Multi-Period Compliance Mean Field Game with Deep FBSDE Solver

Orange Ao

October 10, 2024

This is a research report giving big pictures about:

1. *the problem we aim to solve,*
2. *the key methods/algorithms we propose,*
3. *the main results we get, and*
4. *comparisons between different methods and consequent results.*

Visit my GitHub Repository for more math, algorithm details, and code instructions[1].

[1] Please start an issue at GitHub or reach out to me if you find any problems.

Abstract

This work aims to extend the single-period compliance model in [1] to multi-period, proposing several tricks to improve the numeric stability of the deep solver for FBSDEs with jumps. First by reproducing the aforementioned research by Campbell Steven, et al. (2021), then by considering an additional period to the original model, we make comparisons between long/short-term perspectives when it comes to multi-period production decision-making in renewable electricity certificate markets, as well as between different numeric tricks when it comes to algorithm stability. Meanwhile, some practical takeaways on parameter-tuning are recorded.

1 Problem Overview

Conventional numerical solvers are hard-pressed to solve PA-MFG with market-clearing conditions, which may be faced with the "curse of dimensionality" (Bellman 1957)[2]. Thus in their study [1], Professor Campbell and his fellows proposed an actor-critic approach to optimization, where the agents form a Nash equilibria according to the principal's penalty function and the principal evaluates the resulting equilibria. They apply this approach to a stylized PA problem arising in Renewable Energy Certificate (REC) markets, where agents may *work* overtime (or *rent* capacity), *trade* RECs, and *expand* their long-term capacity to navigate the market at maximum profit. Here we only discuss the agents' problem in the multi-agent-multi-period scenario.

1.1 REC Market Basics

Closely related to carbon cap-and-trade (C&T) markets, REC markets are a type of market-based emissions regulation policies, which are motivating real-world applications of FBSDEs in modeling PA-MFG.

In RES markets, a regulator plays the role of principle, setting a floor on the amount of energy generated from renewable resources (aka. green energy) for each firm (based on a percentage of their total production), and providing certificates for each MWh of green energy generated and delivered to the grid. These certificates can be further traded by individual or companies, i.e. agents, to 1) reduce costs or the greenhouse gas (GHG) emissions impact of their operations; and 2) earn profits from the extra inventories instead of wasting. Since the certificates are traded assets, energy suppliers can trade-off between producing clean electricity themselves, and purchasing the certificates on the market. In all, such policies have played an important role in funding clean energy development, particularly in past years when the cost of green power production was not as competitive with the cost of fossil fuel power.

To ensure compliance, each firm must surrender RECs totaling the floor at the end of a compliance period, with a monetary penalty paid for each lacking certificate. In practice, these systems regulate multiple consecutive and disjoint compliance periods, which are linked together through a mechanism called *banking*, where unused allowances in current period can be carried on to the next period (or multiple future periods). Thus, as an extension to the single-period framework [1], we now consider a 2-period model in this report.[3].

1.2 REC Market Modeling with FBSDEs

Let's consider 2 sub-populations here. Before jumping into the 2-period case, we first reproduce the single-period case following steps in [1]. We denote the period end as T, which can be thought of "the end of the world". Referring to the probabilistic method in [2] (R. Carmona, F. Delarue, 2012), one can show that, for agent i

[2] Bellman, R. E.: Dynamic Programming. Princeton University Press, USA (1957).

[3] At a finite set of joint points, the possible lack of differentiability will not have any significant effects.

in sub-population k, the optimal solution to its problem in a *single* period is exactly the solution to the following coupled FBSDEs:

$$\begin{cases} dX_t^i = (h^k + g_t^i + \Gamma_t^i + C_t^i)dt + \sigma^k dW_t^k, & X_0^i \sim \mathcal{N}(v^k, \eta^k) \\ dC_t^i = a_t^i dt, & C_0^i = 0 \\ dY_t^i = Z_t^k dW_t^k, & Y_T^i = w\mathbf{1}_{X_T^i < K}, \end{cases}$$

where: (1)

$$\begin{aligned} Y_t^i &= \mathbb{E}\left[w\mathbf{1}_{X_T^i<K}|\mathcal{F}_t\right] = w\mathbb{P}\left(X_T^i < K \mid \mathcal{F}_t\right) \\ S_t &= \frac{\sum\limits_{k\in\mathcal{K}}\left(\frac{\pi^k}{\gamma^k}\mathbb{E}\left[Y_t^i \mid i \in \mathfrak{N}^k; \mathcal{F}_t\right]\right)}{\sum\limits_{k\in\mathcal{K}}\left(\frac{\pi^k}{\gamma^k}\right)} \\ g_t^k &= \frac{Y_t^k}{\zeta^k} \\ \Gamma_t^k &= \frac{Y_t^k - S_t}{\gamma^k} \end{aligned} \tag{2}$$

Now consider the 2-agent-2-period MFG with market-clearing conditions. Let's denote the 2 compliance periods $[0, T_1]$ and $(T_1, T_2]$ as $\mathfrak{T}_1$ and $\mathfrak{T}_2$, respectively. Here we think of T_2 as "the end of the world", after which there are no costs occurs and all agents forfeit any remaining RECs. Similarly, one can prove that the optimal operation for agent i in sub-population k ($\forall\, i \in \mathfrak{N}_k,\ k \in \{1, 2\}$) can be modeled with the following coupled FBSDEs:

$$\begin{cases} dX_t^i = (h^k + g_t^i + \Gamma_t^i + C_t^i)dt + \sigma^k dW_t^k - \min\left(X_{T_1}^i, K\right)\mathbf{1}_{t=T_1}, & X_0^i = \zeta^i \sim \mathcal{N}(v^k, \eta^k) \\ dC_t^i = a_t^i dt, & C_0^k = 0 \\ dV_t^i = Z_t^{V,k} dW_t^i, & V_{T_1}^i = w * \mathbf{1}_{X_{T_1}^i < K} \\ dU_t^i = Z_t^{U,k} dW_t^i, & U_{T_1}^i = 1 * Y_{T_1}^i \mathbf{1}_{X_{T_1}^i > K} \\ dY_t^i = Z_t^{Y,k} dW_t^i, & Y_{T_2}^i = w * \mathbf{1}_{X_{T_2}^i < K} \quad , \end{cases} \tag{3}$$

where the optimal controls are given by: (4)

$$S_t = \frac{\sum\limits_{k\in\|}\frac{\pi^k}{\gamma^k}\mathbb{E}\left[V_t^i + U_t^i \mid i \in \mathfrak{N}^k; \mathcal{F}_t\right]}{\sum\limits_{k\in\mathcal{K}}(\pi^k/\gamma^k)}\mathbf{1}_{t\in[0,T_1]} + \frac{\sum\limits_{k\in\|}\frac{\pi^k}{\gamma^k}\mathbb{E}\left[Y_t^i \mid i \in \mathfrak{N}^k; \mathcal{F}_t\right]}{\sum\limits_{k\in\mathcal{K}}(\pi^k/\gamma^k)}\mathbf{1}_{t\in(T_1,T_2]} \tag{5}$$

$$g_t^i = \frac{V_t^i + U_t^i}{\zeta^k}\mathbf{1}_{t\in[0,T_1]} + \frac{Y_t^i}{\zeta^k}\mathbf{1}_{t\in(T_1,T_2]} \tag{6}$$

$$\Gamma_t^i = \frac{V_t^i + U_t^i - S_t}{\gamma^k}\mathbf{1}_{t\in[0,T_1]} + \frac{Y_t^i - S_t}{\gamma^k}\mathbf{1}_{t\in(T_1,T_2]} \tag{7}$$

$$a_t^i = \frac{(T_1 - t)(V_t^i + U_t^i) + (T_2 - T_1)Y_t^i}{\beta^k}\mathbf{1}_{t\in[0,T_1]} + \frac{(T_2 - t)Y_t^i}{\beta^k}\mathbf{1}_{t\in(T_1,T_2]} \tag{8}$$

(9)

The key notations/parameters are interpreted as follows:

- $k \in \mathcal{K}$: a sub-population of agents, within which all individuals are assumed to have identical preferences and similar initial conditions/capacities, yet across which are distinct. The sub-population is annotated by

superscript $[\cdot]^k$. Here we only discuss $k = 1, 2$.

- $i \in \mathfrak{N}$: an individual agent belonging to the sub-population $\mathfrak{N}^k$, annotated by superscript $[\cdot]^i$.
- $X_t := (X_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the current inventories in stock. For some key time points:
  - at $t = 0$, there may be some stochastics in the initial inventories, which are assumed to be normally distributed. $X_0^i \sim \mathcal{N}(v^k, \eta^k),\ \forall k \in \mathcal{K},\ \forall i \in \mathfrak{N}^k$.
  - at $t = T_1$, the terminal RECs pre-submission are X_{T_1} carried over from the first period. Shortly after forfeiting $\min\left(K, X_{T_1}^i\right)$, the remaining inventories in stock are $ReLU\left(X_{T_1}^i - K\right)$, which are treated as new initial values for the second period.
  - at $t = T_2$, the terminal RECs pre-submission are $X_{T_2}^i$.
- $I_t := (I_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the integrated inventory generation. We introduce this process for continuous differentiability at T_1. And X_t has the same initial conditions as I_t. We have:

$$X_t = \begin{cases} I_t\,, & t \in [0, T_1] \\ I_t - \min(I_{T_1}, K)\,, & t \in (T_1, T_2] \end{cases} \quad \text{or} \quad X_t = \begin{cases} I_t\,, & t \in [0, T_1] \\ I_t - I_{T_1} + (I_{T_1} - K)_+\,, & t \in (T_1, T_2]. \end{cases} \tag{10}$$

- K: the quota that agents must meet at the end of each compliance period. Fixed to $K = 0.9$[4].
- $P(\cdot)$: the generic penalty function approximated by ***single-knot penalty functions*** [5] :

$$P(x) = w(0.9 - x)_+ \Rightarrow \partial_x P(x) = -w\mathbf{1}_{x<K}.$$

  Further, by tuning the weight w, we can see the relation between the penalty level (controlled by w) and the agents' behaviour, as well as its market impact.
- h: the baseline generation rate at which agents generate inventories with zero marginal cost.
- $C_t := (C_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: incremental REC capacity of agents, i.e. increase of baseline generation rate over time, accumulated by investing in expansion plans - for instance, by installing more solar panels. [6]
- $a_t := (a_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the control of expansion rate, representing long-term REC capacity added per unit time. Note that it could be made even more realistic by incorporating a *delay* between the decision to expand (a_t) and the increase to the baseline rate h.
- $g_t := (g_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the control of overtime-generation rate, i.e. extra capacity achieved by working extra hours and/or renting short-term REC generation capacity at an assumed quadratic cost - specifically, overhour bonus and/or rental fees.
- $\Gamma_t := (\Gamma_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the control of trading rate, with negative[7] values being the amount sold whereas positive purchased per unit of time.

[4] The choice of knot point is associated with h^k and total time span T_1, T_2. A good target (or quota) should be **"attainable"** - neither too easy nor too hard to achieve. Specifically, even if agents do nothing at all, they will have an initial amount plus a baseline generation of inventories - for instance, $0.2 * 1 + 0.6 = 0.8$ for agents in sub-population 1 at the first-period end. Similarly, for sub-population 2, all agents will also have a *"guaranteed"* level of 0.8 for delivery. Thus a target reasonably higher than that, i.e. 0.9, would be regarded **"attainable"**.

[5] See *Report-StepwiseDetail* for more math details.

[6] The incremental capacity over baseline can be carried forward to future periods.

[7] While trading rate may be positive or negative, expansion and overtime-generation rates must be positive.

- $S_t := (S_t)_{t\in\mathfrak{T}_1\cup\mathfrak{T}_2}$: the equilibrium REC price obtained endogenously through market-clearing condition:

$$\lim_{N\to\text{inf}} \frac{1}{N}\sum_{i\in\mathfrak{N}} \Gamma_t^i = 0$$

- $\zeta,\ \gamma,\ \beta$: scalar cost parameters which are identical for agents within the same sub-population.

- π: the proportion of each sub-population:

$$\pi^k = \frac{|\mathfrak{N}^k|}{\sum\limits_{j\in\mathcal{K}} |\mathfrak{N}^j|}.$$

Their values are given in the following table:

	π^k	h^k	σ^k	ζ^k	γ^k	υ^k	η^k	β^k
k=1	0.25	0.2	0.1	1.75	1.25	0.6	0.1	1.0
k=2	0.75	0.5	0.15	1.25	1.75	0.2	0.1	1.0

Table 1: Sub-Population Parameter Values

The framework above can be extended to more realistic models with more than 2 sub-populations and compliance periods, with a penalty approximated by multi-knot functions.

2 Algorithm And Numeric Tricks

2.1 Algorithms: Joint-Optimization Vs. Separate-Optimization

To solve the said FBSDEs in 1.2, we implement the ***shooting method*** with ***Deep Solvers*** [3], discretizing the SDEs in a fine time grid and parameterizing the co-adjoint processes and initial values with neural nets. Let $\mathfrak{T} = \{t_0,\ \dots, t_m\}$ be a discrete set of points with $t_0 = 0$ and $T_m = T$, where m is the number of time steps. Here the step size $dt = (t_i - t_{i-1})$ is a constant and $dt = T/m$. The smaller the value of h, the closer our discretized paths will be to the continuous-time paths we wish to simulate. Certainly, this will be at the expense of greater computational effort. While there are plenty of discretization schemes available, the simplest and most common scheme is the *Euler scheme*, which is intuitive and easy to implement. In particular, it satisfies the *practical decision-making process* - make decisions for the next point of time conditioned on the current information.

The aforementioned ***shooting method*** is implemented by the ***stepwise approximation***: starting from the initial conditions and *"shoot"* for the "correct" terminal conditions - the "correctness" of terminal approximations will be evaluated by computing the aggregated average forward loss/error over the whole population against corresponding targets (denoted as $\mathcal{L}$). For instance, for the single-period case, the aggregated average forward MSE after m iterations is computed as:

$$\mathcal{L}(\theta^{(m)}) = \sum_{i\in\mathfrak{N}} (Y_T^i - w\mathbf{1}_{X_T^i<K})^2,$$

and for the 2-period case:

$$\mathcal{L}(\theta^{(m)}) = \sum_{i\in\mathfrak{N}} (V_{T_1}^i - w\mathbf{1}_{X_{T_1}^i<K})^2 + \sum_{i\in\mathfrak{N}} (U_{T_1}^i - Y_{T_1}^i\mathbf{1}_{X_{T_1}^i>K})^2 + \sum_{i\in\mathfrak{N}} (Y_{T_2}^i - w\mathbf{1}_{X_{T_2}^i<K})^2.$$

The algorithm takes major steps as follows:

1. start from the neural nets for initial values (i.e. Y_0^i etc.);
2. compute the process values at every time step;
3. get approximations to terminal conditions and compute $\mathcal{L}$;
4. compute gradients of $\mathcal{L}$ against parameters(weights and biases, denoted as $\theta^{(m)}$) in the neural nets (i.e. Y_0^i and Z_t^k etc.) and take gradient steps to determine the next set of parameters.

Alternatively, the above steps can be more explicitly displayed by the following pseudocode.

```
if name == "main":
    ## Define global parameters and initialize processes for each sub-pop.
    initx1, initc1, dB1 = ...  ### same for pop2
    learningrate = ...  ### learning rate
    forwardlosses = []  ### list of average forward losses
    MaxBatch= ...  ### number of batches
    OptimSteps= ...  ### number of optimization iterations per batch
    singlebatch = True  ### whether train on a single batch

    ## Ininitialize neural nets, optimizers, and schedulers.
    ...

    ## Training loop
    for batch in [0,MaxBatch):
        sumloss=0
        for iter in [0,OptimSteps):
            loss = getforwardloss()  ### perform the stepwise approximation and get the average los
            loss.backward()  ### compute and record gradients
            optimizer.zerograd()
            optimizer.step()  ### take gradient steps and update model parameters (weights and bias
            schedular.step()  ### adjust learning rate
            sumloss = sumloss + loss.detach().numpy()
        aveloss = sumloss / OptimSteps  ### average loss for current batch over all iterations
        forwardlosses.append(aveloss)

        if not singlebatch:
            ... ### Generate another batch with new initial processes and models.

    ## Visualize the results and model performances:
    ### FwdLoss, InventoryAndPrice, DecompositionInventory, KeyProcesses, TerminalConvergence.
    plotresults(WrappedTrainingResults)
```

Algorithm 1: Main Algorithm

As benchmarks to the jointly-optimized-2-period model, we first run the 1-period algorithm for each period, i.e. minimize the agents' costs in either period separately. Intuitively, the former algorithm can be interpreted as a long-term perspective, considering the future compliance in the current period and thus planning ahead by investing more in increasing their capacities, even when the first period ends. And the latter one can be seen as a short-sighted approach, caring only for the current quota. These 2 distinctive perspectives can make a huge difference in not only the agents' own positions but also the market prices.

The only difference between 2 algorithms lies in the ***stepwise approximation*** when computing forward losses and recording approximated paths. Specifically, as is shown in 1.2, there are more additional processes (e.g. V_t^i) in the jointly optimized 2-period case. Yet in general, the ***stepwise approximation*** algorithm can be roughly displayed as the following pseudocode:

```
def getforwardloss():
    '''
    Perform the stepwise approximation for a single iteration.
    The annotations for pop1 and pop2 might be omitted.
    '''
    x=x0; c=c0; y=y0model(x0); ## initial values for both pops (@ t==0)
    for j in [1,NT2]:  ## time steps
        ## update the processes at each time step by the discretized FBSDEs
        y = y + zymodels[j-1]*db[:,j] ### update probability of defaulting conditioned on present (@
        s = weighted mean y  ### market price
        g, gamma, a = ... ### overtime-generation, trading, and expansion rates
        c = c + ...  ### update accumulative increments of baseline rate
        x = x + ...  ### update current inventory level

        if j == NT1:
            xt1, yt1, ... = x, y, ... ### record/freeze some processes
            x = nn.ReLU()(x-K)  # @ t==NT1: submit min(K,xt1) inventoreis

    loss = Loss(y,target(x)) + ...  ## summed-up loss for all terminal conditions for both pops
    return loss
```

Algorithm 2: Shooting Method - ***Stepwise Approximation***

2.2 Numeric Tricks

The trickiest problem we are facing is the indicator functions in *terminal conditions*, and one would naturally recall sigmoid approximation for increasing continuity and differentiability:

$$\mathbf{1}_{0.9>x} \approx \sigma(0.9-x), \text{ where the sigmoid function } \sigma(u) = \frac{1}{1+e^{-u/\delta}}. \tag{11}$$

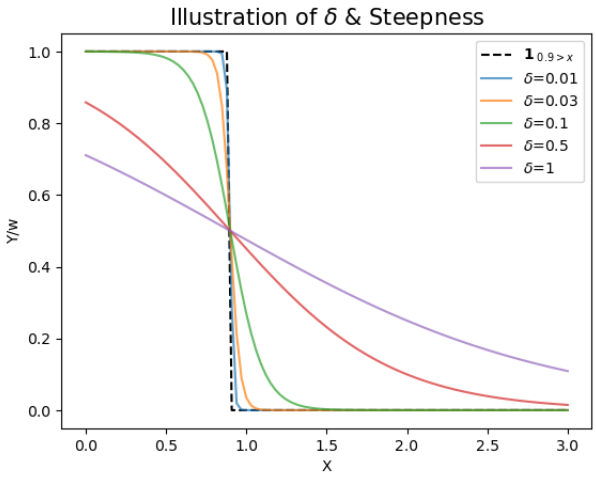

Figure 1: Sigmoid Approximation

In particular, the parameter δ controls the steepness of $\sigma(\cdot)$ and usually is a small positive number - the smaller δ is, the more closely it approximates the step of the indicator function. On the other hand, the ordinary NN models may learn $V_t^i, U_t^i, Y_t^i \notin [0,1]$ (let's fix $w=1$ for now), which is meaningless as they represent the *probabilities* of defaulting (i.e. missing the quota). And instead of using `tensor.clamp` to forcefully clamp values within $[0,1]$ only, we combine it with the **clamp trick** to restrict values while maintaining differentiability. (Same applies to V_t^i, U_t^i.)

$$dY_t^i = Y_t^i(1 - Y_t^i)Z_t dB_t. \tag{12}$$

Nonetheless, both the sigmoid approximation and the clamp trick pose huge challenges to the numeric stability. For the sigmoid function, when δ is too small, there is a great potential for numerical overflow - the exponents could be tremendous especially when X_t is far greater than 0.9, such that `torch.exp(u)` is `inf` when $u \geq 7.1$. This will raise errors/warnings[8] in PyTorch. For the clamp trick to work, we must ensure the initial values strictly fall in $(0, 1)$. Thus we propose logit trick to map the range $[0, 1] \to \mathbb{R}$, which also avoids working with large exponents:

$$\tilde{Y} := w * \text{logit}(Y/w) = w * \ln\left(\frac{Y/w}{1 - Y/w}\right) = f(Y)\ . \tag{13}$$

Then apply *Itô's formula* (with superscript $[\cdot]^i$ omitted):

$$d\tilde{Y}_t = (w/2 - Y_t)Z_t^2 dt + wZ_t dB_t\ . \tag{14}$$

Correspondingly, we use *BCEWithLogitsLoss* as the loss function, which combines a Sigmoid layer and the *BCELoss* in one single class. This version is more numerically stable than using a plain *Sigmoid* followed by a *BCELoss* as, by combining the operations into one layer, it takes advantage of the log-sum-exp trick for numerical stability.

Worth mentioning, we experimented with multiple combinations of tricks and loss functions, paired with different optimizers and schedulers. Eventually, we chose `Adamax` and `StepLR` due to their relatively better and more stable performance for all cases in general. Specifically, the 4 valid combinations of tricks and loss functions are shown as follows.

	target type	trick	loss type
1	indicator	logit	BCEWithLogitsLoss
2	indicator	clamp	BCELoss
3	indicator	clamp	MSELoss
4	sigmoid	clamp	MSELoss

Table 2: Valid Combos

3 Results

To evaluate the algorithm performances, we define a well-wrapped class `plot_results`[9], which also visualizes agents' behaviors (or interactions) and their market impacts. Specifically, it produces results from the following aspects:

- **Agents' behaviors and market impacts**
  - Learnt optimal control processes
  - Decomposed inventory accumulation processes

[8] Examples of RuntimeError and RuntimeWarning on PyTorch Forums.

[9] See more instruction details in GitHub README for the Joint-Optimization or Separate-Optimization.

    - Inventory levels during 2 compliance periods
    - Terminal inventories ready-to-submit
    - Market-clearing prices
- **Algorithm convergency and learning loss**
    - Average forward losses against a number of epochs trained
    - Learnt terminal conditions vs. targets

And here are some example diagrams from either perspective.

3.1 Jointly Optimized 2-Agent-2-Period Sample Results

Here we use the sample results by model[10] with terminal target function: $0.25\,\sigma\big(0.9 - X_T^i\big)$, where $\delta = 0.03$ and $T = T_1, T_2$. The loss function is *MSELoss* and the trick used in the ***stepwise approximation*** is *clamp*, correspondingly. (See Table 2.)

The learned controls are shown as the first row of Figure 2 and the accumulative generations by different means as the second row, correspondingly. The green color denotes the sub-population 1 while red the sub-population 2. (The same applies to all the later plots.) The plots in Figure 3 display a rather good convergence of learned terminal values to their *sigmoid targets*.

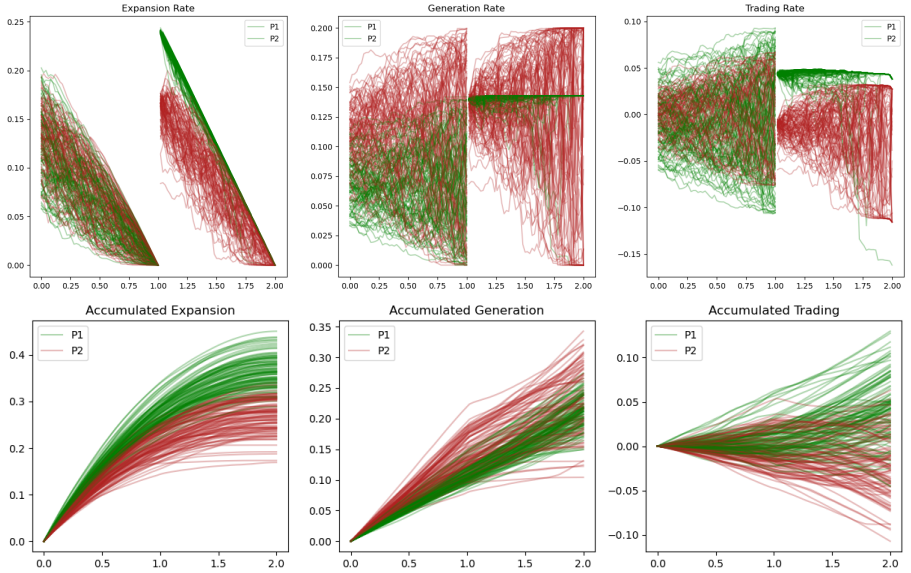

Figure 2: Decomposed Generation of Joint-Optimization

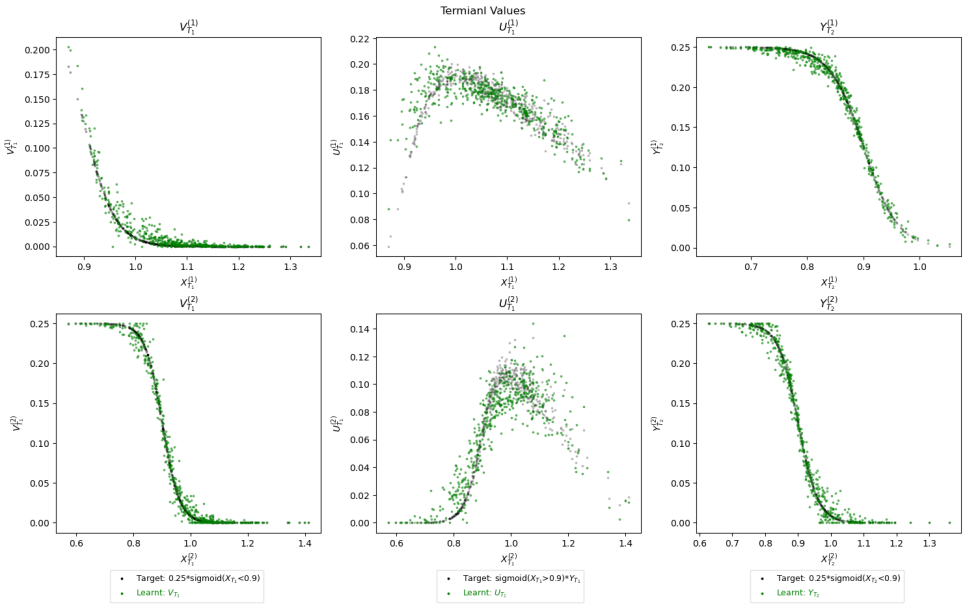

Figure 3: Terminal Values of V, U, Y

[10]The code and full results can be found in the GitHub Repository.

3.2 Separately Optimized 2-Agent-2-Period Sample Results

For the sake of comparability, here we use the same numeric trick, target, and loss function, except for minor changes when fine-tuning the specific model parameters[11].

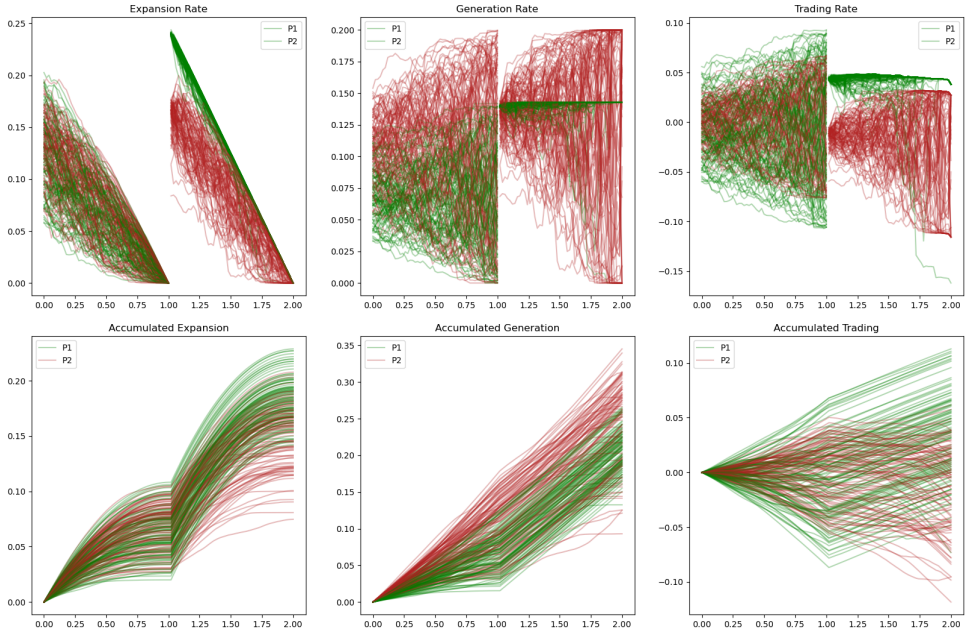

Figure 4: Decomposed Generation of Separate-Optimization

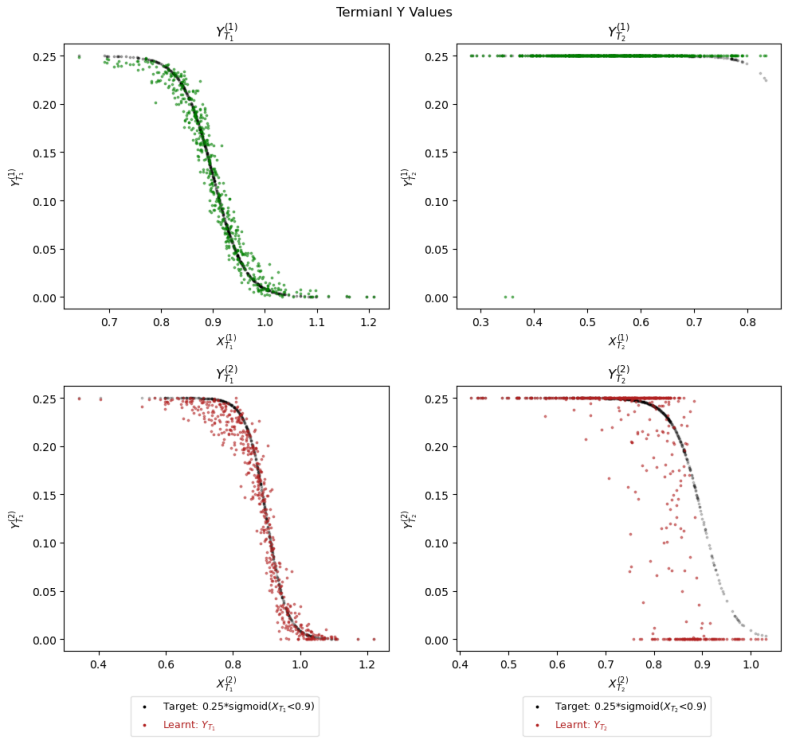

Figure 5: Terminal Values of Y

[11]The code and full results can be found in the GitHub Repository.

3.3 Comparisons And Analyses

Joint Versus Separate Optimization Upon comparing the shown results from 2 different perspectives, one can get enlightening implications.

By planning ahead in $\mathfrak{T}_1$ out of a long-term view, agents tend to invest more in expansion even at the end of the first period, whereas when only dedicated to meeting the current target, all agents reduce the expansion rate to zero since there's no point investing in delayed payoffs. Similarly, the short-sighted agents will trade more actively and might work more overtime at the first-period end in seek of immediate paybacks. Consequently, most of these agents unaware of the upcoming second compliance period will end up "just" meet the quota of 0.9 at T_1, since any extra inventory would be regarded "useless" - yet find themselves having to start over from almost scratch to meet the second quota. This can be seen from the first columns of inventory histograms and the inventory level plots.

Therefore in $\mathfrak{T}_2$, agents starting from 0 inventory will either try very hard to make up for expansion (exemplified by the green "pop1"), or rely heavily on over-hours and trading (exemplified by the red "pop2"), which pushes the market price even higher in the second period. However, both populations would ultimately realize the quota is almost impossible to meet and give up at all. This is shown by a great proportion of $Y^i_{T_2} = w$. Contrarily, only until the true "end of the world" (i.e. T_2) approaches, those who have maintained a reasonable level of stock start to gradually reduce expansion and sell any extra inventories. And since most of them have already accumulated a relatively high baseline rate, there's less need for them to work overtime or purchase inventories than in the first period (shown by the decreased slopes of the accumulated generation plots) - thus the market price goes down.

Sub-Population 1 Versus 2 Then let's take a closer look at either case, analyzing the differences made by distinctive preferences and initial conditions across sub-populations. Starting at generally lower initial level ($v^1 > v^2$) yet blessed with greater baseline rate ($h^1 < h^2$), "pop2" wouldn't worry as much as the green guys "pop1" in terms of expansion, and find working overtime or trading more rewarding. Even further, the red guys would be more interested in overtime than in trading since it's "cheaper" per unit inventory, which is the opposite for the green guys (i.e. $\zeta^1 > \gamma^1, \zeta^2 < \gamma^2$).

However, regardless of agents' perspectives (long/short-term), the the explicit initial advantage in inventory level and the implicit the disadvantage in baseline capacity makes the green guys "lazier", i.e. less motivated to work extra hours. Consequently, in the second compliance period, they are more likely to find themselves hard-pressed to meet the quota and have to purchase from the red guys, which is indicated by the trends and signs (positive for buying and vice versa) of accumulated trading amount.

Model Performances Both algorithms produce descending loss plots and learned terminal conditions that almost overlap with their targets (black dots) given $X^i_{T_1}, X^i_{T_2}$, which suggests converging and stable of model performance as desired. Worth mentioning, since *sigmoid targets* with *MSELoss* have the greatest differentiability, combined with small w (e.g. 0.25) narrowing down the step from 0 to w, models set up as such would produce rather good-looking results. Certainly, there might be other parameter and model settings leading to greater convergence and stability, which are open for experimenting.

4 Conclusions And Takeaways

From the results and analysis above, one can take away some instructive implications and apply not only to the REC markets but also to one's daily life.

- *Always plan ahead and consider for the future.*
- *Always do slightly more than required and maintain a reasonable level of backups.*
- *Don't be blinded by the apparent advantages/achievements, instead care for the growth rate and capacity - that's what you can carry to the future for sure.*
- *When the majority gets lazy or short-sighted, the market gets worse - where any individual will be affected more or less.*

References

[1] S. Campbell, Y. Chen, A. Shrivats, and S. Jaimungal, *Deep learning for principal-agent mean field games*, 2021. arXiv: 2110.01127 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2110.01127.

[2] R. Carmona and F. Delarue, *Probabilistic analysis of mean-field games*, 2012. arXiv: 1210.5780 [math.PR]. [Online]. Available: https://arxiv.org/abs/1210.5780.

[3] J. Han and J. Long, "Convergence of the deep bsde method for coupled fbsdes.," *Probability, Uncertainty and Quantitative Risk*, vol. 5, 2020. [Online]. Available: https://doi.org/10.1186/s41546-020-00047-w.