Notes

Alexander Peytz

Fall 2024

A Benchmark Model of Dynamic Portfolio Choice with Transaction Costs

In this section, we outline a canonical dynamic portfolio choice problem with transaction costs, similar to the one previously studied by Schober et al. (2022). We have chosen this benchmark because this dynamic portfolio choice problem scales naturally with an increasing number of risky assets and, therefore, can serve as an ideal benchmark to assess the performance of our novel solution algorithm (see Sec. 6).

The Asset Market

We consider a discrete-time dynamic portfolio choice model with a finite time horizon T in which an investor seeks to maximize their expected lifetime utility. The horizon $[0, T]$ is discretized into $T + 1$ equally spaced periods at which the investor can rebalance her portfolio. There is a single investor who can allocate her wealth in the asset market among $D + 1$ assets. D assets are risky (also referred to as "stocks") and are subject to proportional transaction costs. That is, when buying or selling a stock, the investor pays a transaction cost $\tau \in [0, 1]$ proportional to the amount spent. The stock's stochastic one-period gross returns are denoted by $R = (R_1, ..., R_D)^\top$.
In addition, the agent can invest in a risk-free bond that is not subject to transaction costs and yields a gross one-period return $R_f = \exp(r)$, where r is the one-period interest rate.

The Investor's Problem

There is a single consumption good (the numeraire) and an investor who maximizes her lifetime utility of consumption, that is,

$$E_0 \left[\sum_{t=0}^{T} \beta^t u(c_t W_t) \right],$$

where $\beta < 1$ is the agent's time discount factor, $u(\cdot)$ is the agent's utility function, c_t is the fraction of wealth spent on the consumption good in period t, and W_t is the investor's wealth. The investor tracks both wealth W_t and, due to the presence of transaction costs, the composition of wealth x_t. In each period, the agent reallocates her wealth W_t among the $D+1$ assets and consumes the consumption good c_t.
Given a t-th-period asset holding x_t, where $x_t = (x_{1,t}, ..., x_{D,t})^\top \in [0, 1]^D$ denotes the fraction of wealth allocated to stocks, the agent decides how much of each stock to buy and sell. Specifically, the investor updates her portfolio by $\delta_t = (\delta_{1,t}, ..., \delta_{D,t})$, where $\delta_{i,t} > 0$ indicates a purchase and $\delta_{i,t} < 0$ indicates a sale. The agent pays a transaction cost of $\tau|\delta_{i,t} W_t|$ for each asset $i \in [1, ..., D]$. After subtracting the transaction costs, the remaining wealth is saved in the bond:

$$b_t W_t = \left(1 - 1^\top \cdot x_t\right) W_t - 1^\top \cdot \delta_t W_t - 1^\top \cdot (\tau|\delta_t|W_t) - c_t W_t.$$

Return dynamics

In these this, the one-period stochastic return vector $\mathbf{R}_t$ is always assumed to be log-normal with

$$\log(\mathbf{R}_t) \sim \mathcal{N}\left(\left(\mu - \frac{\sigma^2}{2}\right)\Delta t,\, (\Lambda\Sigma\Lambda)\Delta t\right)$$

in $\mathbb{R}^k$, where $\mu = (\mu_1, \ldots, \mu_k)^\top$ is the drift, $\sigma = (\sigma_1, \ldots, \sigma_k)^\top$ is the volatility, Σ is the correlation matrix of the log-returns, $\Lambda = \text{diag}(\sigma_1, \ldots, \sigma_k)$, and σ^2 is the elementwise square of σ. We can express the returns in terms of the Cholesky factorization $\Sigma = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L} = (L_{i,j})_{k\times k}$ is a lower triangular matrix, implying

$$\log(R_i) = \left(\mu_i - \frac{\sigma_i^2}{2}\right)\Delta t + \sigma_i\sqrt{\Delta t}\sum_{j=1}^{i} L_{i,j} z_j,$$

where z_i are independent standard normal random variables, for $i = 1, \ldots, k$. Therefore, for...

The Dynamic Portfolio Choice Problem

The dynamic portfolio choice problem the investor faces is given by:

$$V_t(W_t, x_t) = \max_{c_t, \delta_t}\left\{u(c_t W_t) + \beta E_t\left[V_{t+1}(W_{t+1}, x_{t+1})\right]\right\}, \quad t < T,$$

for some initial wealth $W_0 > 0$. The respective bond holdings are defined by Equation (2), and $E_t[\cdot]$ is the expectation operator. Both W_{t+1} and x_{t+1} are random variables as they depend on the random variable R_t. The dynamics of the $D + 1$ state variables $\{W_{t+1}, x_{t+1}\}$ are given by:

$$W_{t+1} = b_t W_t R_f + \left((x_t + \delta_t)W_t\right)^\top \cdot R_t,$$

$$x_{t+1} = \frac{(x_t W_t + \delta_t W_t) \circ R_t}{W_{t+1}}.$$

The symbol $\circ$ represents the Hadamard product. The terminal value function is given by:

$$V_T(W_T, x_T) = u\left((1 - \tau 1^\top \cdot x_T)W_T\right).$$

Furthermore, the following constraints must hold:

$$\delta W_t \geq -x_t W_t, \quad b_t W_t \geq 0, \quad 1^\top \cdot x_t \leq 1, \quad t \in [0, T].$$

.

The Dynamic Problem for an Investor with CRRA Utility

We simplify the problem by normalizing the value function $v(x) = V(W,x)/W^{1-\gamma}$, allowing us to drop wealth as a state variable during optimization. This enables us to solve the dynamic program globally. We redefine the investor's normalized bond holdings b_t as:

$$b_t = 1 - 1^\top \cdot (x_t - \delta_t^+ + \delta_t^- - \tau(\delta_t^+ + \delta_t^-)) - c_t,$$

and the state dynamics can be expressed free of the investor's wealth:

$$\pi_{t+1} := b_t R_f + (x_t + \delta_t)^\top \cdot R_t,$$

$$x_{t+1} = \frac{(x_t + \delta_t) \circ R_t}{\pi_{t+1}}.$$

With this normalization, the investor solves the dynamic program:

$$v_t(x_t) = \max_{c_t, \delta_t} \left\{ u(c_t) + \beta E_t \left[\pi_{t+1}^{1-\gamma} v_{t+1}(x_{t+1}) \right] \right\}, \quad t < T,$$

with the terminal value function:

$$v_T(x_T) = u(1 - \tau 1^\top \cdot x_T).$$

The constraints hold for $t = 0,...,T$:

$$\delta_t >= -x_t \tag{0.1}$$

$$b_t >= 0 \tag{0.2}$$

$$1^\top \cdot x_t <= 1 \tag{0.3}$$

The non-normalized optimal choices can be obtained by multiplying with a given wealth W_t for any time t and state x_t.
Finally, we define the Non-Trading Region (NTR) as:

$$\Omega_t = \{x_t : \delta_t^* = 0\}.$$

Where δ_t^* is the optimal trading strategy.

A Scalable Solution Method for Dynamic Portfolio Choice Problems with Transaction Costs

In this section, we introduce a versatile, scalable, and flexible method based on supervised machine learning for solving discrete-time dynamic portfolio optimization problems that incorporate proportional transaction costs. We propose using discrete-time DP where we approximate the value and policy functions with GPRs. In addition, we incorporate a novel approach for generating an approximation of the NTR within each DP iteration. This combination enables us to efficiently address the aforementioned challenges, allowing for more effective scaling to higher dimensions compared to existing methods.

Abstract problem formulation & dynamic programming

The abstract class of problem we focus on is a discrete-time, finite-horizon stochastic optimal decision-making problem in the context of dynamic portfolio optimization. We provide a brief characterization, following the notation by Renner and Scheidegger (2018): let $x_t \in B \subset \mathbb{R}^D$ represent the state of the economy at time $t \in [0, 1, \ldots, T]$, where $D \in \mathbb{N}$ denotes the dimensionality of the state space. Controls (actions) are described by a period-specific policy function $p_t : B \to \zeta$, with $\zeta \subset \mathbb{R}^{D+1}$ being the space of possible controls. The transition function for the economy between periods is given by a distribution π, which depends on the current state and policies, i.e., $x_{t+1} \sim \pi(\cdot|x_t, p_t(x_t))$. The policy function p_t needs to be determined from optimality conditions, given the stochastic transition function π that maps a state-action pair to a successor state.
Dynamic programming (DP) is the standard approach for finding the sequence of controls $\{\chi_t\}_{t=0}^T$ to

maximize the value function

$$V(x_0) := \mathbb{E}\left[\sum_{t=0}^{T} \beta^t u(x_t, \chi_t)\right]$$

for an initial state $x_0 \in B$. Here, $u(\cdot,\cdot)$ is the return function, and $\chi_t \in \Gamma(x_t) \subset B$, with $\Gamma(x_t)$ denoting the set of feasible choices given x_t. The discount factor $\beta \in (0,1)$ weighs future returns. The DP approach aims to find period-specific policy functions p_t mapping the state x_t into the action χ_t, such that for all t, $\chi_t = p_t(x_t) \in \Gamma(x_t)$, and $\{\chi_t\}_{t=0}^{T}$ solves the original problem. The Bellman equation, which is consistent with the principle of optimality, can be expressed as

$$V_t(x) = \max_{\chi}\{u(x,\chi) + \beta\mathbb{E}[V_{t+1}(\tilde{x})]\},$$

where the successor state is distributed as $\tilde{x} \sim \pi(\cdot|x,\chi)$.
DP solves the problem recursively. Starting with a terminal value function for the terminal period $V_T(\cdot)$ for $t = T$, we solve the corresponding Bellman equation to obtain the previous period's value function values $\{v_{i,t-1}\}_{i=1}^{N}$, where $v_{i,t} = V_t(x_{i,t})$, and controls $\{\chi_{i,t-1}\}_{i=1}^{N}$, for each point in a set of sampled state points $\{x_{i,t-1}\}_{i=1}^{N}$. Iteratively, we compute the previous period's values v_{t-1} and controls χ_{t-1}, using a value function surrogate based on the state-value pairs from the previous iteration (details on how to compute the surrogate will be covered in Section 4.2). We continue this process until we have computed the value and policies for each period $\{\{v_{i,t}, \chi_{i,t-1}\}_{i=1}^{N}\}_{t=0}^{T-1}$.
Independent of the model specification, dynamic optimization problems are always accompanied by the same challenges when attempting to solve them numerically: efficiently approximating and interpolating high-dimensional value and policy functions on potentially irregularly shaped geometries. In the subsequent Sections 4.2 and 4.3, we describe how we tackle these challenges by combining GPRs with an approximation of the NTR.

Approximating functions with Gaussian process regressions

In the following, we briefly introduce GPR, a nonparametric regression method from supervised machine learning with universal approximation properties (see, e.g., Micchelli et al., 2006) that we will use to approximate and interpolate multivariate policy and value functions (for more details, see, e.g., Williams and Rasmussen, 2006; Murphy, 2012).
Consider a training dataset $D = \{X, y\}$ that consists of N input states $x_i \in B \subset \mathbb{R}^d$ with corresponding observations $y_i \in \mathbb{R}$. The observations are assumed to be generated by an unknown function f, such that $y_i = f(x_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. The matrix $X = [x_1, \ldots, x_N]$ contains the training inputs, while $y = [y_1, \ldots, y_N]^\top$ consists of the corresponding targets or training inputs. That training data can be generated using computer code or obtained from empirical data. The goal is to infer a model of the unknown function f, enabling us to make predictions for y^* at a point x^* not present in the training set. Recall from Section 4.1 that these predictions enable us to compute the future period's value of a given point in the state space.
In order to make predictions using the information in the dataset D, we need to make assumptions about the characteristics of the underlying functions. A GP is a distribution over functions, defined by a mean function and a covariance function. The mean function, $\mu(x)$, represents the expected value of the function at input x. The covariance function, also referred to as a covariance kernel $k(x, x')$, models the dependency between function values at different input points x and x' by computing a similarity (Williams and Rasmussen, 2006). A GP prior is assigned to the function f, representing our initial knowledge about it before observing any data:

$$f(x) \sim GP(\mu(x), k(x, x')).$$

Choosing an appropriate covariance function is crucial for GPRs and should be based on assumptions on smoothness and likely patterns in the data. A common assumption is that the correlation between two points decays with their distance. A popular choice of kernel that fulfills these assumptions is the Matern kernel:

$$k_{\text{Matern}}(x, x'; \phi) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu d(x, x')} \right)^{\nu} K_{\nu} \left(\sqrt{2\nu d(x, x')} \right),$$

where $d(x, x') = (x - x')^{\top} (I_{\ell})^{-2} (x - x')$ is the Euclidean distance scaled by the length scale ℓ, $K_{\nu}(\cdot)$ is a modified Bessel function, and $\Gamma(\cdot)$ is the gamma function. The kernel has hyperparameters $\phi = \{\nu, \ell\}$, where ν is a smoothness parameter which takes the values $\{0.5, 1.5, 2.5\}$ and is chosen by the modeler, and $\ell = [\ell_1, \ldots, \ell_d]$ with $\ell_j > 0$ is the characteristic lengthscale of the jth input dimension. The Matern kernel is particularly useful for modeling functions with abrupt changes and high non-linearity.
Once we have collected observations $D = \{X, y\}$, by, for example, solving N Bellman equations at various locations within a set B, our goal is to make predictions for new inputs stored in matrix X^* by drawing the corresponding f^* from the posterior distribution $p(f|D)$. The key GP assumption states that the observations y and the unobserved function values f^* jointly follow a multivariate normal distribution, which can be written as:

$$\begin{bmatrix} y \\ f^* \end{bmatrix} \sim \mathcal{N} \left(0, \begin{bmatrix} k(X, X) + \sigma_{\epsilon}^2 I & k(X, X^*) \\ k(X^*, X) & k(X^*, X^*) \end{bmatrix} \right),$$

where $k(A, B)$ is the pairwise measure of the similarity between all points in matrix A and all points in matrix B (as given by the chosen kernel), X and X^* are the training inputs and out-of-sample observations,
we make two assumptions to capture insights about the NTR's shape. For a dynamic portfolio optimization problem with D risky assets and proportional transaction costs, the assumptions read: A1: The NTR has 2^D vertices. A2: he NTR is a D-dimensional convex polytope.
Assumption A1 states that, in settings with two risky assets, the NTR has $2^D = 4$ verticesæ By Assumption A2, the 2-dimensional NTR is then defined by connecting these vertices with straight lines.
The intuition suggests that, in order to pin-point the vertices of an NTR, the optimization problem must be solved for one state-space point in each region to obtain the corresponding *delta*-policy. Therefore, we first sample a minimum of 2^D points $\tilde{X} = \{\tilde{x_i}\}_{i=1}^{2^D}$ one for each vertex, and compute each point's optimal policy $\{\hat{\tilde{\delta}}_i\}_{i=1}^{2^D}$ Then, we compute the NTR vertices $\{\hat{\omega}_i\}_{i=1}^{2^D}$ where $\hat{\omega}_i = \tilde{x_i} + \hat{\tilde{\delta}}_i$ Using Assumption A2, we define the NTR as the convex hull around the vertices, that is, the NTR is defined as any linear combination of the $\hat{\Omega}_t = \{\lambda \hat{\omega}_t \mid \lambda \in (0,1), \sum_{i=1}^{N} \lambda_i = 1\}$ Algorithm 1 summarizes these steps. For an unknown NTR the best way to sample the points for the optimizer is the simplex of the origin. and the midpoints between each combination of simplex vertices. For a 2-dimensional state space, the $2^2 = 4$ points are: [0,0], [1,0], [0,1], [0.5, 0.5].

Determine δ bounds:
δ was decomposed into its respective buying and selling components in order to better handle the kinks in the value functions induced by the proportional transaction costs, that is, we defined $\delta = \delta_i^+ - \delta_i^-$, with $\delta_i^+, \delta_i^- \geq 0$ for all $i \in [1, ..., D]$. A challenge that arises from this is how to enforce that the agent does not buy and sell the ith asset simultaneously. A simple method is to guess which assets the agent buys and which they sell and to encode the problem so as to only allow this configuration. In the worst case, we must try every configuration of guesses (2^D combinations). The approximated NTR provides valuable information concerning the direction wealth that will be reallocated.
We first define a reference point using the approximated NTR vertices. If we assume A2, any linear

combination of the NTR vertices lies inside the approximated NTR. Therefore, we compute the NTR mid-point by averaging the vertices. This ensures that the bounds for ambiguous points that have at least one asset which is not updated ($\delta_i = 0$) would also be correctly determined. Then, to determine the policy bounds, we simply compare the location of a state point to the said mid-point. If the point lies below the mid-point on a given dimension, we solve for δ_i^+; otherwise, we solve for δ_i^-. Multiple Gaussian process regressions: Finally, approximating the NTR allows us to use multiple GPRs to approximate different value function regions of the state space. As the NTR creates a kink in the value function around the boundary, we can use two GPRs to avoid having to approximate the kinks with smooth approx- imators. Instead, we fit one GPR with state points that lie inside the approximated NTR and one with points outside of the approximated NTR (see Fig. 3b). Formally, after solving the optimization problem for each point in the state space, we create two datasets: $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$ consisting of allocations x and value function approximations either in or outside the NTR. Then, in each period t, one GP is fit over each of the respective training sets. Then, we use the corresponding predictive means $\tilde{\mu}(\tilde{x}_t) = k(\tilde{x}_t, \mathbf{X}_t)\left[k(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \hat{\mathbf{v}}_t$,
to approximate the value function for a $\tilde{x}_t$ in the state space, where $\tilde{v}_t = [\tilde{v}_{1,t}, ..., \tilde{v}_{N,t}]^\top$. If $\tilde{x}_t$ is inside of the approximated NTR then X_t and $\tilde{v}_t$ are composed of the points in $\mathcal{D}_{in,t}$ Otherwise, if $\tilde{x}_t$ is outside of the NTR then X_t and $\tilde{v}_t$ are composed of the points in $\mathcal{D}_{out,t}$.

The novel algorithm we propose for solving dynamic portfolio choice problems in discrete time combines DP, GPR's to approximate the value and policy functions, and an approximation of the NTR. Algorithm 2 summarizes the full solution method, which is outlined as follows. Starting from the terminal period T, we iterate backwards through time. For each DP iteration, we first define the value function surrogate used to complete the Bellman equation (Eq. (10)), that is, in period T, we use the terminal value function as specified by Equation (11). Then, we use Algorithm 1 to compute an approximation of the $t-1$st NTR $\hat{\Omega}_{t-1}$. We do this by sampling 2^D points $\tilde{\mathbf{X}}_{t-1} = \{\tilde{x}_{i,t-1}\}_{i=1}^{2^D}$.
For each point in $\tilde{\mathbf{X}}_{t-1}$, we solve for the corresponding $\tilde{\delta}_{t-1}$-policy, and use it to compute the vertices $\hat{\omega}_{t-1}$. We use the approximated NTR $\hat{\Omega}_{t-1}$ to sample N points from the state space $\mathbf{X}_{t-1}$ (as outlined in Sec. 4.3.2). For each point in $\mathbf{X}_{t-1}$, we solve for the optimal policy $\{\delta_{i,t-1}, c_{i,t-1}\}_{i=1}^N$ and corresponding value $\{\hat{v}_{i,t-1}\}_{i=1}^N$. Again, we do this by solving the recursive problem as defined by the Bellman equation (Eq. (10)) and laws of motion (Eqs. (7)-(9)).
Next, we use the approximated NTR to distinguish state points that fall inside and outside of the NTR, respectively, as defined by Equations (20) and (21). Finally, we train two GPRs to approximate the value functions by representing them with fully probabilistic GP models: one on the state points inside the NTR and one on those outside of the NTR.
This concludes the first iteration of our algorithm. We repeat these steps for the remaining iterations $t \in [T-1, ..., 1]$ using the previous GP approximations of the value function in the next iteration. Furthermore, as we train multiple GPs in each period (on points inside and outside of the NTR), we must determine whether the next period's state $x_{i,t+1}$ (as given by Eq. (9)) falls inside or outside of the corresponding NTR approximation $\hat{\Omega}_{t+1}$ and use the correct GP in the Bellman operator.

Description of the procedure

Starting from period T iterate backwards through time. For each iteration, we define the value function surrogate V used to complete the Bellman equation.

Algorithms

Algorithm 1: Approximate the t-th period NTR in the discrete-time finite-horizon portfolio choice model with proportional transaction costs.

Input: $t+1$st period's value function (surrogate) V_{t+1}.
Data: Set of $N = 2^D$ points $\tilde{X}_t = \{\tilde{x}_{i,t}\}_{i=1}^N \subseteq \mathcal{B}$.
Result: Set of approximated NTR vertices: $\{\tilde{\omega}_{i,t}\}_{i=1}^N$. Approximated NTR: $\hat{\Omega}_t$.

1. **For** each $\tilde{x}_{i,t} \in \tilde{X}_t$:
   a) Obtain policy $\hat{\delta}_{i,t}$ for $\tilde{x}_{i,t}$ by solving the optimization problem given by the Bellman equation (Eq. (10)) using V_{t+1} as next period's value function.
   b) Compute the approximate NTR vertices $\tilde{\omega}_{i,t} = \tilde{x}_{i,t} + \hat{\delta}_{i,t}$.
2. **End for**

3. Compute the NTR approximation:

$$\hat{\Omega}_t = \{\lambda\tilde{\omega}_t \mid \lambda \in (0,1)^N, \sum_{i=1}^N \lambda_i = 1\}.$$

Algorithm 2: Dynamic programming with Gaussian process regressions and the NTR approximation for discrete-time finite-horizon portfolio optimization problems with proportional transaction costs.

Input: Terminal value function v_T; time horizon T; sample size N.
Result: Set of GP surrogates of the value functions $\{v_{t-1}\}_{t=0}^{T-1}$; set of approximated NTRs $\{\hat{\Omega}_t\}_{t=0}^{T-1}$.

1. Set $V_T = v_T$ (Eq. (11)).
2. **For** $t \in \{T, \dots, 1\}$ **do**:
   a) Approximate NTR $\hat{\Omega}_{t-1}$ (Algorithm 1) using V_t as the next period's value function.
   b) Sample N points $X_{t-1} = \{x_{i,t-1}\}_{i=1}^N \subseteq \mathcal{B}$ from a D-dimensional simplex.
   c) **For** $x_{i,t-1} \in X_{t-1}$ **do**:
      i. Obtain value $\hat{v}_{i,t-1}$ and policy $\{\hat{\delta}_{i,t-1}, \hat{c}_{i,t-1}\}$ for $x_{i,t-1}$ by solving the optimization problem given by the Bellman equation (Eq. (10)) using V_t as the next period's value function.
   d) **End for**
3. Define the training sets:

$$\mathcal{D}_{\text{in},t-1} = \left\{(x_{i,t-1}, \hat{v}_{i,t-1}) : x_{i,t-1} \in \hat{\Omega}_{t-1}\right\},$$

$$\mathcal{D}_{\text{out},t-1} = \left\{(x_{i,t-1}, \hat{v}_{i,t-1}) : x_{i,t-1} \notin \hat{\Omega}_{t-1}\right\}.$$

4. Given $\mathcal{D}_{\text{in},t-1}$ and $\mathcal{D}_{\text{out},t-1}$, learn a surrogate of v_{t-1} for inside and outside of the NTR $\{g_{\text{in},t-1}, g_{\text{out},t-1}\}$ (using the respective datasets) with GPs.
5. Set $V_{t-1} = \{g_{\text{in},t-1}, g_{\text{out},t-1}\}$.
6. **End for**

Note on multidimensional numerical integration using quadrature.

In the objective function of the Bellman equations, we often need to compute the conditional expectation of $V(x^+ \mid \mathbf{x}, \mathbf{a})$. When the random variable is continuous, we have to use numerical integration to compute the expectation.
One naive way is to apply Monte Carlo or pseudo Monte Carlo methods to compute the expectation. By the Central Limit Theorem, the numerical error of the integration computed by (pseudo) Monte Carlo methods has a distribution that is close to normal, and so no bound for this numerical error exists. Moreover, the optimization problem often needs hundreds or thousands of evaluations of the objective

function. This implies that once one evaluation of the objective function has a big numerical error, the previous iterations to solve the optimization problem may make no sense, and the iterations may never converge to the optimal solution. Thus it is not practical to apply (pseudo) Monte Carlo methods to the optimization problem generally, unless the stopping criterion of the optimization problem is set very loosely.

Therefore, it will be better to have a numerical integration method with a bounded numerical error. Here we present a common numerical integration method to use when the random variable is normal.

In the expectation operator of the objective function of the Bellman equation, if the random variable has a normal distribution, then it will be good to apply the Gauss-Hermite quadrature formula to compute the numerical integration. That is, if we want to compute $\mathbb{E}\left\{f(Y)\right\}$ where Y has a distribution $\mathcal{N}(\mu, \sigma^2)$, then

$$\begin{aligned}\mathbb{E}\left\{f(Y)\right\} &= (2\pi\sigma^2)^{-1/2}\int_{-\infty}^{\infty} f(y)e^{-(y-\mu)^2/(2\sigma^2)}dy \\ &= (2\pi\sigma^2)^{-1/2}\int_{-\infty}^{\infty} f(\sqrt{2\sigma}x+\mu)e^{-x^2}\sqrt{2\sigma}dx \\ &\approx \pi^{-1/2}\sum_{i=1}^{m}\omega_i f\left(\sqrt{2\sigma}x_i+\mu\right),\end{aligned}$$

where ω_i and x_i are the Gauss-Hermite quadrature weights and nodes over $(-\infty, \infty)$, and m is the number of quadrature nodes.

If Y is log normal, i.e., $\log(Y)$ has a distribution $\mathcal{N}(\mu, \sigma^2)$, then we can assume that $Y = e^X$ where $X \sim \mathcal{N}(\mu, \sigma^2)$, thus

$$\mathbb{E}\left\{f(Y)\right\} = \mathbb{E}\left\{f(e^X)\right\} \approx \pi^{-1/2}\sum_{i=1}^{m}\omega_i f\left(e^{\sqrt{2\sigma}x_i+\mu}\right).$$

If we want to compute a multidimensional integration, we could apply the product rule. For example, suppose that we want to compute $\mathbb{E}\left\{f(\mathbf{X})\right\}$, where $\mathbf{X}$ is a random vector with multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ over $(-\infty, +\infty)^k$, where μ is the mean column vector and Σ is the covariance matrix, then we could do the Cholesky factorization first, i.e., find a lower triangular matrix L such that $\Sigma = LL^\top$. This is feasible as Σ must be a positive semi-definite matrix, from the covariance property. Thus,

$$\begin{aligned}\mathbb{E}\left\{f(\mathbf{X})\right\} &= ((2\pi)^k \det(\Sigma))^{-1/2}\int_{\mathbb{R}^k} f(\mathbf{y})e^{-(\mathbf{y}-\mu)^\top\Sigma^{-1}(\mathbf{y}-\mu)/2}d\mathbf{y} \\ &= ((2\pi)^k \det(L^2))^{-1/2}\int_{\mathbb{R}^k} f\left(\sqrt{2}L\mathbf{x}+\mu\right)e^{-\mathbf{x}^\top\mathbf{x}/2}\det(L)d\mathbf{x}\end{aligned}$$

$$\approx \pi^{-k/2}\sum_{i_1=1}^{m}\cdots\sum_{i_k=1}^{m}\omega_{i_1}\cdots\omega_{i_k} f\left(\sqrt{2}L_{1,1}x_{i_1}+\mu_1, \sqrt{2}\left(L_{2,1}x_{i_1}+L_{2,2}x_{i_2}\right)+\mu_2\ldots, \sqrt{2}L_{1,k}x_{i_k}+\mu_k\right),$$

where ω_i and x_i are the Gauss-Hermite quadrature weights and nodes over $(-\infty, \infty)$, $L_{i,j}$ is the (i,j)-element of L, and $\det(\cdot)$ means the matrix determinant operator.