

Market_3



An energy company is interested in investing in wind-power production. To assess the viability of this investment, the company wants to evaluate the expected revenue of a wind-power producer in a competitive spot market.

Suppose there are five energy producers participating in the day-ahead spot market. Three of them use conventional sources for energy production, such as coal, while the other two producers use solar and wind power, respectively. The market is cleared once per day for 24 hourly intervals of the following day. For each dispatch interval $t=1, \dots, 24$, every producer submits a bid (b_t, p_t) specifying the energy quantity b_t (MWh) it is willing to supply and the corresponding minimum acceptable price p_t (per MWh). Then, the market operator follows a merit-order process to determine the market-clearing price c_t and dispatch schedule, ensuring the total market demand D_t is met, where $D_t \sim \mathcal{N}(\mu_D, \sigma_D^2)$ with constant mean and variance. Merit-order ensures that any bid (b_t, p_t) at interval t , if $p_t \leq c_t$, the bid is accepted, and the producer is committed to delivering b_t at c_t . Otherwise, the producer earns zero revenue.

From the perspective of the wind-power producer, the bid (b_{it}, p_{it}) from a conventional producer i can be modeled as follows: b_{it} can be assumed to be stable, while $p_{it} \sim \mathcal{N}(\mu_{pi}, \sigma_{pi}^2)$ with deeply uncertain mean and variance since its bidding strategy is unknown. For the solar-power producer, its b_{st} is associated with the time t and assumed to be $b_{st} = \max(0, a + b \cos(\frac{2\pi t}{24}))$, where a and b can be forecast based on historical data thereby assumed to be constants. Its p_{st} is also uncertain, where $p_{st} \sim \mathcal{N}(\mu_{ps}, \sigma_{ps}^2)$ with deeply uncertain mean and variance.

The actual energy production of the wind-power producer G_t is also uncertain, as it depends on weather conditions. The production G_t can be forecast from historical data and is modeled as $G_t \sim \mathcal{N}(\mu_G, \sigma_G^2)$ with constant mean and variance. In this case, the actual production may differ from the quantity committed to dispatch one day in advance. If the producer under-delivers relative to its dispatched quantity, the shortfall incurs a penalty of q_u per MWh. This cost is imposed by the market operator to maintain the grid's safe operational conditions and to ensure supply-demand balance in real time.

The energy company aims to build a simulation model to evaluate the expected revenue of the wind-power producer on any future day by selecting the hourly bid quantity and price (b_{wt}, p_{wt}) , while accounting for uncertain generation and stochastic market clearing prices. This model can then be used to explore outcomes across various short-term and long-term scenarios, thereby informing the viability of the investment.

Please formalize this problem description as a simulation model and specify the key components, including state variables, decision variables, transition functions, objective functions, stochastic variables, constant parameters and deep uncertainties.

Below is a clean **formalization of the wind-power producer's problem as a simulation model**, suitable for Monte-Carlo evaluation and scenario analysis. The structure is consistent with stochastic

market-clearing models used in electricity economics.

1. Time Structure and Index Sets

- **Time horizon:** one day, divided into hourly intervals

$$t \in \mathcal{T} = \{1, \dots, 24\}$$

- **Producers:**

- Conventional producers: $i \in \mathcal{I} = \{1, 2, 3\}$
- Solar producer: s
- Wind producer (decision maker): w

2. Decision Variables (Wind Producer)

Chosen **day-ahead**, before uncertainties are realized:

- **Bid quantity:**

$$b_{wt} \geq 0 \quad (\text{MWh})$$

- **Bid price:**

$$p_{wt} \geq 0 \quad (\text{price per MWh})$$

Collectively:

$$x_t = (b_{wt}, p_{wt}), \quad t = 1, \dots, 24$$

3. Stochastic Variables (Short-Term Uncertainty)

These are sampled in the simulation.

3.1 Market Demand

$$D_t \sim \mathcal{N}(\mu_D, \sigma_D^2)$$

3.2 Conventional Producers' Bids

For each $i \in \mathcal{I}$:

$$p_{it} \sim \mathcal{N}(\mu_{pi}, \sigma_{pi}^2), \quad b_{it} = \bar{b}_i$$

3.3 Solar Producer

- Quantity:

$$b_{st} = \max\left(0, a + b \cos\left(\frac{2\pi t}{24}\right)\right)$$

- Price:

$$p_{st} \sim \mathcal{N}(\mu_{ps}, \sigma_{ps}^2)$$

3.4 Wind Generation (Real-Time)

$$G_t \sim \mathcal{N}(\mu_G, \sigma_G^2)$$

4. Market-Clearing Mechanism (Transition Function)

For each hour t :

4.1 Merit-Order Sorting

Define the bid set:

$$\mathcal{B}_t = \{(b_{it}, p_{it})\}_{i \in \mathcal{I}} \cup (b_{st}, p_{st}) \cup (b_{wt}, p_{wt})$$

Sort bids in ascending order of price.

4.2 Clearing Price

The **market-clearing price** c_t is the marginal bid price that satisfies:

$$\sum_{j: p_{jt} \leq c_t} b_{jt} \geq D_t$$

4.3 Dispatch Quantity for Wind

$$d_{wt} = \begin{cases} b_{wt}, & p_{wt} \leq c_t \\ 0, & p_{wt} > c_t \end{cases}$$

5. Penalty for Under-Delivery

Real-time shortfall:

$$u_t = \max(0, d_{wt} - G_t)$$

Penalty cost:

$$\text{Penalty}_t = q_u \cdot u_t$$

6. Revenue Function (Per Hour)

$$R_t = c_t \cdot d_{wt} - q_u \cdot \max(0, d_{wt} - G_t)$$

7. Objective Function

The wind producer seeks to **maximize expected daily profit**:

$$\max_{\{b_{wt}, p_{wt}\}_{t=1}^{24}} \mathbb{E} \left[\sum_{t=1}^{24} R_t \right]$$

Expectation is taken over:

$$\{D_t, p_{it}, p_{st}, G_t\}_{t=1}^{24}$$

8. State Variables (Simulation-Level)

At each hour t , the state includes:

- Realized demand D_t
- Realized bids $\{p_{it}, p_{st}\}$
- Clearing price c_t
- Dispatch d_{wt}
- Realized wind generation G_t

9. Constant Parameters

Known or forecastable inputs:

- μ_D, σ_D (demand)
- μ_G, σ_G (wind production)
- a, b (solar profile)
- q_u (penalty rate)
- \bar{b}_i (conventional capacities)

10. Deep Uncertainties (Scenario-Level)

These are **not probabilistically well characterized** and should be explored via scenario analysis:

- (μ_{pi}, σ_{pi}) : conventional bidding behavior
- (μ_{ps}, σ_{ps}) : solar bidding strategy
- Structural changes in market competitiveness
- Changes in penalty regime q_u
- Correlations between demand, prices, and renewable output

11. Simulation Workflow Summary

1. Choose bidding strategy $(b_{wt}, p_{wt})_{t=1}^{24}$
2. Sample stochastic variables
3. Clear the market hour-by-hour
4. Compute dispatch, penalties, and revenues
5. Aggregate daily profit
6. Repeat under Monte-Carlo and deep-uncertainty scenarios

This formulation provides a **fully specified stochastic simulation model** suitable for:

- Expected revenue estimation
- Risk analysis (variance, CVaR)
- Sensitivity analysis under deep uncertainty
- Investment viability assessment for wind power

If you'd like, I can next:

- Derive an **optimal bidding heuristic**
- Show **pseudo-code** for the simulation
- Extend the model to **risk-averse objectives**
- Introduce **correlations** or **balancing market effects**



Various stakeholders approach this problem from different perspectives. Please refine the model specification from the perspectives of (1) the wind-power producer; and (2) the system regulator. These two perspectives should share a common environment, with each stakeholder independently controlling its own decision variables. In the specification, specify the common environment, including the global parameters and variables. For each perspective, also specify the corresponding model and its key components, including the decision variables, transition functions, and objective functions. If there are multiple objectives, provide multiple objective functions instead of a weighted objective function.

Below is a **two-agent refinement of the simulation model**, explicitly separating the perspectives of **(1) the wind-power producer** and **(2) the system regulator**, while embedding both within a **shared market environment**.

The formulation is suitable for simulation, equilibrium analysis, and policy stress-testing.

I. Common Environment (Shared by All Stakeholders)

The **environment** defines the electricity market physics, stochastic processes, and clearing rules. Both stakeholders observe and interact with this environment but control **different decision variables**.

I.1 Time and Index Sets

- Time intervals:

$$t \in \mathcal{T} = \{1, \dots, 24\}$$

- Producers:
 - Conventional: $i \in \mathcal{I} = \{1, 2, 3\}$
 - Solar: s
 - Wind: w

I.2 Exogenous Stochastic Processes

These are realized by the environment during simulation.

Demand

$$D_t \sim \mathcal{N}(\mu_D, \sigma_D^2)$$

Wind generation

$$G_t \sim \mathcal{N}(\mu_G, \sigma_G^2)$$

Conventional bids

$$p_{it} \sim \mathcal{N}(\mu_{pi}, \sigma_{pi}^2), \quad b_{it} = \bar{b}_i$$

Solar bids

$$b_{st} = \max\left(0, a + b \cos\left(\frac{2\pi t}{24}\right)\right)$$

$$p_{st} \sim \mathcal{N}(\mu_{ps}, \sigma_{ps}^2)$$

I.3 Global Parameters

- μ_D, σ_D : demand statistics
- μ_G, σ_G : wind output statistics
- \bar{b}_i : conventional capacities
- a, b : solar production profile parameters
- q_u : penalty rate for under-delivery
- Market rule: **uniform-price merit-order dispatch**

I.4 Market-Clearing Transition Function

For each hour t :

1. Aggregate bids:

$$\mathcal{B}_t = \{(b_{it}, p_{it})\}_{i \in \mathcal{I}} \cup (b_{st}, p_{st}) \cup (b_{wt}, p_{wt})$$

2. Sort bids by ascending price.
3. Determine clearing price:

$$c_t = \inf \left\{ p : \sum_{j: p_{jt} \leq p} b_{jt} \geq D_t \right\}$$

4. Dispatch rule:

$$d_{jt} = \begin{cases} b_{jt}, & p_{jt} \leq c_t \\ 0, & \text{otherwise} \end{cases}$$

I.5 Shared State Variables

At time t , the environment state is:

$$S_t = (D_t, c_t, \{d_{jt}\}_j, G_t)$$

II. Perspective 1: Wind-Power Producer

The wind producer is a **profit-maximizing price-taking agent** facing production uncertainty and imbalance penalties.

II.1 Decision Variables (Controlled by Wind Producer)

Chosen day-ahead:

$$\{b_{wt}, p_{wt}\}_{t=1}^{24}$$

II.2 Local Transition Functions

Given environment outcomes:

- Dispatch:

$$d_{wt} = \begin{cases} b_{wt}, & p_{wt} \leq c_t \\ 0, & \text{otherwise} \end{cases}$$

- Under-delivery:

$$u_t = \max(0, d_{wt} - G_t)$$

II.3 Revenue and Cost

Hourly profit:

$$\pi_t = c_t d_{wt} - q_u u_t$$

II.4 Objective Function (Wind Producer)

Objective 1: Expected profit maximization

$$\max_{\{b_{wt}, p_{wt}\}} \mathbb{E} \left[\sum_{t=1}^{24} \pi_t \right]$$

Objective 2 (optional, risk-aware):

Minimize profit volatility:

$$\min \text{Var} \left(\sum_{t=1}^{24} \pi_t \right)$$

(Objectives may be analyzed via Pareto front rather than weighted sum.)

III. Perspective 2: System Regulator

The regulator is a **non-profit agent** concerned with system reliability, efficiency, and market outcomes.

III.1 Decision Variables (Controlled by Regulator)

- Penalty rate:

$$q_u \geq 0$$

- (Optional extensions)
 - Price caps $c_t \leq \bar{c}$
 - Renewable priority rules
 - Capacity adequacy constraints

III.2 Regulator-Controlled Transitions

The regulator does **not** clear the market but **modifies incentives** by:

- Setting imbalance penalties
- Enforcing dispatch and settlement rules

These parameters affect agents' bidding behavior and realized shortfalls.

III.3 System-Level Performance Metrics

Aggregate imbalance:

$$U_t = \sum_j \max(0, d_{jt} - G_{jt})$$

Renewable curtailment:

$$C_t = \max(0, G_t - d_{wt})$$

Market price volatility:

$$\text{Var}(c_t)$$

III.4 Objective Functions (Regulator)

Objective 1: Minimize system imbalance

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} U_t \right]$$

Objective 2: Minimize total system cost

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} \left(\sum_j c_t d_{jt} + q_u U_t \right) \right]$$

Objective 3: Promote renewable participation

$$\max_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} d_{wt} \right]$$

IV. Interaction Structure Summary

| Component | Wind Producer | Regulator |
|-----------|------------------------------|-----------------------------|
| Controls | b_{wt}, p_{wt} | q_u |
| Observes | Prices, dispatch, penalties | Imbalances, prices, welfare |
| Objective | Profit & risk | Reliability & efficiency |
| Coupling | Via market price & penalties | Via incentives |

V. Interpretation

- The **wind producer** solves a stochastic decision problem under price and quantity uncertainty.
- The **regulator** solves a policy design problem under behavioral and structural uncertainty.
- Both operate in a **shared stochastic environment**, enabling:
 - Scenario analysis
 - Policy stress testing
 - Investment viability studies
 - Incentive compatibility analysis

If you want, I can next:

- Formulate this as a **bi-level optimization**
- Translate it into a **Markov game**
- Provide **simulation pseudocode**
- Extend to **multiple wind producers or storage integration**



According to the specification, q_u is a decision variable controlled by the system regulator and therefore should not be treated as a global parameter. Please refine your specification.

You are absolutely right — thank you for the correction.

Below is a **refined and internally consistent specification** in which q_u is **no longer a global parameter**, but a **policy decision variable controlled by the system regulator**, while both stakeholders continue to interact through a **shared environment**.

Refined Two-Perspective Model Specification

I. Common Environment (Policy-Dependent)

The **environment** defines the physical market, stochastic processes, and clearing rules, but **does not own policy instruments**.

Instead, it **takes regulator decisions as inputs**.

I.1 Time and Agents

- Time periods:

$$t \in \mathcal{T} = \{1, \dots, 24\}$$

- Agents:

- Wind producer: w
- Solar producer: s
- Conventional producers: $i \in \mathcal{I} = \{1, 2, 3\}$
- System regulator: r

I.2 Exogenous Stochastic Variables

These are generated by the environment independently of agent actions.

Demand

$$D_t \sim \mathcal{N}(\mu_D, \sigma_D^2)$$

Wind generation

$$G_t \sim \mathcal{N}(\mu_G, \sigma_G^2)$$

Conventional producer bids

$$p_{it} \sim \mathcal{N}(\mu_{pi}, \sigma_{pi}^2), \quad b_{it} = \bar{b}_i$$

Solar producer bids

$$b_{st} = \max\left(0, a + b \cos\left(\frac{2\pi t}{24}\right)\right)$$

$$p_{st} \sim \mathcal{N}(\mu_{ps}, \sigma_{ps}^2)$$

I.3 Global Parameters (Fixed)

These parameters are **not controlled by any stakeholder**:

- μ_D, σ_D : demand statistics
- μ_G, σ_G : wind production statistics
- \bar{b}_i : conventional capacities
- a, b : solar production parameters
- Market design: uniform-price merit-order clearing

Removed: q_u is no longer listed here.

I.4 Market-Clearing Transition Function (Environment)

Given bids and demand, for each t :

1. Aggregate bids:

$$\mathcal{B}_t = \{(b_{it}, p_{it})\}_{i \in \mathcal{I}} \cup (b_{st}, p_{st}) \cup (b_{wt}, p_{wt})$$

2. Sort bids by price.
3. Clearing price:

$$c_t = \inf \left\{ p : \sum_{j: p_{jt} \leq p} b_{jt} \geq D_t \right\}$$

4. Dispatch:

$$d_{jt} = \begin{cases} b_{jt}, & p_{jt} \leq c_t \\ 0, & \text{otherwise} \end{cases}$$

I.5 Environment State Variables

At hour t :

$$S_t = (D_t, c_t, \{d_{jt}\}_j, G_t)$$

Penalty outcomes depend on **regulator policy** q_u , not on the environment alone.

II. Wind-Power Producer Perspective

The wind producer is a **price-taking agent** responding to both **market uncertainty** and **regulatory incentives**.

II.1 Decision Variables (Wind Producer)

Chosen day-ahead:

$$\{b_{wt}, p_{wt}\}_{t=1}^{24}$$

II.2 Policy-Dependent Transition Functions

Given environment outcomes and regulator's decision q_u :

- Dispatch:

$$d_{wt} = \begin{cases} b_{wt}, & p_{wt} \leq c_t \\ 0, & \text{otherwise} \end{cases}$$

- Under-delivery:

$$u_t = \max(0, d_{wt} - G_t)$$

II.3 Payoff Function

Hourly profit:

$$\pi_t(q_u) = c_t d_{wt} - q_u u_t$$

II.4 Objective Functions (Wind Producer)

Objective 1: Maximize expected profit

$$\max_{\{b_{wt}, p_{wt}\}} \mathbb{E} \left[\sum_{t=1}^{24} \pi_t(q_u) \right]$$

Objective 2: Minimize downside risk

$$\min \text{Var} \left(\sum_{t=1}^{24} \pi_t(q_u) \right)$$

These objectives are evaluated **conditional on the regulator's chosen q_u** .

III. System Regulator Perspective

The regulator **does not bid or dispatch**.
Instead, it **sets market incentives** that influence producer behavior.

III.1 Decision Variables (Regulator)

- Imbalance penalty:

$$q_u \in [0, \bar{q}]$$

(Optionally extended to other policy instruments.)

III.2 Regulator-Influenced Quantities

Given q_u and producer behavior:

- Total system imbalance:

$$U_t = \sum_j \max(0, d_{jt} - G_{jt})$$

- Wind shortfall:

$$u_t = \max(0, d_{wt} - G_t)$$

- Price volatility:

$$\text{Var}(c_t)$$

III.3 Objective Functions (Regulator)

Objective 1: Minimize expected system imbalance

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} U_t \right]$$

Objective 2: Minimize total expected system cost

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} \left(\sum_j c_t d_{jt} + q_u U_t \right) \right]$$

Objective 3: Support renewable participation

$$\max_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} d_{wt} \right]$$

IV. Corrected Interaction Structure

| Component | Wind Producer | Regulator | Environment |
|----------------|------------------|-----------------------|------------------|
| Controls | b_{wt}, p_{wt} | q_u | — |
| Takes as given | q_u, c_t | Producer responses | Market rules |
| Generates | Bids | Incentives | Prices, dispatch |
| Objective | Profit & risk | Reliability & welfare | Feasibility |

V. Interpretation and Consistency Check

- q_u is now **clearly a strategic policy choice**, not a fixed parameter.
- The environment remains **policy-neutral** but **policy-responsive**.
- The model can be interpreted as:
 - a **Stackelberg game** (regulator \rightarrow producers), or
 - a **co-evolutionary simulation** under deep uncertainty.

If you'd like, I can next:

- Explicitly write this as a **bi-level optimization problem**
- Convert it into a **Markov (stochastic) game**
- Discuss **identifiability and calibration** issues
- Extend the regulator's controls to **price caps or reserve markets**



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 decisions and objectives. The composed model should: 1. embed each perspective's decision variables and objectives; 2. enable modular evaluation of each perspective; 3. for each perspective, treat other perspectives' decisions as exogenous inputs; and 4. incorporate all other components from every perspective.

Below is a **single, unified (composed) model** that embeds both perspectives—**wind-power producer** and **system regulator**—into one internally consistent framework. The model shares a **common environment, state space, and transition dynamics**, while allowing **modular, perspective-specific evaluation** in which each agent treats the other's decisions as **exogenous inputs**.

Unified Composed Model for Wind-Power Market Simulation

0. Modeling Paradigm

The composed model is a **multi-agent stochastic control system with a shared environment**:

- **Agents:**
 - Wind producer w
 - System regulator r
- **Key property:**
Each agent optimizes **its own objectives**, while taking the **other agent's decisions as fixed inputs**.

This enables:

- independent policy evaluation,
- Stackelberg-style analysis,
- scenario-based simulation under deep uncertainty.

I. Common Environment (Shared Across Perspectives)

The environment defines **market rules, stochastic processes, and physical feasibility**. It does **not** optimize; it **evolves states** given agent decisions and uncertainty.

I.1 Time and Index Sets

- Time:

$$t \in \mathcal{T} = \{1, \dots, 24\}$$

- Producers:
 - Wind: w
 - Solar: s
 - Conventional: $i \in \mathcal{I} = \{1, 2, 3\}$

I.2 Exogenous Stochastic Processes

Sampled independently by the environment.

- Demand:

$$D_t \sim \mathcal{N}(\mu_D, \sigma_D^2)$$

- Wind generation:

$$G_t \sim \mathcal{N}(\mu_G, \sigma_G^2)$$

- Conventional bids:

$$p_{it} \sim \mathcal{N}(\mu_{pi}, \sigma_{pi}^2), \quad b_{it} = \bar{b}_i$$

- Solar bids:

$$b_{st} = \max\left(0, a + b \cos\left(\frac{2\pi t}{24}\right)\right)$$

$$p_{st} \sim \mathcal{N}(\mu_{ps}, \sigma_{ps}^2)$$

I.3 Global (Non-Strategic) Parameters

These are fixed and not controlled by any perspective:

- μ_D, σ_D : demand statistics
- μ_G, σ_G : wind statistics
- \bar{b}_i : conventional capacities
- a, b : solar profile parameters
- Market design: uniform-price, merit-order dispatch

I.4 Shared State Space

At each hour t , the **global state** is:

$$S_t = (D_t, c_t, \{d_{jt}\}_{j \in \{w, s, \mathcal{I}\}}, G_t)$$

The state is **fully observable** to all perspectives.

I.5 Environment Transition Function

Given:

- wind bids (b_{wt}, p_{wt}) ,
- regulator policy q_u ,
- stochastic realizations,

the environment performs:

(a) Market Clearing

$$c_t = \inf \left\{ p : \sum_{j: p_{jt} \leq p} b_{jt} \geq D_t \right\}$$

(b) Dispatch

$$d_{jt} = \begin{cases} b_{jt}, & p_{jt} \leq c_t \\ 0, & \text{otherwise} \end{cases}$$

(c) Shortfall Accounting

$$u_{jt} = \max(0, d_{jt} - G_{jt})$$

II. Embedded Perspective 1: Wind-Power Producer

The wind producer is embedded as a **decision-making module** within the unified model.

II.1 Decision Variables (Controlled by Wind Producer)

Day-ahead bids:

$$x_w = \{(b_{wt}, p_{wt})\}_{t=1}^{24}$$

II.2 Exogenous Inputs (from Other Perspectives)

From the wind producer's viewpoint, **taken as given**:

- Regulator policy q_u
- Market clearing rules
- Other producers' bids
- All stochastic processes

II.3 Wind-Specific Payoff Mapping

Hourly profit:

$$\pi_{wt}(S_t, q_u) = c_t d_{wt} - q_u \max(0, d_{wt} - G_t)$$

II.4 Wind Producer Objectives

The wind module can be evaluated against multiple objectives:

Objective W1: Expected profit

$$\max_{x_w} \mathbb{E} \left[\sum_{t=1}^{24} \pi_{wt}(S_t, q_u) \right]$$

Objective W2: Profit risk

$$\min_{x_w} \text{Var} \left(\sum_{t=1}^{24} \pi_{wt}(S_t, q_u) \right)$$

III. Embedded Perspective 2: System Regulator

The regulator is embedded as a **policy-setting module**.

III.1 Decision Variables (Controlled by Regulator)

- Imbalance penalty:

$$x_r = q_u \in [0, \bar{q}]$$

III.2 Exogenous Inputs (from Other Perspectives)

From the regulator's viewpoint, **taken as given**:

- Wind bidding strategy (b_{wt}, p_{wt})
- Market structure and clearing rules
- Stochastic demand and generation

III.3 Regulator Performance Metrics

Derived from the shared state:

- Total system imbalance:

$$U_t = \sum_j \max(0, d_{jt} - G_{jt})$$

- Renewable dispatch:

$$R_t = d_{wt}$$

- Market price volatility:

$$\text{Var}(c_t)$$

III.4 Regulator Objectives

Objective R1: Reliability

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} U_t \right]$$

Objective R2: System cost

$$\min_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} \left(\sum_j c_t d_{jt} + q_u U_t \right) \right]$$

Objective R3: Renewable participation

$$\max_{q_u} \mathbb{E} \left[\sum_{t=1}^{24} R_t \right]$$

IV. Composed Model Structure

The unified model can be summarized as:

$$\mathcal{M} = \left(\mathcal{S}, \mathcal{X}_w, \mathcal{X}_r, \mathcal{W}, f, \Pi_w, \Pi_r \right)$$

Where:

- \mathcal{S} : shared state space
- \mathcal{X}_w : wind decisions
- \mathcal{X}_r : regulator decisions
- \mathcal{W} : stochastic processes
- f : environment transition function
- Π_w : wind objectives
- Π_r : regulator objectives

V. How the Composition Meets the Requirements

| Requirement | Satisfied by |
|--------------------------------------|--------------------------------------|
| Shared environment and states | Section I |
| Embedded decisions & objectives | Sections II & III |
| Modular evaluation | Each agent has its own objective map |
| Other decisions treated as exogenous | Explicit in II.2 & III.2 |
| Full component integration | Single transition function |

VI. Interpretation

- The model can be simulated **under fixed** q_u to evaluate wind profitability.
- It can be simulated **under fixed wind strategies** to evaluate regulatory policy.
- It supports:
 - Monte Carlo simulation
 - Stackelberg analysis
 - Multi-objective Pareto exploration
 - Deep uncertainty stress testing

If you want, I can next:

- Express this as a **formal Markov game**
- Provide **simulation pseudocode**
- Show **policy-response surfaces**
- Extend to **multiple wind producers or storage**



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 take as inputs a dictionary of decision variable values, a dictionary of uncertain parameter values, and a dictionary of constant parameter values; simulate the model dynamics; and return the value of each objective function. This function does not need to be executed.

Below is a **modular, object-oriented Python implementation** of the **unified composed model**. The code is written for **clarity and extensibility**, not execution efficiency, and directly mirrors the formal specification.

- Each **perspective** is a class with its own decisions and objectives.
- The **environment** is shared and owns the stochastic dynamics and market clearing.
- The **interface function** at the end composes everything and returns **all objectives**, enabling modular evaluation.

python

```
import numpy as np
from typing import Dict, Any, List

# =====
# Common Environment
# =====

class MarketEnvironment:
    """
    Shared environment: stochastic processes, market clearing,
    and state transitions.
    """

    def __init__(self, constants: Dict[str, Any], uncertainties: Dict[str, Any]):
        self.constants = constants
        self.uncertainties = uncertainties

    def sample_demand(self, t: int) -> float:
        return np.random.normal(
            self.uncertainties["mu_D"],
            self.uncertainties["sigma_D"]
        )

    def sample_wind_generation(self, t: int) -> float:
        return np.random.normal(
            self.uncertainties["mu_G"],
            self.uncertainties["sigma_G"]
        )

    def sample_conventional_bids(self, t: int):
        bids = []
        for i in range(self.constants["n_conventional"]):
            p = np.random.normal(
                self.uncertainties["mu_pi"][i],
                self.uncertainties["sigma_pi"][i]
            )
            b = self.constants["b_i"][i]
            bids.append({"agent": f"conv_{i}", "b": b, "p": p})
        return bids
```

```

def sample_solar_bid(self, t: int):
    a = self.constants["solar_a"]
    b = self.constants["solar_b"]
    quantity = max(0.0, a + b * np.cos(2 * np.pi * t / 24))
    price = np.random.normal(
        self.uncertainties["mu_ps"],
        self.uncertainties["sigma_ps"]
    )
    return {"agent": "solar", "b": quantity, "p": price}

def clear_market(self, bids: List[Dict[str, float]], demand: float):
    """
    Merit-order clearing with uniform pricing.
    """
    bids_sorted = sorted(bids, key=lambda x: x["p"])
    supplied = 0.0
    clearing_price = None
    dispatch = {}

    for bid in bids_sorted:
        supplied += bid["b"]
        dispatch[bid["agent"]] = bid["b"]
        clearing_price = bid["p"]
        if supplied >= demand:
            break

    # Non-accepted bids
    for bid in bids_sorted:
        if bid["p"] > clearing_price:
            dispatch[bid["agent"]] = 0.0

    return clearing_price, dispatch

# =====
# Wind Producer Perspective
# =====

class WindProducer:
    """
    Wind producer decision module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.bids = decisions["wind_bids"] # dict: t -> (b_wt, p_wt)

    def submit_bid(self, t: int):
        b, p = self.bids[t]
        return {"agent": "wind", "b": b, "p": p}

    def profit(self, clearing_price, dispatch, wind_generation, penalty):
        d_w = dispatch.get("wind", 0.0)
        shortfall = max(0.0, d_w - wind_generation)
        return clearing_price * d_w - penalty * shortfall

# =====
# Regulator Perspective
# =====

class SystemRegulator:
    """
    Regulator policy module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.q_u = decisions["q_u"]

    def penalty(self):
        return self.q_u

# =====
# Unified Simulation

```

```
# =====

def simulate_unified_model(
    decisions: Dict[str, Any],
    uncertainties: Dict[str, Any],
    constants: Dict[str, Any],
    n_mc: int = 1000
) -> Dict[str, float]:
    """
    Unified interface function.

    Inputs
    -----
    decisions : dict
        {
            "wind_bids": {t: (b_wt, p_wt)},
            "q_u": penalty rate
        }

    uncertainties : dict
        Distributional parameters for demand, bids, and wind.

    constants : dict
        Fixed market and technology parameters.

    n_mc : int
        Number of Monte Carlo samples.

    Returns
    -----
    objectives : dict
        Values of all objective functions.
    """

    env = MarketEnvironment(constants, uncertainties)
    wind = WindProducer(decisions)
    regulator = SystemRegulator(decisions)

    wind_profits = []
    system_imbalances = []
    system_costs = []
    wind_dispatch_total = []

    for _ in range(n_mc):
        daily_profit = 0.0
        daily_imbalance = 0.0
        daily_cost = 0.0
        daily_wind_dispatch = 0.0

        for t in range(1, 25):
            demand = env.sample_demand(t)
            wind_gen = env.sample_wind_generation(t)

            bids = []
            bids.extend(env.sample_conventional_bids(t))
            bids.append(env.sample_solar_bid(t))
            bids.append(wind.submit_bid(t))

            price, dispatch = env.clear_market(bids, demand)

            # Wind profit
            profit = wind.profit(
                price,
                dispatch,
                wind_gen,
                regulator.penalty()
            )
            daily_profit += profit

            # System imbalance and cost
            for agent, d in dispatch.items():
                if agent == "wind":
                    imbalance = max(0.0, d - wind_gen)
                else:
                    imbalance = 0.0 # conventional assumed firm
```



```

        daily_imbalance += imbalance
        daily_cost += price * d + regulator.penalty() * imbalance

    daily_wind_dispatch += dispatch.get("wind", 0.0)

    wind_profits.append(daily_profit)
    system_imbalances.append(daily_imbalance)
    system_costs.append(daily_cost)
    wind_dispatch_total.append(daily_wind_dispatch)

# =====
# Objective Evaluation
# =====






objectives = {
    # Wind producer objectives
    "wind_expected_profit": float(np.mean(wind_profits)),
    "wind_profit_variance": float(np.var(wind_profits)),

    # Regulator objectives
    "expected_system_imbalance": float(np.mean(system_imbalances)),
    "expected_system_cost": float(np.mean(system_costs)),
    "expected_wind_dispatch": float(np.mean(wind_dispatch_total)),
}

return objectives

```

What this implementation guarantees

-  **Single shared environment and state evolution**
-  **Perspective-specific decision modules**
-  **Each agent treats other decisions as exogenous**
-  **Multiple objectives returned without aggregation**
-  **Plug-and-play extensibility** (storage, multiple wind firms, price caps)

If you want next steps, I can:

- Add **risk measures (CVaR)**
- Convert this into a **bi-level Stackelberg solver**
- Introduce **correlated uncertainties**
- Add **learning or adaptive bidding strategies**



You are not asked to perform multiple Monte Carlo samples in the simulation function. Please remove part of code.

You're right — thank you for the clarification.

Below is the **refined Python implementation** with **all Monte Carlo logic removed**. The simulation now performs **a single stochastic realization** of the unified model and returns the value of **each objective function** for that run.

The modular structure and unified composition are preserved.

Refined Modular Python Implementation (Single Simulation Run)

```
python

import numpy as np
from typing import Dict, Any, List

# =====
# Common Environment
# =====

class MarketEnvironment:
    """
    Shared environment: stochastic processes, market clearing,
    and state transitions.
    """

    def __init__(self, constants: Dict[str, Any], uncertainties: Dict[str, Any]):
        self.constants = constants
        self.uncertainties = uncertainties

    def sample_demand(self, t: int) -> float:
        return np.random.normal(
            self.uncertainties["mu_D"],
            self.uncertainties["sigma_D"]
        )

    def sample_wind_generation(self, t: int) -> float:
        return np.random.normal(
            self.uncertainties["mu_G"],
            self.uncertainties["sigma_G"]
        )

    def sample_conventional_bids(self, t: int):
        bids = []
        for i in range(self.constants["n_conventional"]):
            p = np.random.normal(
                self.uncertainties["mu_pi"][i],
                self.uncertainties["sigma_pi"][i]
            )
            b = self.constants["b_i"][i]
            bids.append({"agent": f"conv_{i}", "b": b, "p": p})
        return bids

    def sample_solar_bid(self, t: int):
        a = self.constants["solar_a"]
        b = self.constants["solar_b"]
        quantity = max(0.0, a + b * np.cos(2 * np.pi * t / 24))
        price = np.random.normal(
```

```

        self.uncertainties["mu_ps"],
        self.uncertainties["sigma_ps"]
    )
    return {"agent": "solar", "b": quantity, "p": price}

def clear_market(self, bids: List[Dict[str, float]], demand: float):
    """
    Merit-order clearing with uniform pricing.
    """
    bids_sorted = sorted(bids, key=lambda x: x["p"])
    supplied = 0.0
    clearing_price = None
    dispatch = {}

    for bid in bids_sorted:
        if supplied < demand:
            supplied += bid["b"]
            dispatch[bid["agent"]] = bid["b"]
            clearing_price = bid["p"]
        else:
            dispatch[bid["agent"]] = 0.0

    return clearing_price, dispatch

# =====
# Wind Producer Perspective
# =====

class WindProducer:
    """
    Wind producer decision module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.bids = decisions["wind_bids"] # dict: t -> (b_wt, p_wt)

    def submit_bid(self, t: int):
        b, p = self.bids[t]
        return {"agent": "wind", "b": b, "p": p}

    def profit(self, clearing_price, dispatch, wind_generation, penalty):
        d_w = dispatch.get("wind", 0.0)
        shortfall = max(0.0, d_w - wind_generation)
        return clearing_price * d_w - penalty * shortfall

# =====
# System Regulator Perspective
# =====

class SystemRegulator:
    """
    Regulator policy module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.q_u = decisions["q_u"]

    def penalty(self):
        return self.q_u

# =====
# Unified Simulation Interface (Single Run)
# =====

def simulate_unified_model(
    decisions: Dict[str, Any],
    uncertainties: Dict[str, Any],
    constants: Dict[str, Any]
) -> Dict[str, float]:
    """
    Unified interface function.
    """

```

```

Inputs
-----
decisions : dict
    {
        "wind_bids": {t: (b_wt, p_wt)},
        "q_u": imbalance penalty
    }

uncertainties : dict
    Distributional parameters for demand, bids, and wind.

constants : dict
    Fixed market and technology parameters.

Returns
-----
objectives : dict
    Realized values of all objective functions for one simulation.
"""

env = MarketEnvironment(constants, uncertainties)
wind = WindProducer(decisions)
regulator = SystemRegulator(decisions)

total_profit = 0.0
total_imbalance = 0.0
total_system_cost = 0.0
total_wind_dispatch = 0.0

for t in range(1, 25):
    demand = env.sample_demand(t)
    wind_gen = env.sample_wind_generation(t)

    bids = []
    bids.extend(env.sample_conventional_bids(t))
    bids.append(env.sample_solar_bid(t))
    bids.append(wind.submit_bid(t))

    price, dispatch = env.clear_market(bids, demand)

    # Wind producer profit
    profit = wind.profit(
        price,
        dispatch,
        wind_gen,
        regulator.penalty()
    )
    total_profit += profit

    # System-level quantities
    for agent, d in dispatch.items():
        if agent == "wind":
            imbalance = max(0.0, d - wind_gen)
        else:
            imbalance = 0.0 # conventional and solar assumed firm

        total_imbalance += imbalance
        total_system_cost += price * d + regulator.penalty() * imbalance

    total_wind_dispatch += dispatch.get("wind", 0.0)

# =====
# Objective Values (Single Realization)
# =====





objectives = {
    # Wind producer objectives
    "wind_profit": total_profit,

    # Regulator objectives
    "system_imbalance": total_imbalance,
    "system_cost": total_system_cost,
    "wind_dispatch": total_wind_dispatch,
}

```

return objectives

What changed (and why)

-  Removed Monte Carlo loop and averaging
-  One stochastic realization per function call
-  Objective values are **realized outcomes**, not expectations
-  Fully consistent with:
 - modular perspectives
 - shared environment
 - exogenous treatment of other agents' decisions

This structure is ideal for:

- outer-loop Monte Carlo or scenario analysis
- policy sweeps over q_u
- optimization or robust decision-making frameworks

If you'd like, I can next:

- Wrap this in a **scenario generator**
- Add **deterministic mode** (no randomness)
- Refactor into a **Markov game interface**
- Add **unit tests or validation checks**



According to the specification, μ_D , σ_D , μ_G and σ_G are constants. Please refine your implementation.

Good catch — this is an **internal-consistency issue**, and your interpretation is correct.

Per the specification:

- μ_D, σ_D (demand moments)
- μ_G, σ_G (wind generation moments)

are **constants**, not uncertainties.

They should therefore be passed via the **constants dictionary**, not the **uncertainties dictionary**.

Below is the **refined implementation**, with **only the necessary changes applied**.

No other logic is altered.

Refined Python Implementation

(with μ_D , σ_D , μ_G , σ_G treated as constants)

```
python

import numpy as np
from typing import Dict, Any, List

# =====
# Common Environment
# =====

class MarketEnvironment:
    """
    Shared environment: stochastic processes, market clearing,
    and state transitions.
    """

    def __init__(self, constants: Dict[str, Any], uncertainties: Dict[str, Any]):
        self.constants = constants
        self.uncertainties = uncertainties

    def sample_demand(self, t: int) -> float:
        return np.random.normal(
            self.constants["mu_D"],
            self.constants["sigma_D"]
        )

    def sample_wind_generation(self, t: int) -> float:
        return np.random.normal(
            self.constants["mu_G"],
            self.constants["sigma_G"]
        )

    def sample_conventional_bids(self, t: int):
        bids = []
        for i in range(self.constants["n_conventional"]):
            p = np.random.normal(
                self.uncertainties["mu_pi"][i],
```

```

        self.uncertainties["sigma_pi"][i]
    )
    b = self.constants["b_i"][i]
    bids.append({"agent": f"conv_{i}", "b": b, "p": p})
    return bids

def sample_solar_bid(self, t: int):
    a = self.constants["solar_a"]
    b = self.constants["solar_b"]
    quantity = max(0.0, a + b * np.cos(2 * np.pi * t / 24))
    price = np.random.normal(
        self.uncertainties["mu_ps"],
        self.uncertainties["sigma_ps"]
    )
    return {"agent": "solar", "b": quantity, "p": price}

def clear_market(self, bids: List[Dict[str, float]], demand: float):
    """
    Merit-order clearing with uniform pricing.
    """
    bids_sorted = sorted(bids, key=lambda x: x["p"])
    supplied = 0.0
    clearing_price = None
    dispatch = {}

    for bid in bids_sorted:
        if supplied < demand:
            supplied += bid["b"]
            dispatch[bid["agent"]] = bid["b"]
            clearing_price = bid["p"]
        else:
            dispatch[bid["agent"]] = 0.0

    return clearing_price, dispatch

```

```

# =====
# Wind Producer Perspective
# =====

```

```

class WindProducer:
    """
    Wind producer decision module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.bids = decisions["wind_bids"] # dict: t -> (b_wt, p_wt)

    def submit_bid(self, t: int):
        b, p = self.bids[t]
        return {"agent": "wind", "b": b, "p": p}

    def profit(self, clearing_price, dispatch, wind_generation, penalty):
        d_w = dispatch.get("wind", 0.0)
        shortfall = max(0.0, d_w - wind_generation)
        return clearing_price * d_w - penalty * shortfall

```

```

# =====
# System Regulator Perspective
# =====

```

```

class SystemRegulator:
    """
    Regulator policy module.
    """

    def __init__(self, decisions: Dict[str, Any]):
        self.q_u = decisions["q_u"]

    def penalty(self):
        return self.q_u

```

```

# =====

```

```
# Unified Simulation Interface (Single Run)
# =====

def simulate_unified_model(
    decisions: Dict[str, Any],
    uncertainties: Dict[str, Any],
    constants: Dict[str, Any]
) -> Dict[str, float]:
    """
    Unified interface function.

    Inputs
    -----
    decisions : dict
        {
            "wind_bids": {t: (b_wt, p_wt)},
            "q_u": imbalance penalty
        }

    uncertainties : dict
        Distributional parameters for bidding behavior
        (e.g., mu_pi, sigma_pi, mu_ps, sigma_ps).

    constants : dict
        Fixed market and technology parameters, INCLUDING
        mu_D, sigma_D, mu_G, sigma_G.

    Returns
    -----
    objectives : dict
        Realized values of all objective functions for one simulation.
    """

    env = MarketEnvironment(constants, uncertainties)
    wind = WindProducer(decisions)
    regulator = SystemRegulator(decisions)

    total_profit = 0.0
    total_imbalance = 0.0
    total_system_cost = 0.0
    total_wind_dispatch = 0.0

    for t in range(1, 25):
        demand = env.sample_demand(t)
        wind_gen = env.sample_wind_generation(t)

        bids = []
        bids.extend(env.sample_conventional_bids(t))
        bids.append(env.sample_solar_bid(t))
        bids.append(wind.submit_bid(t))

        price, dispatch = env.clear_market(bids, demand)

        # Wind producer profit
        profit = wind.profit(
            price,
            dispatch,
            wind_gen,
            regulator.penalty()
        )
        total_profit += profit

        # System-level quantities
        for agent, d in dispatch.items():
            if agent == "wind":
                imbalance = max(0.0, d - wind_gen)
            else:
                imbalance = 0.0 # conventional and solar assumed firm

            total_imbalance += imbalance
            total_system_cost += price * d + regulator.penalty() * imbalance

        total_wind_dispatch += dispatch.get("wind", 0.0)

    # =====
```



```
# Objective Values (Single Realization)
# =====

objectives = {
    # Wind producer objective
    "wind_profit": total_profit,

    # Regulator objectives
    "system_imbalance": total_imbalance,
    "system_cost": total_system_cost,
    "wind_dispatch": total_wind_dispatch,
}

return objectives
```

Summary of the Refinement

✓ What changed

- `mu_D`, `sigma_D`, `mu_G`, `sigma_G`
→ moved from **uncertainties** to **constants**
- Demand and wind generation sampling now uses **constants**

✓ What did NOT change

- Market clearing logic
- Perspective modularity
- Interface signature
- Objective definitions

This version is now **fully consistent with the formal specification** and cleanly separates:

- **deep uncertainties** (bidding behavior),
- **constants** (system statistics),
- **decisions** (bids and policy).

If you want, I can next:

- Enforce **non-negativity / truncation** on normals
- Add **correlation structures**
- Provide a **JSON schema** for inputs
- Show how to wrap this in an **outer-loop optimizer**



Please refine your implementation so that any bid with a price less than or equal to the clearing price is accepted, even if the total market demand has already been met.

Good point — this is a **merit-order consistency fix**.

Per the specification, **acceptance depends only on price**, not on whether demand is already met. So the correct rule is:

All bids with $p_{jt} \leq c_t$ are accepted at full quantity, even if total accepted supply exceeds demand.

Below is the **refined implementation**, with the **minimal necessary change** applied to the market-clearing logic. Everything else is left untouched.

✓ Corrected Market-Clearing Logic

Key idea

1. **First pass**: determine the clearing price c_t as the marginal price where cumulative supply meets demand.
2. **Second pass**: accept **all bids with price $\leq c_t$** at full quantity.

Refined Python Implementation (Correct Merit Order)

```
python

import numpy as np
from typing import Dict, Any, List

# =====
# Common Environment
# =====

class MarketEnvironment:
    """
    Shared environment: stochastic processes, market clearing,
    and state transitions.
    """

    def __init__(self, constants: Dict[str, Any], uncertainties: Dict[str, Any]):
        self.constants = constants
        self.uncertainties = uncertainties

    def sample_demand(self, t: int) -> float:
        return np.random.normal(
            self.constants["mu_D"],
            self.constants["sigma_D"]
        )

    def sample_wind_generation(self, t: int) -> float:
        return np.random.normal(
```

```

        self.constants["mu_G"],
        self.constants["sigma_G"]
    )

def sample_conventional_bids(self, t: int):
    bids = []
    for i in range(self.constants["n_conventional"]):
        p = np.random.normal(
            self.uncertainties["mu_pi"][i],
            self.uncertainties["sigma_pi"][i]
        )
        b = self.constants["b_i"][i]
        bids.append({"agent": f"conv_{i}", "b": b, "p": p})
    return bids

def sample_solar_bid(self, t: int):
    a = self.constants["solar_a"]
    b = self.constants["solar_b"]
    quantity = max(0.0, a + b * np.cos(2 * np.pi * t / 24))
    price = np.random.normal(
        self.uncertainties["mu_ps"],
        self.uncertainties["sigma_ps"]
    )
    return {"agent": "solar", "b": quantity, "p": price}

def clear_market(self, bids: List[Dict[str, float]], demand: float):
    """
    Merit-order clearing with uniform pricing.

    Any bid with price <= clearing price is accepted in full,
    even if total accepted supply exceeds demand.
    """
    bids_sorted = sorted(bids, key=lambda x: x["p"])

    # -----
    # Step 1: Determine clearing price
    # -----
    cumulative_supply = 0.0
    clearing_price = None

    for bid in bids_sorted:
        cumulative_supply += bid["b"]
        if cumulative_supply >= demand:
            clearing_price = bid["p"]
            break

    # -----
    # Step 2: Accept all bids with p <= clearing price
    # -----
    dispatch = {}
    for bid in bids_sorted:
        if bid["p"] <= clearing_price:
            dispatch[bid["agent"]] = bid["b"]
        else:
            dispatch[bid["agent"]] = 0.0

    return clearing_price, dispatch

```

✓ What This Fix Achieves

- ✓ Fully consistent with the **uniform-price merit-order rule**
- ✓ Allows **over-supply**, as intended in the specification
- ✓ Ensures **price alone** determines acceptance
- ✓ Removes implicit rationing artifacts

Why This Matters Conceptually

Without this fix, the model:

- implicitly introduces **quantity rationing**, and
- distorts incentives for **low-price renewable bids**.

With the corrected logic:

- wind and solar are rewarded for bidding low,
- clearing prices reflect **true marginal cost**, and
- imbalance penalties correctly capture over-commitment risk.

If you'd like next steps, I can:

- Add **explicit curtailment accounting**
- Separate **dispatch vs. settlement quantities**
- Introduce **negative prices**
- Validate the clearing logic with a **numerical example**