parameters:

$$X_{\text{crit}} = f(b, q) = \text{root of } \frac{x^q}{1 + x^q} - bx = 0.$$

Each combination of (b, q) implies a **different critical threshold**, potentially changing the safe pollution boundary.

---

5. Reward (Objective) Function

At each time t:

$$r_t = \alpha a_t,$$

where α is the benefit-to-pollution ratio.

The total **discounted benefit** is:

$$f_{\text{economic}} = \sum_{t=0}^{T} \alpha a_t \, \delta^t,$$

where δ is the discount factor.

---

6. Constraints

1. **Initial condition:**

$$X_0 = 0.$$

2. **Control bounds:**

$$0 \leq a_t \leq a_{\max}.$$

3. **Dynamic feasibility:**
The lake must remain below the (uncertain) critical threshold to avoid irreversible eutrophication:

$$X_t < X_{\text{crit}}(b, q), \quad \forall t.$$

Because X_{crit} depends on uncertain parameters, this constraint must be interpreted **robustly or probabilistically**.

---

7. Sources of Uncertainty

Symbol	Description	Type	Nature
ϵ_t	Natural inflow	**Stochastic variable**	Lognormal(μ, σ^2)
μ	Mean of lognormal inflow	**Deep uncertainty**	Unknown; scenario-dependent
σ	Std. deviation of inflow	**Deep uncertainty**	Unknown; scenario-dependent
b	Natural removal rate	**Deep uncertainty**	Unknown structural parameter

Symbol	Description	Type	Nature
q	Recycling rate	**Deep uncertainty**	Unknown structural parameter
δ	Discount rate	**Deep uncertainty**	Unknown preference parameter

Thus, both the **transition dynamics** and the **safety threshold** $X_{\text{crit}}(b, q)$ are functions of deeply uncertain quantities.

8. Formal Dynamic Optimization Problem

Under stochastic uncertainty only (for fixed $b, q, \mu, \sigma, \delta$):

$$\begin{aligned}
\max_{\{a_t\}_{t=0}^T} \quad & \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right] \\
\text{s.t.} \quad & X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t, \\
& X_t < X_{\text{crit}}(b, q), \\
& X_0 = 0, \\
& a_t \in [0, a_{\max}], \\
& \epsilon_t \sim \text{Lognormal}(\mu, \sigma^2).
\end{aligned}$$

Under **deep uncertainty**, the problem becomes a **robust stochastic control problem**:

$$\max_{\{a_t\}} \min_{(b,q,\mu,\sigma,\delta)\in\mathcal{U}} \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right],$$

subject to the same dynamics, where $\mathcal{U}$ is the set of plausible parameter combinations.

Alternative formulations include **info-gap**, **min–max regret**, or **multi-scenario** approaches (RDM, MORDM, etc.).

9. Model Component Summary (Refined)

Category	Symbol	Description	Notes
State variable	X_t	Lake pollution level	Evolves by transition equation
Decision variable	a_t	Anthropogenic pollution rate	Chosen each period
Stochastic variable	ϵ_t	Natural inflow	Lognormal(μ, σ^2)
Transition function	$X_{t+1} = X_t + a_t + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$	Determines pollution evolution	
Reward function	$r_t = \alpha a_t$	Economic benefit	
Objective	$\max \mathbb{E}\left[\sum_t \alpha a_t \delta^t\right]$	Maximize discounted benefit	
Constraint	$X_t < X_{\text{crit}}(b, q)$	Safe pollution limit	

Category	Symbol	Description	Notes
Endogenous threshold	$X_{\text{crit}}(b,q)$	Solution of $\frac{x^q}{1+x^q} = bx$	
Deep uncertainties	$\mu, \sigma, b, q, \delta$	Affect both dynamics and threshold	
Stochastic uncertainty	ϵ_t	Random natural inflow	

10. Conceptual Interpretation

- The **deep uncertainties** (b, q) alter both:
  - the **speed of recovery** (through b),
  - and the **stability threshold** $X_{\text{crit}}(b,q)$.
- Thus, the lake's resilience to pollution is itself uncertain.
- Decision strategies must therefore be evaluated not just under a single model, but across a set of plausible worlds (parameter combinations).

Would you like me to express this refined model in a **computational-ready form** (e.g., a dynamic programming or reinforcement-learning state-space model) that explicitly encodes $X_{\text{crit}}(b,q)$ as a derived variable?

Various stakeholders approach this problem from different perspectives. Please refine the model specification from the perspectives of: (1) the local community; and (2) the environmental regulator. These two perspectives should share a common environment, with each stakeholder independently controlling its own control variables and indirectly influencing each other through the shared environment. In the specification, specify the common environment, including global parameters and variables. For each perspective, also specify the corresponding model and its key components, such as control variables, transition functions, and objective functions. If there are multiple objectives, provide multiple objective functions instead of a weighted objective function.

Perfect — you're now moving from a **single-agent** to a **multi-agent, shared-environment formulation** of the *Lake Problem*.

This is an excellent step because in reality, lake management involves **multiple stakeholders** with **different objectives** and **control levers** who interact indirectly through a **shared environmental state** (the lake pollution stock).

Let's formalize this **two-agent lake problem**:

I. Shared Environment

The **shared environment** is the **lake system** itself — it provides the *physical coupling* between the two stakeholders:

- the **local community**, and
- the **environmental regulator**.

All stakeholders observe and influence a **common pollution state** X_t, which evolves under **biophysical dynamics** and **stochastic natural inflows**.

---

1. Common Global Variables and Parameters

Symbol	Description	Type
X_t	Lake pollution (phosphorus concentration) at time t	**Shared state variable**
ϵ_t	Natural stochastic inflow	**Stochastic variable**
b	Natural removal rate	**Global parameter (deeply uncertain)**
q	Natural recycling rate	**Global parameter (deeply uncertain)**
μ, σ	Mean and variance of inflow distribution	**Global parameters (deeply uncertain)**
$X_{\text{crit}}(b, q)$	Critical eutrophication threshold	**Derived variable** (depends on b, q)
T	Time horizon	Global constant

---

2. Shared Transition Function (Environmental Dynamics)

$$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where:

- $a_t^{(L)}$: anthropogenic pollution from the **local community**,
- $a_t^{(R)}$: pollution allowance or offset determined by the **regulator**,
- $\epsilon_t \sim \mathrm{Lognormal}(\mu, \sigma^2)$: stochastic inflow.

This function defines the **shared system transition**—the “common environment” that both agents affect and observe.

II. Perspective 1: Local Community

1. Role

The **local community** seeks to maximize **economic benefits** from industrial and agricultural activities that emit pollution.

2. Control Variable

Symbol	Description	Type
$a_t^{(L)}$	Anthropogenic emissions (e.g., fertilizer use, industrial runoff)	Control variable

3. Observed Variables

- Lake pollution level X_t
- Possibly regulatory signals (tax/subsidy, permit cap, etc.) set by regulator

4. Objective Functions

The local community may have **two main objectives**:

1. **Economic benefit maximization**

$$f_{\text{econ}}^{(L)} = \sum_{t=0}^{T} \alpha^{(L)} a_t^{(L)} \left(\delta^{(L)}\right)^t$$

2. **Avoidance of local damage or loss of amenities** (e.g., tourism, fisheries, water quality degradation):

$$f_{\text{env}}^{(L)} = -\sum_{t=0}^{T} \beta^{(L)} \max(0, X_t - X_{\text{safe}})$$

where $X_{\text{safe}} < X_{\text{crit}}(b, q)$ is a tolerable pollution level.

Thus, the local community faces a **multi-objective trade-off**:

$$\max \left(f_{\text{econ}}^{(L)},\ f_{\text{env}}^{(L)}\right)$$

5. Transition Function

The community takes $a_t^{(L)}$ as control input to the **shared** transition:

$$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

6. Constraints

$$a_t^{(L)} \geq 0, \quad X_t < X_{\text{crit}}(b, q)$$

and possibly:

$$a_t^{(L)} \leq A_{\text{max}}^{(L)}.$$

III. Perspective 2: Environmental Regulator

1. Role

The **regulator** seeks to ensure the **long-term ecological sustainability** of the lake system, possibly while maintaining some level of economic productivity.

2. Control Variables

Symbol	Description	Type
$a_t^{(R)}$	Regulatory control influencing total emissions	Control variable
Examples	Emission cap, tax, permit allocation, offset investment	—

3. Observed Variables

- Pollution state X_t
- Possibly local emissions $a_t^{(L)}$ (observed or reported)

4. Objective Functions

The regulator may have **multiple objectives**:

1. **Environmental quality (lake health) preservation**

$$f_{\text{env}}^{(R)} = -\sum_{t=0}^{T} \gamma^{(R)} \max(0, X_t - X_{\text{crit}}(b, q))$$

   penalizing exceedance of the critical threshold.

2. **Economic welfare or social stability**

$$f_{\text{econ}}^{(R)} = \sum_{t=0}^{T} \eta^{(R)} a_t^{(L)} (\delta^{(R)})^t$$

   reflecting the regulator's desire to sustain economic productivity or tax revenue.

The regulator thus faces a **multi-objective control problem**:

$$\max \left(f_{\text{env}}^{(R)},\ f_{\text{econ}}^{(R)} \right)$$

5. Transition Function

Same shared environment:

$$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

6. Constraints

$$X_t < X_{\text{crit}}(b, q), \quad a_t^{(R)} \in [A_{\min}^{(R)}, A_{\max}^{(R)}].$$

IV. Common Environment and Interdependence

Aspect	Description
Shared variable	X_t: pollution stock affects both agents' payoffs.
Coupling mechanism	Both $a_t^{(L)}$ and $a_t^{(R)}$ enter the same state transition.
Indirect influence	The local community's emissions worsen pollution, triggering stricter regulation; the regulator's controls constrain community's choices.
Uncertainties	$b, q, \mu, \sigma, \delta$ affect both stakeholders identically.
Threshold dependence	$X_{\text{crit}}(b, q)$ depends on global parameters shared by both.

V. Formal Multi-Agent System

Joint Transition

$$X_{t+1} = F(X_t, a_t^{(L)}, a_t^{(R)}, \epsilon_t; b, q) = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

Shared Stochastic Process

$$\epsilon_t \sim \text{Lognormal}(\mu, \sigma^2)$$

Joint Deep Uncertainty Set

$$\Theta = \{b, q, \mu, \sigma, \delta\}$$

Individual Objective Sets

- **Local community:**

$$\max_{a_t^{(L)}} \left(f_{\text{econ}}^{(L)}, f_{\text{env}}^{(L)}\right)$$

- **Regulator:**

$$\max_{a_t^{(R)}} \left(f_{\text{env}}^{(R)}, f_{\text{econ}}^{(R)}\right)$$

Equilibrium or Coordination Concept (optional)

Depending on the analytical goal:

- *Non-cooperative (game-theoretic)*: Nash equilibrium in controls $a_t^{(L)}$ and $a_t^{(R)}$.
- *Cooperative (social planner)*: Pareto-optimal joint policy maximizing both sets of objectives.
- *Regulator-led (Stackelberg)*: Regulator as leader, community as follower.

VI. Summary Table

Component	Local Community	Environmental Regulator	Shared Environment
State variable	observes X_t	observes X_t	X_t
Control variable	$a_t^{(L)}$: pollution emissions	$a_t^{(R)}$: regulatory action	affects X_{t+1}
Transition function	depends on $a_t^{(L)}, a_t^{(R)}, \epsilon_t$	same	$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$
Objectives	$f_{\text{econ}}^{(L)}, f_{\text{env}}^{(L)}$	$f_{\text{env}}^{(R)}, f_{\text{econ}}^{(R)}$	—
Uncertainties	shares global $(b, q, \mu, \sigma, \delta)$	shares global $(b, q, \mu, \sigma, \delta)$	ϵ_t, $X_{\text{crit}}(b, q)$
Interactions	Emits pollutants → increases X_t → triggers regulator's response	Regulates emissions → constrains $a_t^{(L)}$	Coupled through X_t

Interpretation

- Both agents act **independently**, each optimizing its **own multi-objective function**.
- The **lake pollution stock** X_t is the *only coupling mechanism* — changes in X_t propagate both agents' payoffs.
- The **deep uncertainties** (e.g., b, q) affect both agents' problem structures and the critical threshold $X_{\text{crit}}(b, q)$.
- The model can represent multiple governance regimes:
  - **Decentralized (non-cooperative)**: Each agent acts myopically or strategically.
  - **Centralized (cooperative)**: Joint optimization (Pareto frontier).
  - **Hierarchical (Stackelberg)**: Regulator leads, community follows.

Would you like me to extend this into a **game-theoretic dynamic model** (e.g., Stackelberg formulation with regulator leadership), showing each agent's optimization problem and equilibrium concept explicitly?

To ensure internal consistency across these models, please compose them into a single, unified model that shares a common environment and states, while embedding perspective-specific controls and objectives. The composed model should: 1. embed each perspective's control variables and objectives; 2. enable modular evaluation of each perspective; 3. for each perspective, treat other perspectives' controls as exogenous inputs; and 4. incorporate all other components from every perspective.

Excellent — this is the right next step.

You now want a **unified, internally consistent model** that formally integrates all stakeholder perspectives into one **modular multi-agent framework**, while preserving each perspective's autonomy, objectives, and control levers.

Below is a rigorous formulation of the **Composed Multi-Agent Lake Problem Model**.

Unified Multi-Agent Lake Problem Model

I. Model Structure Overview

We model the *lake system* as a **shared stochastic dynamic environment** $\mathcal{E}$, observed and influenced by multiple decision-making stakeholders (agents):

- **Local Community (L)** – seeks economic gain from pollution-emitting activities, while avoiding local environmental damage.
- **Environmental Regulator (R)** – seeks to maintain lake ecological integrity and sustainable economic use.

Both stakeholders interact **indirectly** through their influence on the shared environmental state X_t.

II. Shared Environment $\mathcal{E}$

1. State Variable

Symbol	Meaning	Type
X_t	Pollution concentration (phosphorus) in the lake at time t	Shared state variable

2. Stochastic Process

Symbol	Meaning	Distribution
ϵ_t	Natural stochastic inflow of pollution	$\epsilon_t \sim \mathrm{Lognormal}(\mu, \sigma^2)$

3. Global Parameters

Symbol	Description	Nature
b	Natural removal rate of the lake	Deep uncertainty
q	Natural recycling rate	Deep uncertainty
μ, σ	Parameters of natural inflow distribution	Deep uncertainty
δ	Discount rate	Deep uncertainty
T	Time horizon	Known constant

4. Derived Global Quantity — Critical Threshold

The eutrophication threshold is **endogenous**:

$$\frac{X_{\text{crit}}^q}{1+X_{\text{crit}}^q} = bX_{\text{crit}} \quad \Longrightarrow \quad X_{\text{crit}} = X_{\text{crit}}(b,q)$$

This defines the pollution level beyond which the lake transitions irreversibly to eutrophic state.

---

5. Shared State Transition Function

$$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$$

where:

- $a_t^{(L)}$: local community emissions (pollution source),
- $a_t^{(R)}$: regulator action (policy intervention that increases or decreases net emissions).

Both stakeholders observe X_t, and the state evolves **jointly** according to their actions and stochastic inflow.

III. Embedded Stakeholder Models

Each stakeholder has its own **control variable(s)**, **objectives**, and **evaluation module**, while treating the other's controls as **exogenous inputs**.
All share the same environmental state and transition process.

A. Local Community Model $\mathcal{M}^{(L)}$

Category	Definition
Control variable	$a_t^{(L)} \in [0, A_{\max}^{(L)}]$ — rate of anthropogenic emissions
State variable	X_t (shared)
Exogenous inputs	$a_t^{(R)}$ (regulator's policy)
Transition	$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$
Uncertainties	ϵ_t, $(b, q, \mu, \sigma, \delta)$
Constraints	$X_t < X_{\text{crit}}(b,q)$

Objectives

The community typically pursues two **separate objectives**:

1. **Economic benefit:**

$$f_{\text{econ}}^{(L)} = \sum_{t=0}^{T} \alpha^{(L)} a_t^{(L)} (\delta^{(L)})^t$$

2. **Environmental quality / local welfare:**

$$f_{\text{env}}^{(L)} = -\sum_{t=0}^{T} \beta^{(L)} \max(0, X_t - X_{\text{safe}})$$

Multi-objective formulation:

$$\max_{a_t^{(L)}} \left(f_{\text{econ}}^{(L)},\ f_{\text{env}}^{(L)} \right)$$

B. Environmental Regulator Model $\mathcal{M}^{(R)}$

Category	Definition
Control variable	$a_t^{(R)} \in [A_{\text{min}}^{(R)}, A_{\text{max}}^{(R)}]$ — regulatory intervention (e.g., tax offset, removal effort)
State variable	X_t (shared)
Exogenous inputs	$a_t^{(L)}$ (local emissions)
Transition	$X_{t+1} = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1+X_t^q} - bX_t + \epsilon_t$
Uncertainties	ϵ_t, $(b, q, \mu, \sigma, \delta)$
Constraints	$X_t < X_{\text{crit}}(b, q)$

Objectives

The regulator typically pursues two **separate objectives**:

1. **Environmental protection (avoid eutrophication):**

$$f_{\text{env}}^{(R)} = -\sum_{t=0}^{T} \gamma^{(R)} \max(0, X_t - X_{\text{crit}}(b, q))$$

2. **Socioeconomic welfare:**

$$f_{\text{econ}}^{(R)} = \sum_{t=0}^{T} \eta^{(R)} a_t^{(L)} (\delta^{(R)})^t$$

(representing the regulator's interest in maintaining regional economic activity).

Multi-objective formulation:

$$\max_{a_t^{(R)}} \left(f_{\text{env}}^{(R)},\ f_{\text{econ}}^{(R)} \right)$$

IV. Composed System Representation

We now compose the shared environment and the two stakeholder submodels into a single integrated framework.

---

1. Common Transition Kernel

Define the **joint transition function** F over the shared state space:

$$F : (X_t, a_t^{(L)}, a_t^{(R)}, \epsilon_t; b, q) \;\mapsto\; X_{t+1}$$

Explicitly:

$$X_{t+1} = F(X_t, a_t^{(L)}, a_t^{(R)}, \epsilon_t; b, q) = X_t + a_t^{(L)} + a_t^{(R)} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

---

2. Unified Model Tuple

The **composed multi-agent lake model** can be expressed as:

$$\mathcal{M} = \left(\mathcal{E},\ \mathcal{M}^{(L)},\ \mathcal{M}^{(R)}\right)$$

where:

$$\begin{aligned}
\mathcal{E} &= \left(X_t, \epsilon_t, b, q, \mu, \sigma, \delta, X_{\text{crit}}(b,q), F\right), \\
\mathcal{M}^{(L)} &= \left(a_t^{(L)}, f_{\text{econ}}^{(L)}, f_{\text{env}}^{(L)}\right), \\
\mathcal{M}^{(R)} &= \left(a_t^{(R)}, f_{\text{env}}^{(R)}, f_{\text{econ}}^{(R)}\right).
\end{aligned}$$

---

3. Temporal Process

At each time t:

1. The environment exposes the current state X_t.
2. Each stakeholder independently chooses its control:

$$a_t^{(L)} = \pi^{(L)}(X_t), \quad a_t^{(R)} = \pi^{(R)}(X_t)$$

   according to its policy (possibly adaptive or stochastic).
3. The environment evolves:

$$X_{t+1} = F(X_t, a_t^{(L)}, a_t^{(R)}, \epsilon_t; b, q)$$

4. Rewards/objectives are accumulated separately:

$$\begin{aligned}
r_t^{(L)} &= (\alpha^{(L)} a_t^{(L)},\ -\beta^{(L)} \max(0, X_t - X_{\text{safe}})), \\
r_t^{(R)} &= (-\gamma^{(R)} \max(0, X_t - X_{\text{crit}}(b,q)),\ \eta^{(R)} a_t^{(L)}).
\end{aligned}$$

---

4. Independence and Modularity

Each perspective can be **evaluated modularly** by fixing the other's policy as an **exogenous input**:

- **Community evaluation:** Fix $a_t^{(R)} = \bar{a}_t^{(R)}$, optimize $a_t^{(L)}$ for $\mathcal{M}^{(L)}$.

- **Regulator evaluation:** Fix $a_t^{(L)} = \bar{a}_t^{(L)}$, optimize $a_t^{(R)}$ for $\mathcal{M}^{(R)}$.
- **Joint (cooperative) evaluation:** Co-optimize $(a_t^{(L)}, a_t^{(R)})$.

This modular structure supports **independent policy design**, **sensitivity analysis**, and **multi-agent coordination or game-theoretic extensions**.

V. Summary Tables

A. Shared Environment

Symbol	Description	Nature
X_t	Lake pollution concentration	Shared state
ϵ_t	Natural inflow (lognormal)	Stochastic
b, q	Removal & recycling rates	Deeply uncertain parameters
$X_{\text{crit}}(b, q)$	Critical threshold	Derived from b, q
$F(\cdot)$	Transition function	Shared dynamic rule
μ, σ, δ	Inflow and discount parameters	Deep uncertainties

B. Stakeholder Submodels

Element	Local Community $\mathcal{M}^{(L)}$	Environmental Regulator $\mathcal{M}^{(R)}$
Control variable	$a_t^{(L)}$	$a_t^{(R)}$
State	Shared X_t	Shared X_t
Transition	Shared $F(\cdot)$	Shared $F(\cdot)$
Exogenous input	$a_t^{(R)}$	$a_t^{(L)}$
Objectives	$f_{\text{econ}}^{(L)}, f_{\text{env}}^{(L)}$	$f_{\text{env}}^{(R)}, f_{\text{econ}}^{(R)}$
Evaluation mode	Optimize $a_t^{(L)}$ given fixed $a_t^{(R)}$	Optimize $a_t^{(R)}$ given fixed $a_t^{(L)}$
Deep uncertainties	$b, q, \mu, \sigma, \delta$	$b, q, \mu, \sigma, \delta$

VI. Interpretation

- The **shared lake system** forms the environmental “game board.”
- Each agent acts **independently**, with **coupled consequences** through X_t.
- The model allows:
  - **Individual optimization** (treating others as exogenous),
  - **Cooperative optimization** (joint objectives), or
  - **Strategic interaction** (Nash or Stackelberg equilibria).

- Deep uncertainties affect both agents symmetrically through shared parameters and the dynamic environment.

Would you like me to extend this unified specification into a **computational form** (e.g., a modular stochastic dynamic game model) where each perspective's Bellman equations or value functions are explicitly defined?

Now, provide a modular Python implementation of the unified model, using classes to represent different perspectives. At the end of the program, define a function as the interface. This function should accept a dictionary of control variables and a dictionary of uncertain parameter values as input, simulate the model dynamics, and return the value of each objective function as output. This function does not need to be executed.

Here's a **clean, modular Python implementation** of the *unified multi-agent lake problem model*, built to reflect the composed specification you developed — with a shared environment and two independent stakeholder perspectives (`LocalCommunity` and `Regulator`).

The code is organized to be **extensible and inspectable**, supporting later integration with optimization, sensitivity analysis, or game-theoretic frameworks.

```python
import numpy as np

# ============================================================
# Shared Environment: Lake System
# ============================================================

class LakeEnvironment:
    """
    Shared stochastic lake environment.
    Evolves pollution stock X_t according to both agents' actions and natural inflows.
    """

    def __init__(self, params):
        """
        params: dict containing global/environment parameters.
            {
                'b': natural removal rate,
                'q': natural recycling rate,
                'mu': mean of lognormal inflow,
                'sigma': std. dev. of lognormal inflow,
                'T': time horizon,
                'X0': initial pollution,
            }
        """
        self.b = params["b"]
        self.q = params["q"]
        self.mu = params["mu"]
        self.sigma = params["sigma"]
        self.T = params["T"]
        self.X0 = params.get("X0", 0.0)

        # Compute critical threshold
        self.X_crit = self.compute_critical_threshold()

    def compute_critical_threshold(self):
        """
```

```
        Computes X_crit by solving (x^q)/(1+x^q) = b*x.
        For simplicity, use numerical root-finding via binary search.
        """
        b, q = self.b, self.q
        f = lambda x: (x**q) / (1 + x**q) - b * x
        # Search over reasonable interval
        x_low, x_high = 0.0, 10.0
        for _ in range(100):
            x_mid = 0.5 * (x_low + x_high)
            if f(x_mid) * f(x_low) < 0:
                x_high = x_mid
            else:
                x_low = x_mid
        return x_mid

    def step(self, X_t, a_L, a_R):
        """
        Advances the lake state one period forward.
        X_{t+1} = X_t + a_L + a_R + (X_t^q)/(1+X_t^q) - b*X_t + epsilon_t
        """
        epsilon_t = np.random.lognormal(self.mu, self.sigma)
        X_next = X_t + a_L + a_R + (X_t ** self.q) / (1 + X_t ** self.q) - self.b * X_t +
epsilon_t
        return X_next, epsilon_t


# ==============================================================
# Perspective 1: Local Community
# ==============================================================

class LocalCommunity:
    """
    Local community agent with control over anthropogenic emissions.
    """
    def __init__(self, params):
        """
        params: dict containing community-specific parameters
            {
                'alpha': benefit-to-pollution ratio,
                'beta': penalty weight for environmental loss,
                'delta': discount factor,
                'X_safe': safe pollution level,
            }
        """
        self.alpha = params["alpha"]
        self.beta = params["beta"]
        self.delta = params["delta"]
        self.X_safe = params["X_safe"]

    def objectives(self, X_series, A_L_series):
        """
        Compute economic and environmental objectives for the local community.
        """
        T = len(A_L_series)
        f_econ = sum(self.alpha * A_L_series[t] * (self.delta ** t) for t in range(T))
        f_env = -sum(self.beta * max(0, X_series[t] - self.X_safe) for t in range(T))
        return {"economic": f_econ, "environmental": f_env}


# ==============================================================
# Perspective 2: Environmental Regulator
# ==============================================================

class Regulator:
    """
    Regulator agent controlling policy interventions (e.g., emissions offsets).
    """
    def __init__(self, params):
        """
        params: dict containing regulator-specific parameters
            {
                'gamma': penalty for exceeding X_crit,
                'eta': weight on economic activity,
                'delta': discount factor,
            }
```

```
        """
        self.gamma = params["gamma"]
        self.eta = params["eta"]
        self.delta = params["delta"]

    def objectives(self, X_series, A_L_series, X_crit):
        """
        Compute environmental and economic objectives for the regulator.
        """
        T = len(A_L_series)
        f_env = -sum(self.gamma * max(0, X_series[t] - X_crit) for t in range(T))
        f_econ = sum(self.eta * A_L_series[t] * (self.delta ** t) for t in range(T))
        return {"environmental": f_env, "economic": f_econ}


# ==============================================================
# Unified Multi-Agent Lake Problem Simulation
# ==============================================================

def simulate_lake_problem(control_dict, uncertainty_dict):
    """
    Interface function to simulate the unified multi-agent lake model.

    Inputs:
    -------
    control_dict: dict of control time series for each stakeholder
        {
            'a_L': [a_L0, a_L1, ..., a_LT-1],
            'a_R': [a_R0, a_R1, ..., a_RT-1]
        }

    uncertainty_dict: dict of uncertain/global parameter values
        {
            'b': ..., 'q': ..., 'mu': ..., 'sigma': ..., 'T': ..., 'X0': ...,
            'alpha_L': ..., 'beta_L': ..., 'delta_L': ..., 'X_safe': ...,
            'gamma_R': ..., 'eta_R': ..., 'delta_R': ...
        }

    Output:
    -------
    dict containing objective function values for each perspective:
        {
            'LocalCommunity': {'economic': ..., 'environmental': ...},
            'Regulator': {'environmental': ..., 'economic': ...},
            'Environment': {'X_series': [...], 'X_crit': ...}
        }
    """

    # ----------------------------
    # 1. Initialize Environment
    # ----------------------------
    env_params = {
        "b": uncertainty_dict["b"],
        "q": uncertainty_dict["q"],
        "mu": uncertainty_dict["mu"],
        "sigma": uncertainty_dict["sigma"],
        "T": uncertainty_dict["T"],
        "X0": uncertainty_dict.get("X0", 0.0),
    }
    env = LakeEnvironment(env_params)

    # ----------------------------
    # 2. Initialize Agents
    # ----------------------------
    community_params = {
        "alpha": uncertainty_dict["alpha_L"],
        "beta": uncertainty_dict["beta_L"],
        "delta": uncertainty_dict["delta_L"],
        "X_safe": uncertainty_dict["X_safe"],
    }
    regulator_params = {
        "gamma": uncertainty_dict["gamma_R"],
        "eta": uncertainty_dict["eta_R"],
        "delta": uncertainty_dict["delta_R"],
    }
```

```
    community = LocalCommunity(community_params)
    regulator = Regulator(regulator_params)

    # ----------------------------
    # 3. Simulate Dynamics
    # ----------------------------
    T = env.T
    X = np.zeros(T + 1)
    X[0] = env.X0
    A_L = np.array(control_dict["a_L"])
    A_R = np.array(control_dict["a_R"])

    for t in range(T):
        X[t + 1], _ = env.step(X[t], A_L[t], A_R[t])

    # ----------------------------
    # 4. Evaluate Objectives
    # ----------------------------
    results_L = community.objectives(X[:-1], A_L)
    results_R = regulator.objectives(X[:-1], A_L, env.X_crit)

    # ----------------------------
    # 5. Return Structured Results
    # ----------------------------
    return {
        "LocalCommunity": results_L,
        "Regulator": results_R,
        "Environment": {
            "X_series": X.tolist(),
            "X_crit": env.X_crit,
        },
    }


# ============================================================
# Example usage (not executed)
# ============================================================
# control_inputs = {
#     'a_L': [0.5]*100,
#     'a_R': [0.0]*100
# }
# uncertain_params = {
#     'b': 0.5, 'q': 2.0, 'mu': 0.1, 'sigma': 0.02, 'T': 100, 'X0': 0.0,
#     'alpha_L': 1.0, 'beta_L': 0.5, 'delta_L': 0.95, 'X_safe': 0.8,
#     'gamma_R': 10.0, 'eta_R': 0.5, 'delta_R': 0.95
# }
# results = simulate_lake_problem(control_inputs, uncertain_params)
# print(results)
```

✅ Design Features

1. **Common environment (`LakeEnvironment`)** — shared state X_t, stochastic inflow, and global uncertain parameters b, q, μ, σ.
2. **Perspective modules (`LocalCommunity`, `Regulator`)** — each has its own control, objectives, and evaluation method.
3. **Unified interface (`simulate_lake_problem`)** — provides a clean, modular entry point:
   - Inputs: dictionaries for **controls** and **uncertain parameters**.
   - Output: dictionary of all **objective values** and the **state trajectory**.
4. **Exogenous treatment** — each agent's control can be pre-specified or optimized independently (consistent with modular evaluation).
5. **Extensible** — can easily incorporate:

- Additional agents (e.g., downstream users),
- Cooperative or strategic coupling,
- Scenario sampling for deep uncertainty.

Would you like me to extend this into a **dynamic game version**, where each agent has a policy (e.g., function of X_t) and the system simulates interactions to convergence (Nash or Stackelberg)?

> There are two areas where your implementation could be improved: 1. The possible value of X_{crit} is in the range [0.01, 1.5], and its exact value can be estimated using the root-finding function; 2. Your implementation of the lognormal distribution is incorrect. Please refine your implementation.

Excellent — you are absolutely right on both counts.

✅ **First**, the possible range for X_{crit} should be restricted to the *ecologically plausible interval* $[0.01, 1.5]$, and computed using a **numerical root-finding routine** (e.g., Brent's method) instead of the rough binary search.

✅ **Second**, in NumPy's **`np.random.lognormal(mean, sigma)`**, the parameters correspond to the *underlying normal distribution*, not the lognormal's actual mean/variance. In the lake problem, ϵ_t is usually modeled as a **lognormal with specified mean** μ and **variance** σ^2 of the *lognormal*, not of its underlying normal distribution.

To correct this, we must transform the specified (μ, σ) (mean and std. of lognormal) to the parameters of the underlying normal distribution:

$$m = \ln\left(\frac{\mu^2}{\sqrt{\sigma^2 + \mu^2}}\right), \quad s = \sqrt{\ln\left(1 + \frac{\sigma^2}{\mu^2}\right)}.$$

Below is the **refined, fully corrected, and internally consistent Python implementation**.

```python
import numpy as np
from scipy.optimize import brentq

# ============================================================
# Shared Environment: Lake System
# ============================================================

class LakeEnvironment:
    """
    Shared stochastic lake environment.
    Evolves pollution stock X_t according to both agents' actions and natural inflows.
    """

    def __init__(self, params):
        """
        params: dict containing global/environment parameters.
            {
                'b': natural removal rate,
                'q': natural recycling rate,
                'mu': mean of lognormal inflow (lognormal mean),
```

```
                'sigma': std. dev. of lognormal inflow (lognormal std),
                'T': time horizon,
                'X0': initial pollution,
            }
        """
        self.b = params["b"]
        self.q = params["q"]
        self.mu = params["mu"]
        self.sigma = params["sigma"]
        self.T = params["T"]
        self.X0 = params.get("X0", 0.0)

        # Convert lognormal (mean, std) → underlying normal (mean, std)
        self.norm_mean, self.norm_std = self._lognormal_to_normal(self.mu, self.sigma)

        # Compute critical threshold within [0.01, 1.5]
        self.X_crit = self.compute_critical_threshold()

    def _lognormal_to_normal(self, mu, sigma):
        """
        Convert lognormal mean (mu) and std (sigma)
        into parameters of the underlying normal distribution.
        """
        if mu <= 0:
            raise ValueError("Mean (mu) of lognormal inflow must be positive.")
        s = np.sqrt(np.log(1 + (sigma ** 2) / (mu ** 2)))
        m = np.log(mu) - 0.5 * s**2
        return m, s

    def compute_critical_threshold(self):
        """
        Computes X_crit by solving (x^q)/(1+x^q) = b*x within [0.01, 1.5].
        """
        b, q = self.b, self.q
        def f(x): return (x ** q) / (1 + x ** q) - b * x
        # Check signs before root finding
        f_low, f_high = f(0.01), f(1.5)
        if f_low * f_high > 0:
            # No root in range — fallback to boundary that minimizes |f(x)|
            grid = np.linspace(0.01, 1.5, 200)
            vals = [abs(f(xx)) for xx in grid]
            return grid[np.argmin(vals)]
        else:
            return brentq(f, 0.01, 1.5)

    def sample_inflow(self):
        """
        Draws a stochastic natural inflow ε_t from lognormal(mean=μ, std=σ).
        Uses derived normal parameters (mean, std).
        """
        return np.random.lognormal(self.norm_mean, self.norm_std)

    def step(self, X_t, a_L, a_R):
        """
        Advances the lake state one period forward.
        X_{t+1} = X_t + a_L + a_R + (X_t^q)/(1+X_t^q) - b*X_t + ε_t
        """
        epsilon_t = self.sample_inflow()
        X_next = X_t + a_L + a_R + (X_t ** self.q) / (1 + X_t ** self.q) - self.b * X_t +
epsilon_t
        return X_next, epsilon_t


# ============================================================
# Perspective 1: Local Community
# ============================================================

class LocalCommunity:
    """
    Local community agent with control over anthropogenic emissions.
    """
    def __init__(self, params):
        """
        params: dict containing community-specific parameters
            {
```

```
                'alpha': benefit-to-pollution ratio,
                'beta': penalty weight for environmental loss,
                'delta': discount factor,
                'X_safe': safe pollution level,
            }
        """
        self.alpha = params["alpha"]
        self.beta = params["beta"]
        self.delta = params["delta"]
        self.X_safe = params["X_safe"]

    def objectives(self, X_series, A_L_series):
        """
        Compute economic and environmental objectives for the local community.
        """
        T = len(A_L_series)
        f_econ = sum(self.alpha * A_L_series[t] * (self.delta ** t) for t in range(T))
        f_env = -sum(self.beta * max(0, X_series[t] - self.X_safe) for t in range(T))
        return {"economic": f_econ, "environmental": f_env}


# ============================================================
# Perspective 2: Environmental Regulator
# ============================================================

class Regulator:
    """
    Regulator agent controlling policy interventions (e.g., emissions offsets).
    """
    def __init__(self, params):
        """
        params: dict containing regulator-specific parameters
            {
                'gamma': penalty for exceeding X_crit,
                'eta': weight on economic activity,
                'delta': discount factor,
            }
        """
        self.gamma = params["gamma"]
        self.eta = params["eta"]
        self.delta = params["delta"]

    def objectives(self, X_series, A_L_series, X_crit):
        """
        Compute environmental and economic objectives for the regulator.
        """
        T = len(A_L_series)
        f_env = -sum(self.gamma * max(0, X_series[t] - X_crit) for t in range(T))
        f_econ = sum(self.eta * A_L_series[t] * (self.delta ** t) for t in range(T))
        return {"environmental": f_env, "economic": f_econ}


# ============================================================
# Unified Multi-Agent Lake Problem Simulation
# ============================================================

def simulate_lake_problem(control_dict, uncertainty_dict):
    """
    Interface function to simulate the unified multi-agent lake model.

    Inputs:
    -------
    control_dict: dict of control time series for each stakeholder
        {
            'a_L': [a_L0, a_L1, ..., a_LT-1],
            'a_R': [a_R0, a_R1, ..., a_RT-1]
        }

    uncertainty_dict: dict of uncertain/global parameter values
        {
            'b': ..., 'q': ..., 'mu': ..., 'sigma': ..., 'T': ..., 'X0': ...,
            'alpha_L': ..., 'beta_L': ..., 'delta_L': ..., 'X_safe': ...,
            'gamma_R': ..., 'eta_R': ..., 'delta_R': ...
        }
```

```
    Output:
    -------
    dict containing objective function values for each perspective:
        {
            'LocalCommunity': {'economic': ..., 'environmental': ...},
            'Regulator': {'environmental': ..., 'economic': ...},
            'Environment': {'X_series': [...], 'X_crit': ...}
        }
    """

    # ----------------------------
    # 1. Initialize Environment
    # ----------------------------
    env_params = {
        "b": uncertainty_dict["b"],
        "q": uncertainty_dict["q"],
        "mu": uncertainty_dict["mu"],
        "sigma": uncertainty_dict["sigma"],
        "T": uncertainty_dict["T"],
        "X0": uncertainty_dict.get("X0", 0.0),
    }
    env = LakeEnvironment(env_params)

    # ----------------------------
    # 2. Initialize Agents
    # ----------------------------
    community_params = {
        "alpha": uncertainty_dict["alpha_L"],
        "beta": uncertainty_dict["beta_L"],
        "delta": uncertainty_dict["delta_L"],
        "X_safe": uncertainty_dict["X_safe"],
    }
    regulator_params = {
        "gamma": uncertainty_dict["gamma_R"],
        "eta": uncertainty_dict["eta_R"],
        "delta": uncertainty_dict["delta_R"],
    }
    community = LocalCommunity(community_params)
    regulator = Regulator(regulator_params)

    # ----------------------------
    # 3. Simulate Dynamics
    # ----------------------------
    T = env.T
    X = np.zeros(T + 1)
    X[0] = env.X0
    A_L = np.array(control_dict["a_L"])
    A_R = np.array(control_dict["a_R"])

    for t in range(T):
        X[t + 1], _ = env.step(X[t], A_L[t], A_R[t])

    # ----------------------------
    # 4. Evaluate Objectives
    # ----------------------------
    results_L = community.objectives(X[:-1], A_L)
    results_R = regulator.objectives(X[:-1], A_L, env.X_crit)

    # ----------------------------
    # 5. Return Structured Results
    # ----------------------------
    return {
        "LocalCommunity": results_L,
        "Regulator": results_R,
        "Environment": {
            "X_series": X.tolist(),
            "X_crit": env.X_crit,
        },
    }


# ============================================================
# Example usage (not executed)
# ============================================================
# control_inputs = {
```

```
#     'a_L': [0.5]*100,
#     'a_R': [0.0]*100
# }
# uncertain_params = {
#     'b': 0.5, 'q': 2.0, 'mu': 0.1, 'sigma': 0.02, 'T': 100, 'X0': 0.0,
#     'alpha_L': 1.0, 'beta_L': 0.5, 'delta_L': 0.95, 'X_safe': 0.8,
#     'gamma_R': 10.0, 'eta_R': 0.5, 'delta_R': 0.95
# }
# results = simulate_lake_problem(control_inputs, uncertain_params)
# print(results)
```

✅ Improvements made

1. **Root-finding for X_{crit}**
   - Now uses **`scipy.optimize.brentq()`** with domain **`[0.01, 1.5]`**.
   - Falls back gracefully if no root is found.
2. **Correct lognormal sampling**
   - Converts the lognormal mean/variance (μ, σ) to the *underlying normal parameters* before sampling.
   - Ensures the expected value and variance of ϵ_t match the specified quantities.
3. **Numerically stable and modular design**
   - Each class encapsulates a logical component (environment, agent).
   - The top-level **`simulate_lake_problem()`** provides a unified, extensible interface.

Would you like me to add **vectorized simulation support** (e.g., multiple stochastic runs for scenario sampling under deep uncertainty)?