Pontryagin-Guided Deep Learning for Large-Scale Constrained Dynamic Portfolio Choice

Hyelin Choi

Department of Mathematics
Sungkyunkwan University

April 10, 2025

Pontryagin-Guided Deep Learning for Large-Scale Constrained Dynamic Portfolio Choice

Jeonggyu Huh[1], Jaegi Jeon[*2], Hyeng Keun Koo[3], and Byung Hwa Lim[4]

[1]Department of Mathematics, Sungkyunkwan University, South Korea
[2]Graduate School of Data Science, Chonnam National University, South Korea
[3]Department of Financial Engineering, Ajou University, South Korea
[4]Business School, Sungkyunkwan University, Seoul, South Korea

February 18, 2025

PG-DPO and PG-DPO-OneShot

Algorithm 1 PG-DPO

Inputs:

- Policy nets $(\boldsymbol{\pi}_\theta, C_\phi)$ (optionally with constraint-enforcing final activations);
- Step sizes $\{\alpha_k\}$, total iterations K;
- Domain sampler η for $(t_0^{(i)}, x_0^{(i)})$ in $\mathcal{D} \subset [0,T] \times (0,\infty)$;
- Integer N (steps per path).

for $j = 1$ to K **do**

(a) Sample mini-batch of size M. For each $i \in \{1,\dots,M\}$, draw $(t_0^{(i)}, x_0^{(i)}) \sim \eta$.

(b) Local Single-Path Simulation. For each i:

(a) $\Delta t^{(i)} \leftarrow \frac{T-t_0^{(i)}}{N}$; $X_0^{(i)} \leftarrow x_0^{(i)}$.

(b) For $k = 0,\dots,N-1$:

$$\boldsymbol{\pi}_k^{(i)} = \boldsymbol{\pi}_\theta(t_k^{(i)}, X_k^{(i)}),\ C_k^{(i)} = C_\phi(t_k^{(i)}, X_k^{(i)}),$$

$$X_{k+1}^{(i)} = X_k^{(i)} \exp\Big(\Big[(\boldsymbol{\pi}_k^{(i)})^\top \widetilde{\boldsymbol{\mu}}_k^{(i)} - \tfrac{1}{2}(\boldsymbol{\pi}_{1:n,k}^{(i)})^\top \Sigma_k^{(i)} \boldsymbol{\pi}_{1:n,k}^{(i)} - \frac{C_k^{(i)}}{X_k^{(i)}}\Big]\Delta t^{(i)} + (\boldsymbol{\pi}_k^{(i)})^\top \widetilde{V}_k^{(i)} \Delta \mathbf{W}_k^{(i)}\Big).$$

(c) Compute

$$J^{(i)}(\theta,\phi) = \sum_{k=0}^{N-1} e^{-\rho t_k^{(i)}} U(C_k^{(i)})\,\Delta t^{(i)} + \kappa\, e^{-\rho T} U(X_N^{(i)}).$$

(c) Backprop & Averaging:

$$\widehat{J}(\theta,\phi) = \frac{1}{M}\sum_{i=1}^{M} J^{(i)}(\theta,\phi), \quad \nabla_{(\theta,\phi)}\widehat{J} \leftarrow \text{BPTT}.$$

(d) Parameter Update:

$$(\theta,\phi) \leftarrow (\theta,\phi) + \alpha_k \nabla_{(\theta,\phi)}\widehat{J}.$$

end for

return $(\boldsymbol{\pi}_\theta, C_\phi)$.

Algorithm 2 PG-DPO-OneShot

Additional Inputs:

- A brief "warm-up" phase (e.g., K_0 iterations of PG-DPO).
- Suboptimal adjoint λ_k and its spatial derivative $\frac{\partial}{\partial x}\lambda_k$, *both* obtained via automatic differentiation (BPTT) for each node (t_k, X_k).

(a) Warm-Up Training:

(a) Run PG-DPO for K_0 iterations. Although $\boldsymbol{\pi}_\theta$, C_ϕ may remain suboptimal, the costate $\lambda_t \approx \frac{\partial J}{\partial X_t}$ typically stabilizes quickly under BPTT.

(b) After warm-up, at each node (t_k, X_k), retrieve λ_k and $\frac{\partial}{\partial x}\lambda_k$ from autodiff.

(b) OneShot Pontryagin Controls (Unconstrained vs. Barrier).

(a) *If unconstrained*, apply a closed-form Pontryagin FOC $(\boldsymbol{\pi}_k^{\text{PMP}}, C_k^{\text{PMP}})$ (e.g. (19) in a multi-asset Merton model).

(b) *If constraints exist*, solve $\max_{\boldsymbol{\pi}_k, C_k} \widetilde{\mathcal{H}}_{\text{barrier}}$ (see (20)) via a small-scale *barrier-based* Newton–line-search at $(t_k, X_k, \lambda_k, \frac{\partial}{\partial x}\lambda_k)$. Return $(\boldsymbol{\pi}_k^{\text{PMP}}, C_k^{\text{PMP}})$.

(c) Deploy OneShot Controls:

(a) At test time, *ignore* the network outputs $(\boldsymbol{\pi}_\theta, C_\phi)$. Use $(\boldsymbol{\pi}_k^{\text{PMP}}, C_k^{\text{PMP}})$ from step (c) instead.

(b) This requires only a short warm-up plus local solves (closed-form or barrier). Experiments (Section 5) confirm near-optimal solutions with significantly reduced training cost.

Problem Formulation

We formulate the following utility maximization problem faced by a risk-averse investor in a continuous-time financial market:

$$\max_{(\pi_t, C_t)} J(\pi_t, C_t) = \mathbb{E}\left[\int_0^T e^{-\rho t} U(C_t) dt + \kappa e^{-\rho T} U(X_T)\right] \tag{1.1}$$

$$\text{subject to} \;\; dX_t = \left(X_t \pi_t^T \tilde{\mu}_t - C_t\right) dt + X_t \pi_t^T \tilde{V}_t dW_t, \quad X_0 = x_0 > 0 \tag{1.2}$$

where $W_t \in \mathbb{R}^n$ is an n-dimensional standard Brownian motion, $\rho > 0$ is a continuous discount rate, $\kappa > 0$ is a bequest parameter, and U is a CRRA utility function defined as:

$$U(x) = \begin{cases} \frac{x^{1-\gamma}}{1-\gamma}, & \gamma > 0, \; \gamma \neq 1 \\ \ln(x), & \gamma = 1 \end{cases} \tag{1.3}$$

where γ represents the investor's relative risk aversion.

We consider two types of constraints:

- **Portfolio weight constraints**: $\pi_{i,t} \geq 0 \quad \text{for} \quad i = 0, 1, \ldots, n.$
- **Consumption bounds**: $C_{\min} \leq C_t \leq C_{\max}$

where $C_{\min} > 0$ might capture mandatory living expenses, and $C_{\max} < \infty$ might limit overconsumption.

Closed-Form Solutions without Constraints

In the finite-horizon Merton problem-specifically, unconstrained portfolio choice, CRRA utility, and deterministic parameters μ_t, Σ_t, r_t - we can derive a closed-form solution. The optimal investment proportion is given by:

$$\pi^*_{1:n,t} = \frac{1}{\gamma}\Sigma_t^{-1}(\mu_t - r_t\mathbf{1}), \quad \pi^*_{0,t} = 1 - \sum_{i=1}^{n} \pi^*_{i,t}. \tag{1.4}$$

The optimal consumption rate C_t^* takes the form

$$C_t^* = \alpha(t)X_t \tag{1.5}$$

where $\alpha(t)$ is a time-varying function derived from the HJB equation.

Closed-Form Solutions without Constraints

For instance, if μ_t, Σ_t, r_t are all constant in t, the consumption rate simplifies to a known closed-form expression:

$$\alpha(t) = \frac{\kappa}{\gamma}(1 - e^{-\kappa(T-t)})^{-1} \tag{1.6}$$

with decay rate

$$\kappa = \rho - (1-\gamma)\left(r + \frac{(\mu - r\mathbf{1})^T \Sigma^{-1}(\mu - r\mathbf{1})}{2\gamma}\right). \tag{1.7}$$

Closed-Form Solutions without Constraints

However, once constraints are introduced, such as portfolio bounds or consumption limits, the analytical solution no longer holds.
In these cases, numerical methods are required. One such approach is **Pontryagin-guided direct policy optimization**, which is scalable even in high-dimensional or constrained settings.

Recall that

$$\max_{(\pi_t, C_t)} J(\pi_t, C_t) = \mathbb{E}\left[\int_0^T e^{-\rho t} U(C_t) dt + \kappa e^{-\rho T} U(X_T)\right] \tag{2.1}$$

$$\text{subject to} \quad dX_t = \left(X_t \pi_t^T \tilde{\mu}_t - C_t\right) dt + X_t \pi_t^T \tilde{V}_t dW_t, \quad X_0 = x_0 > 0. \tag{2.2}$$

We define the Hamiltonian as:

$$\mathcal{H}(t, X_t, \pi_t, C_t, \lambda_t, Z_t) = e^{-\rho t} U(C_t) + \lambda_t \left[X_t \pi_t^T \tilde{\mu}_t - C_t\right] + Z_t^T \left(X_t \tilde{V}_t{}^T \pi_t\right), \tag{2.3}$$

- $\lambda_t \approx \frac{\partial J}{\partial X_t}$: sensitivity to wealth
- Z_t: sensitivity to randomness from W_t.

When the controls (π_t^*, C_t^*) are optimal, the wealth process X_t^* and adjoint processes (λ_t^*, Z_t^*) jointly satisfy the *coupled forward-backward Pontryagin system*:

$$dX_t^* = \left[X_t^* \left(\pi_t^*\right)^T \tilde{\mu}_t - C_t^*\right] dt + X_t^* \left(\pi_t^*\right)^T \tilde{V}_t dW_t, \quad X_0^* = x_0 > 0, \tag{2.4}$$

$$d\lambda_t^* = -\frac{\partial \mathcal{H}}{\partial X}\left(t, X_t^*, \pi_t^*, C_t^*, \lambda_t^*, Z_t^*\right) dt + Z_t^{*T} dW_t, \quad \lambda_T^* = \frac{\partial}{\partial X}\left[\kappa e^{-\rho T} U(X_T^*)\right]. \tag{2.5}$$

Additionally, (π_t^*, C_t^*) satisfies

$$(\pi_t^*, C_t^*) = \arg\max_{\pi_t, C_t} \mathcal{H}(t, X_t^*, \pi_t, C_t, \lambda_t^*, Z_t^*). \tag{2.6}$$

This yields a **closed-loop structure**.

At each t, the optimal control (π_t^*, C_t^*) locally maximizes the Hamiltonian $\mathcal{H}$. This gives the following first-order conditions:

$$\frac{\partial \mathcal{H}}{\partial C_t} = e^{-\rho t} U'(C_t^*) - \lambda_t^* = 0, \quad \Rightarrow \quad C_t^* = \left(e^{\rho t} \lambda_t^*\right)^{-\frac{1}{\gamma}}, \tag{2.7}$$

$$\frac{\partial \mathcal{H}}{\partial \pi_t} = \lambda_t^* X_t^* \tilde{\mu}_t + X_t^* \tilde{V}_t^T Z_t^* = \mathbf{0}. \tag{2.8}$$

From the BSDE (2.5), we typically have:

$$Z_t^* = (\partial_x \lambda_t^*) \left(X_t^* \tilde{V}_t{}^T \pi_t^*.\right) \tag{2.9}$$

Plugging into the first-order condition, we get the optimal portfolio weights in feedback form:

$$\pi_{1:n,t}^* = -\frac{\lambda_t^*}{X_t^*(\partial_x \lambda_t^*)} \Sigma_t^{-1} (\mu_{1,t} - r_t, \ldots, \mu_{n,t} - r_t). \tag{2.10}$$

We impose nonnegativity and full investment constraints:

$$\pi_{i,t} \geq 0, \quad \sum_{i=0}^{n} \pi_{i,t} = 1. \tag{3.1}$$

These can be written as:

$$h(\pi_t) = 1 - \sum_{i=0}^{n} \pi_{i,t} = 0, \quad g_i(\pi_t) = -\pi_{i,t} \leq 0, \quad i = 0, \dots, n. \tag{3.2}$$

Next, we build an augmented Hamiltonian:

$$\tilde{\mathcal{H}}_{\mathrm{KKT}} = \mathcal{H} + \eta h(\pi_t) + \sum_{i=0}^{n} \zeta_{i,t} g_i(\pi_t), \tag{3.3}$$

where η_t and $\zeta_{i,t} \geq 0$ are Lagrange multipliers.

By KKT theory, we have

$$\frac{\partial \tilde{\mathcal{H}}_{\mathrm{KKT}}}{\partial \pi_{i,t}} = 0, \quad h(\pi_t) = 0, \quad g_i(\pi_t) \le 0, \quad \zeta_{i,t} g_i(\pi_t) = 0. \tag{3.4}$$

These conditions ensure that the portfolio respects **no short-selling**, **no borrowing**, and **full investment**.
However, solving KKT systems in high dimensions can be **computationally expensive**, especially due to the nonlinear complementarity conditions.

As an alternative to KKT, we use the *log-barrier method*, which handles inequality constraints smoothly by adding a logarithmic penalty:

$$\tilde{\mathcal{H}}_{\text{barrier}} = \mathcal{H} + \eta \left(1 - \sum_{i=0}^{n} \pi_{i,t} \right) + \epsilon \sum_{i=0}^{n} \ln(\pi_{i,t}) \tag{3.5}$$

where $\epsilon > 0$. We derive the first-order conditions:

$$\frac{\partial \mathcal{H}}{\partial \pi_{i,t}} = \eta_t + \frac{\epsilon}{\pi_{i,t}}, \quad i = 0, \ldots, n, \quad \text{with } \sum_{i=0}^{n} \pi_{i,t} = 1. \tag{3.6}$$

To solve this system, we define a function $\mathbf{F} : \mathbb{R}^{n+2} \to \mathbb{R}^{n+2}$ as:

$$\mathbf{F}(\pi_t, \eta_t) = (F_0, \ldots, F_n, F_{\text{sum}})^T, \tag{3.7}$$

where each component encodes either a barrier-FOC or the sum-to-one constriant:

$$F_i(\pi_t, \eta_t) = \frac{\partial \mathcal{H}}{\partial \pi_{i,t}} - \left(\eta_t + \frac{\epsilon}{\pi_{i,t}} \right), \quad i = 1, \ldots, n, \tag{3.8}$$

$$F_{\text{sum}}(\pi_t, \eta_t) = \sum_{i=0}^{n} \pi_{i,t} - 1. \tag{3.9}$$

We solve $\mathbf{F}(\pi_t, \eta_t) = 0$ using **Newton's method**:

$$D\mathbf{F}(\pi_t^{(k)}, \eta_t^{(k)})\Delta = -\mathbf{F}(\pi_t^{(k)}, \eta_t^{(k)}), \quad \begin{pmatrix} \pi_t^{(k+1)} \\ \eta_t^{(k+1)} \end{pmatrix} = \begin{pmatrix} \pi_t^{(k)} \\ \eta_t^{(k)} \end{pmatrix} + \alpha_k \Delta, \tag{3.10}$$

where $0 < \alpha_k \leq 1$ is chosen so that $\pi_{i,t}^{(k+1)} > 0$.

Repeating this process until $\left\| \mathbf{F}(\pi_t^{(k)}, \eta_t^{(k)}) \right\| \to 0$ yields the barrier-based solution. This **avoids the cost of KKT methods**, ensures **positivity**, and **scales well** in high dimensions.

We parameterize the controls using neural networks:

$$\pi_t = \pi_\theta(t, X_t), \quad C_t = C_\phi(t, X_t), \tag{3.11}$$

where θ and ϕ denote the neural network parameters. Given a fixed policy (π_θ, C_ϕ), the induced adjoint processes (λ_t, Z_t) remain well-defined and satisfy a backward stochastic differential equation (BSDE):

$$\begin{aligned} d\lambda_t &= -\frac{\partial}{\partial X}\tilde{\mathcal{H}}\left(t, X_t, \pi_\theta(t, X_t), C_\phi(t, X_t), \lambda_t, Z_t\right) dt + Z_t^T dW_t, \\ \lambda_T &= \frac{\partial}{\partial X}\left[\kappa e^{-\rho T} U(X_T)\right], \end{aligned} \tag{3.12}$$

where X_t evolves under the suboptimal policy (π_θ, C_ϕ). However, Modern deep learning frameworks like PyTorch do not numerically solve (3.12) directly.

Instead of solving the BSDE directly, we leverage the key relationship:

$$\lambda_t = \frac{\partial J}{\partial X_t}. \tag{3.13}$$

This allows us to compute the policy-fixed adjoint λ_t through backpropagation. Once λ_t is obtained, the process Z_t can be recovered via:

$$Z_t = (\partial_x \lambda_t) \left(X_t \tilde{V_t}^{T} \pi_t \right). \tag{3.14}$$

As a result, the adjoint processes (λ_t, Z_t) emerge as byproducts of the gradient calculation $\nabla_{\theta,\phi} J$, eliminating the need for a standalone BSDE solver and simplifying the time discretization process.

To calculate the gradients $\nabla_\theta J$ and $\nabla_\phi J$, we first rewrite the state process X_t under the suboptimal policy (π_θ, C_ϕ):

$$dX_t = b(t; \theta, \phi)dt + \sigma(X_t, ; \theta, \phi)dW_t \tag{3.15}$$

where

$$\begin{aligned} b(X_t; \theta, \phi) &= rX_t + \pi_\theta(t, X_t)(\mu - r)X_t - C_\phi(t, X_t), \\ \sigma(X_t; \theta, \phi) &= \sigma\pi_\theta(t, X_t)X_t. \end{aligned} \tag{3.16}$$

Then, the parameter gradients adopt a Pontryagin-like form

$$\nabla_\theta J = \mathbb{E}\left[\int_0^T \left(\lambda_t \frac{\partial b}{\partial \theta} + Z_t^T \frac{\partial \sigma}{\partial \theta}\right) dt\right] + (\text{direct payoff term in } \theta), \tag{3.17}$$

$$\nabla_\phi J = \mathbb{E}\left[\int_0^T \left(\lambda \frac{\partial b}{\partial \phi} + Z_t^T \frac{\partial \sigma}{\partial \phi}\right) dt\right] + (\text{direct payoff term in } \phi). \tag{3.18}$$

Here λ_t and Z_t are precisely the *policy-fixed* adjoint processes from (3.12), while b and σ denote the drift and diffusion of X_t under (π_θ, C_ϕ).

Finally, we have

$$\nabla_\theta J = \mathbb{E}\left[\int_0^T \left\{\lambda_t X_t \tilde{\mu}_t + X_t(\tilde{V}_t Z_t)\right\}^T \frac{\partial \pi_\theta}{\partial \theta} dt\right], \tag{3.19}$$

$$\nabla_\phi J = \mathbb{E}\left[\int_0^T \lambda_t \left(-\frac{\partial C_\phi}{\partial \phi}\right) dt\right] + \mathbb{E}\left[\int_0^T e^{-\rho t} U'(C_\phi(\cdot)) \frac{\partial C_\phi(\cdot)}{\partial \phi} dt\right]. \tag{3.20}$$

The consumption gradient has two clear terms: a *wealth sensitivity* part $(-\lambda_t \partial_\phi C_\phi)$ and *direct utility part* $(e^{-\rho t} U'(C_\phi)\partial_\phi C_\phi)$.
Equation (3.19)-(3.20) illustrate how the adjoint variables (λ_t, Z_t), obtained via automatic differentiation, dictate the directions to update (θ, ϕ). Thus, neural network training aligns with Pontryagin's principle, without requiring explicit BSDE solvers.

The following steps outline the core of the PG-DPO approach in discrete time:

Step 1: Choose Final Activations for Network Constraints

- *Unconstrained*: Simply output real-valued coordinates; no explicit activation is needed.
- *Constrained* (No Borrowing / Short Selling): Use a softmax of length $n+1$ to ensure $\pi_k \geq \mathbf{0}$ and $\sum_{i=0}^{n} \pi_{i,k} = 1$.
- *Consumption Bounds*: If $0 \leq C_k \leq \alpha X_k$ must hold, a scaled sigmoid final activation can keep C_k within that range. ex) $C_k = \alpha X_k \sigma(h_k)$

Step 2: Discretize Dynamics and Objective
We discretize the time interval $[0, T]$ into N uniform steps of size $\Delta t = T/N$ and define $t_k = k\Delta t$ for $k = 0, \dots, N$, so that $t_0 = 0$ and $t_N = T$. We approximate the continuous-time SDE via an *exponential Euler* scheme that preserves the geometric nature of wealth updates.
Concretely, we freeze the controls (π_t, C_t) on each interval $[t_k, t_k + \Delta t]$. Here we let

$$\pi_k = \pi_\theta(t_k, X_k), \quad C_k = C_\phi(t_k, X_k). \tag{4.1}$$

Applying Itô's lemma to $\ln(X_s)$ over $[t_k, t_k + \Delta t]$ yields the local exponential update for the wealth process:

$$X_{k+1} = X_k \exp\left[\left(\pi_k^\top \tilde{\mu}_k - \frac{1}{2}\pi_{1:n,k}^\top \Sigma_k \pi_{1:n,k} - \frac{C_k}{X_k}\right)\Delta t + \pi_k^\top \tilde{V}_k \Delta W_k\right] \tag{4.2}$$

where

- $\pi_{1:n,k}$ denotes the subvector $(\pi_{1,k}, \dots, \pi_{n,k})$,
- $\Sigma_k = \tilde{V}_k \tilde{V}_k^\top$ is the covariance matrix,
- $\Delta W_k = W_{t_{k+1}} - W_{t_k} \sim \mathcal{N}(0, \Delta t \cdot I_n)$,
- $\tilde{\mu}_k \in \mathbb{R}^{n+1}$ stacks $(r_{t_k}, \mu_{1,t_k}, \dots, \mu_{n,t_k})^\top$,
- $\tilde{V}_k \in \mathbb{R}^{(n+1)\times n}$ includes a zero row for the risk-free asset and a Cholesky-type factorization for risky assets.

Step 3: Single Forward Path per (t_k, X_k)
At each node (t_k, X_k), we run exactly one forward simulation from k to the terminal index N. This yields a single-sample payoff, unbiased but subject to Monte Carlo variance. After the simulation, backpropagation (autodiff) yields local adjoint estimates λ_k and Z_k under the current policy (θ, ϕ).

Step 4: Compute λ_k via BPTT

In typical deep learning frameworks such as PyTorch or JAX, we build a computational graph from (θ, ϕ) through $\{\pi_k, C_k\}$ and $\{X_k\}$ to the approximate objective $J(\theta, \phi)$.
A single call to `.backward()` (or an equivalent autodiff routine) computes the gradients $\nabla_\theta J$ and $\nabla_\phi J$, while simultaneously yielding the partial derivatives $\frac{\partial J}{\partial X_k}$ at each time step.
Identifying

$$\lambda_k = \frac{\partial J}{\partial X_k} \tag{4.3}$$

aligns with the Pontryagin perspective that λ_k is the (suboptimal) adjoint measuring the sensitivity of the overall cost to changes in X_k.

Step 5: Obtain $\partial_x \lambda_k$ and hence Z_k

To compute Z_k, we typically need $\partial_x \lambda_k$. In the multi-asset zero-indexed Merton model, a common approach idfferentiate the Hamiltonian or uses the relation

$$Z_k \approx [\partial_x \lambda_k] \left(X_k \tilde{V}_k^T \pi_k \right). \tag{4.4}$$

Although this step does not impose optimality, it provides additional derivatives needed for constructing the exact parameter gradients.

Step 6: Update Network Parameters
Finally, we collect $\nabla_\theta J$ and $\nabla_\phi J$ and update (θ, ϕ) using a stochastic optimizer such as Adam or SGD. The expanded $\nabla_\theta J$ might be approximated by

$$\nabla_\theta J \approx \mathbb{E}\left[\sum_{k=0}^{N-1} \left\{\lambda_k X_k \tilde{\mu}_k + X_k(\tilde{V}_k Z_k)\right\}^T \frac{\partial \pi_\theta(t_k, X_k)}{\partial \theta} \Delta t\right] \tag{4.5}$$

while the expanded $\nabla_\phi J$ might be approximated by

$$\nabla_\phi J = \mathbb{E}\left[\sum_{k=0}^{N-1} \lambda_k \left(-\frac{\partial C_\phi(t_k, X_k)}{\partial \phi}\right) \Delta t\right] + \mathbb{E}\left[\sum_{k=0}^{N-1} e^{-\rho t_k} U'(C_\phi(\cdot)) \frac{\partial C_\phi(t_k, X_k)}{\partial \phi} \Delta t\right]. \tag{4.6}$$

Averaging these over M trajectories yields a stochastic approximation of ∇_θ and $\nabla_\phi J$. Repeating this process eventually produces a stationary policy.

Algorithm 1 PG-DPO

Inputs:

- Policy nets $(\boldsymbol{\pi}_\theta, C_\phi)$ (optionally with constraint-enforcing final activations);
- Step sizes $\{\alpha_k\}$, total iterations K;
- Domain sampler η for $(t_0^{(i)}, x_0^{(i)})$ in $\mathcal{D} \subset [0,T] \times (0,\infty)$;
- Integer N (steps per path).

for $j = 1$ to K **do**

(a) Sample mini-batch of size M. For each $i \in \{1,\dots,M\}$, draw $(t_0^{(i)}, x_0^{(i)}) \sim \eta$.

(b) Local Single-Path Simulation. For each i:

(a) $\Delta t^{(i)} \leftarrow \frac{T-t_0^{(i)}}{N}$; $X_0^{(i)} \leftarrow x_0^{(i)}$.

(b) For $k = 0,\dots,N-1$:

$$\boldsymbol{\pi}_k^{(i)} = \boldsymbol{\pi}_\theta(t_k^{(i)}, X_k^{(i)}),\ C_k^{(i)} = C_\phi(t_k^{(i)}, X_k^{(i)}),$$

$$X_{k+1}^{(i)} = X_k^{(i)} \exp\Big(\Big[\big(\boldsymbol{\pi}_k^{(i)}\big)^\top \tilde{\boldsymbol{\mu}}_k^{(i)} - \tfrac{1}{2}\big(\boldsymbol{\pi}_{1:n,k}^{(i)}\big)^\top \Sigma_k^{(i)} \boldsymbol{\pi}_{1:n,k}^{(i)} - \frac{C_k^{(i)}}{X_k^{(i)}}\Big]\Delta t^{(i)} + \big(\boldsymbol{\pi}_k^{(i)}\big)^\top \tilde{V}_k^{(i)} \Delta \mathbf{W}_k^{(i)}\Big).$$

(c) Compute

$$J^{(i)}(\theta,\phi) = \sum_{k=0}^{N-1} e^{-\rho t_k^{(i)}} U\big(C_k^{(i)}\big) \Delta t^{(i)} + \kappa e^{-\rho T} U\big(X_N^{(i)}\big).$$

(c) Backprop & Averaging:

$$\widehat{J}(\theta,\phi) = \frac{1}{M}\sum_{i=1}^{M} J^{(i)}(\theta,\phi), \quad \nabla_{(\theta,\phi)}\widehat{J} \leftarrow \text{BPTT}.$$

(d) Parameter Update:

$$(\theta,\phi) \leftarrow (\theta,\phi) + \alpha_k \nabla_{(\theta,\phi)}\widehat{J}.$$

end for

return $(\boldsymbol{\pi}_\theta, C_\phi)$.

PG-DPO-OneShot

Algorithm 2 PG-DPO-OneShot

Additional Inputs:

- A brief "warm-up" phase (e.g., K_0 iterations of PG-DPO).
- Suboptimal adjoint λ_k and its spatial derivative $\frac{\partial}{\partial x}\lambda_k$, *both* obtained via automatic differentiation (BPTT) for each node (t_k, X_k).

1: **(a) Warm-Up Training:**

(a) Run PG-DPO for K_0 iterations. Although $\boldsymbol{\pi}_\theta$, C_ϕ may remain suboptimal, the costate $\lambda_t \approx \frac{\partial J}{\partial X_t}$ typically stabilizes quickly under BPTT.

(b) After warm-up, at each node (t_k, X_k), retrieve λ_k and $\frac{\partial}{\partial x}\lambda_k$ from autodiff.

2: **(b) OneShot Pontryagin Controls (Unconstrained vs. Barrier).**

(a) *If unconstrained*, apply a closed-form Pontryagin FOC $(\boldsymbol{\pi}_k^{\mathrm{PMP}}, C_k^{\mathrm{PMP}})$ (e.g. (19) in a multi-asset Merton model).

(b) *If constraints exist*, solve $\max_{\boldsymbol{\pi}_k, C_k} \widetilde{\mathcal{H}}_{\mathrm{barrier}}$ (see (20)) via a small-scale *barrier-based* Newton–line-search at $(t_k, X_k, \lambda_k, \frac{\partial}{\partial x}\lambda_k)$. Return $(\boldsymbol{\pi}_k^{\mathrm{PMP}}, C_k^{\mathrm{PMP}})$.

3: **(c) Deploy OneShot Controls:**

(a) At test time, *ignore* the network outputs $(\boldsymbol{\pi}_\theta, C_\phi)$. Use $(\boldsymbol{\pi}_k^{\mathrm{PMP}}, C_k^{\mathrm{PMP}})$ from step (c) instead.

(b) This requires only a short warm-up plus local solves (closed-form or barrier). Experiments (Section 5) confirm near-optimal solutions with significantly reduced training cost.