Optimal control by policy iterations and constrained Gaussian process regressions

Donatien Hainaut*
UCLouvain - LIDAM
Jean-Loup Dupret†
ETH Zürich - RiskLab

This article introduces a novel iterative algorithm that combines policy iterations with Gaussian process regressions to solve stochastic control problems. At each iteration, an approximate value function is updated, followed by an improvement of the control strategy. State variables are sampled both within the interior domain and on the terminal boundary of the Hamilton-Jacobi-Bellman (HJB) equation. The approximate value function, interpreted as the solution to the HJB equation, is obtained by fitting a constrained regression model. This regression function aligns with the terminal utility on the boundary sample and satisfies the HJB equation on the interior sample. Assuming the regression function follows a Gaussian process, we derive closed-form approximations for both the value function and its derivatives. A numerical illustration demonstrates the efficiency of the proposed method in solving the consumption-investment problem, the linear-quadratic regulator, and a constrained consumption-investment problem with stochastic volatility. As benchmarks, we compare our results with those obtained using backward partial differential equation (PDE) methods and Physics-Informed Neural Networks (PINNs).

Keywords: Optimal control, Gaussian process regression, Hamilton-Jacobi-Bellman equation, reinforcement learning, policy iterations.

1 Introduction

This work introduces a novel iterative algorithm based on constrained Gaussian process regressions (CGPR) for solving optimal control problems in continuous time. We show that this method is an accurate and numerically efficient alternative to existing approaches. Its methodological foundations are Gaussian process regressions and policy iterations.

Gaussian processes are used for regressing a response y on a vector of covariates $\boldsymbol{x}$. We refer the reader to Rasmussen and Williams (2006, chapter 6) or to Murphy (2012, chapter 15) for a detailed presentation. The Gaussian process regression is a Bayesian method using as a-priori distribution for responses y, a Gaussian process (GP), with zero mean and covariance $k(\boldsymbol{x}, \boldsymbol{x\prime})$, called kernel. This choice is made prior to taking into account observations. The regression

*Corresponding author. Postal address: Voie du Roman Pays 20, 1348 Louvain-la-Neuve Belgium. E-mail to: donatien.hainaut(at)uclouvain.be
†Postal address: Rämistrasse 101HG F 428092 Zurich Switzerland E-mail to: jeanloup.dupret@math.ethz.ch

function is the a-posteriori expectation of this GP, conditioned by observations. Seminal works discussing the use of GPs as surrogate models for computational science and engineering applications include the papers of Sacks et al. (1989) and Santner et al. (2003), and the book by Rasmussen and Williams (2006). Without being exhaustive, GPs are used in various fields such as time-series, Roberts et al. (2013), data mining of large datasets, Banerjee (2013), materials sciences, Deringer et al. (2021), operational research, Price et al. (2019), budget decision, Hu et al. (2025), or missing data, Choi et al. (2024).

In a similar manner to Physics-Informed Neural Networks (PINNs), we can constrain a Gaussian process regression with a partial differential equation (PDE). This PDE represents the physical law governing the experiment from which responses are measured. We call this category of models, Constrained Gaussian Process Regressions (CGPRs), or Physics-Informed Gaussian Processes. The literature on CGPR is vast, and we refer the reader to Swiler et al. (2020) for a detailed survey. Among the references therein, we highlight Graepel (2003), who approximates the solution of PDEs with noisy responses using a CGPR. Calderhead et al. (2009) who present an accelerated sampling procedure to solve nonlinear ordinary and delay differential equations with Gaussian processes. Cialenco et al. (2012) who combine an Euler discretization scheme on the time axis with a CGPR. Barber and Wang (2014), and Cockayne et al. (2019) who propose a CGPR that improves the gradient matching technique. Särkkä (2011) and Nguyen and Peraire (2015) develop procedures for handling responses of the PDE solution, given in the form of linear functionals of the state variables. Pförtner et al. (2022) consider physics-informed Gaussian process regression for solving both weak and strong formulations of linear PDEs.

In this article, we consider a Gaussian Process Regression constrained by a PDE has the form $\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}}g = z(\boldsymbol{x})$ where g, $\mathcal{L}_{x,\boldsymbol{c}}$, $\boldsymbol{x}$ and $z(\boldsymbol{x})$ are respectively, the value, a linear differential operator controlled by $\boldsymbol{c}$, a vector of risk processes and a non-linear function. The boundary constraint is $g(\boldsymbol{x}) = h(\boldsymbol{x})$, where $h(\boldsymbol{x})$ is the terminal utility at expiry. Our framework differs from the existing literature on CGPRs by managing the non-linearity of $z(\boldsymbol{x})$ with policy iterations combined to the methodology developed in Hainaut and Vrins (2024) and Hainaut (2024) for pricing financial derivatives. We fit a constrained regression on two samples of points, $\mathring{\boldsymbol{X}}$ and $\bar{\boldsymbol{X}}$, respectively in the inner and boundary domains of the PDE. In this setting, the solution is a regression function $g(\boldsymbol{x})$ with $g(\boldsymbol{x}) = h(\boldsymbol{x})$ for all $\boldsymbol{x} \in \bar{\boldsymbol{X}}$, which satisfies a PDE constraint, $\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}}g = z(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathring{\boldsymbol{X}}$.

A parallel can be drawn with methods based on Physics-Informed Neural Networks (PINNs), in which the constrained Gaussian process regression (CGPR) is replaced by a neural network. For a general overview of PINNs, we refer the reader to Raissi et al. (2019). Additional examples of PINNs applied to financial valuation and stochastic control include Sirignano and Spiliopoulos (2018), Al-Aradi et al. (2022), Glau and Wunderlich (2022, 2024), Hainaut (2024), and Hainaut and Casas (2024). As detailed in this article, a key drawback of PINNs compared to CGPR is their reliance on numerical differentiation to solve partial differential equations (PDEs). In contrast, CGPR approximations using Gaussian kernels are fully analytical. Moreover, the PI-CGPR framework leverages strong prior assumptions through Bayesian inference, enabling it to generalize effectively from limited data—whereas PINN methods typically require larger datasets to learn underlying structures from scratch.

On the other hand, the value function, $V(\boldsymbol{x})$, of a stochastic optimal control problem in continuous time, is the solution of the Hamilton-Jacobi-Bellman (HJB) equation. In many cases, the optimal control admits an explicit representation in terms of the partial derivatives of $V(\boldsymbol{x})$, and the HJB equation—once the optimal controls are substituted—becomes a nonlinear partial differential equation (PDE). For further details, we refer the reader to Touzi (2012). Since CGPR

is designed for linear PDEs, it cannot be directly applied in this context. A solution inspired by the reinforcement learning literature involves using a policy iteration (PI) algorithm, which iteratively approximates the value function and updates the control strategy. We refer the reader to Howard (1960), Werbos (1992) or Sutton and Barto (2018) for a detailed presentation, and to Jacka and Mijatovic (2017) for an analysis of the convergence. The PI algorithm is used for various purposes from process quality improvement, Fine et Porteus (1989) or Bachouch et al. (2022) to deep learning for stochastic control, Dupret et Hainaut (2024). For a survey, we refer to Bertsekas (2011).

In this article, we combine the policy iteration (PI) algorithm with constrained Gaussian process regression (CGPR) to solve stochastic control problems in continuous time. This approach differs from existing literature, where PI and Gaussian processes are typically applied to discrete-time and deterministic optimization problem, as in the seminal work of Rasmussen and Kuss (2004) or later in Mlakar et al. (2014) and Jakubik et al. (2021). In our framework, the HJB equation is solved using CGPR for a given control at each iteration, after which the control is updated to improve the value function in the subsequent step. Numerical illustrations confirm the efficiency of this approach, which offers a novel alternative to existing methods for solving optimal control problems. To the best of our knowledge, the PI-CGPR framework is new to the literature. The most closely related work is the recent paper by Yang and Zhang (2025), which also explores Gaussian process-based policy iteration. However, our approach differs by providing analytical expressions for the value function and its partial derivatives, along with a detailed comparison against PINNs and PDE-based methods across three examples.

The article is organized as follows. We first present the optimal control problem and review the algorithm of policy iterations (PI). We prove that under mild assumptions, that this procedure increases the value function at each iteration. Section 3 starts with a brief review of general principles of the CGPR. Next we present the full algorithm for solving the HJB equation by policy iterations and constrained Gaussian process regressions (PI-CGPR algorithm). In section 4, we develop the algorithm in the case of Gaussian kernels. These kernels allow us to find analytical expressions of the approximate value function and controls. In Section 5, we provide a thorough numerical analysis of the PI-CGPR algorithm. We solve the investment-consumption, the linear-regulator problems and the constrained investment-consumption problem with stochastic volatility. As benchmarks, we compare our results with those obtained using PDE-based methods and Physics-Informed Neural Networks (PINNs).

2 Optimal control by policy iterations

We consider a system ruled by a $p-1$ vector of risk processes[1],

$$(\boldsymbol{Y}_t)_{t\geq 0} = \left(\left(Y_t^{(1)}, ..., Y_t^{(p-1)}\right)\right)_{t\geq 0}.$$

They are defined on a complete probability space, Ω, endowed with a filtration $\mathbb{F} = (\mathcal{F}_t)_{t\geq 0}$ and a probability measure, $\mathbb{P}$. We assume that risk processes are ruled by a system of stochastic differential equations (SDEs):

$$d\boldsymbol{Y}_t \quad = \quad \boldsymbol{\mu}_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t, \boldsymbol{c}_t)dt + \Sigma_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t, \boldsymbol{c}_t)d\boldsymbol{B}_t\,, \tag{1}$$

where $\boldsymbol{B} = (\boldsymbol{B}_t)_{t\geq 0}$ is a v-vector of independent Brownian motions, $\boldsymbol{\mu}_{\boldsymbol{Y}}(.)$ is a vector of dimension $(p-1)$ and $\Sigma_{\boldsymbol{Y}}(.)$ is a $(p-1)\times v$ matrix. The $\mathbb{R}^l$-valued control $(\boldsymbol{c}_t)_{t\geq 0}$ is $\mathbb{F}$-adapted and takes values in a subset A of $\mathbb{R}^l$.

[1] We use bold notations for vectors.

The set of admissible strategies is denoted by $\mathcal{A}$. In later developments, we consider $\mathbb{F}$-adapted Markov controls that can be written as $\boldsymbol{c}_t = \boldsymbol{c}(t, \boldsymbol{Y}_t)$, $\forall t \geq 0$ for some measurable map $\boldsymbol{c} : \mathbb{R}^+ \times \mathbb{R}^{p-1} \to \mathbb{R}^l$, with upper and lower bounds:

$$\mathcal{A} = \{(c_t)_{t\geq 0} \mid \mathbb{F} - \text{adapted and } \boldsymbol{c}_{min}(t, \boldsymbol{Y}_t) \leq \boldsymbol{c}_t \leq \boldsymbol{c}_{max}(t, \boldsymbol{Y}_t)\},$$

where $\boldsymbol{c}_{min}(t, \boldsymbol{Y}_t)$ and $\boldsymbol{c}_{max}(t, \boldsymbol{Y}_t)$ are general boundary functions on controls from $\mathbb{R}^+ \times \mathbb{R}^{p-1} \to A \subset \mathbb{R}^l$. The functions $\boldsymbol{\mu_Y} : \mathbb{R}^+ \times \mathbb{R}^{p-1} \times A \to \mathbb{R}^{p-1}$ and $\Sigma_{\boldsymbol{Y}} : \mathbb{R}^+ \times \mathbb{R}^{p-1} \times A \to \mathbb{R}^{(p-1)\times v}$ satisfy a condition of linear growth and are Lipschitz. Furthermore, $\boldsymbol{\mu_Y}(.,.,\boldsymbol{c})$ and $\Sigma_{\boldsymbol{Y}}(.,.,\boldsymbol{c})$ are of class C^1 in t and $\boldsymbol{y}$. Those conditions guarantee the existence of a unique solution to SDE (1). Next, we define the criterion to be maximized, $J(t, \boldsymbol{y}, \boldsymbol{c})$, that we name payoff. Let U_1, U_2 be continuous positive functions which satisfy an usual polynomial growth conditions. Given the pair $\boldsymbol{x} = (t, \boldsymbol{y}) \in \mathbb{R}^+ \times \mathbb{R}^{p-1}$ and an admissible control[2] $\boldsymbol{c}$, we define the payoff as follows

$$J(\boldsymbol{x}, \boldsymbol{c}) = \mathbb{E}\left(\int_t^T e^{-\alpha(s-t)} U_1(s, \boldsymbol{Y}_s, \boldsymbol{c}_s) ds + e^{-\alpha(T-t)} U_2(T, \boldsymbol{Y}_T) \mid \boldsymbol{Y}_t = \boldsymbol{y}\right), \tag{2}$$

where T is the time horizon of the problem and α is a discount rate. Let $\mathring{\mathcal{X}} = [0, T) \times \mathbb{R}^{(p-1)}$, $\bar{\mathcal{X}} = T \times \mathbb{R}^{(p-1)}$ and $\mathcal{X} = \mathring{\mathcal{X}} \cup \bar{\mathcal{X}}$. We aim to find the control $(\boldsymbol{c}_t)_{t\geq 0}$ maximizing $J(\boldsymbol{x}, \boldsymbol{c})$. The value function $V(\boldsymbol{x})$ for $\boldsymbol{x} = (t, \boldsymbol{y})$ is the maximum of Eq. (2) among all admissible strategies:

$$V(\boldsymbol{x}) = \max_{c\in\mathcal{A}} J(\boldsymbol{x}, \boldsymbol{c}). \tag{3}$$

If we adopt the notation $\boldsymbol{X}_t = (t, \boldsymbol{Y}_t)$ and use nested expectations, we can prove that for any stopping time θ such that $t \leq \theta \leq T$,

$$V(\boldsymbol{x}) = \max_{\boldsymbol{c}\in\mathcal{A}} \mathbb{E}\left(\int_t^\theta e^{-\alpha(s-t)} U_1(\boldsymbol{X}_s, \boldsymbol{c}_s)\, ds + e^{-\alpha(\theta-t)} V(\boldsymbol{X}_\theta) \mid \boldsymbol{Y}_t = \boldsymbol{y}\right). \tag{4}$$

Assuming that $V \in C^{1,2}(\mathring{\mathcal{X}})$, the gradient and the Hessian of $V(.)$ with respect to $\boldsymbol{y}$ are denoted by $\nabla_{\boldsymbol{y}} V$ and $\mathcal{H}_{\boldsymbol{y}}(V)$. Let us define the linear differential operator:

$$\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}} \cdot = \partial_t \cdot - \alpha \cdot + \boldsymbol{\mu_Y}(t, \boldsymbol{y}, \boldsymbol{c})^\top \nabla_{\boldsymbol{y}} \cdot + \frac{1}{2} \mathrm{tr}\left(\Sigma_{\boldsymbol{Y}}(t, \boldsymbol{y}, \boldsymbol{c}) \Sigma_{\boldsymbol{Y}}(t, \boldsymbol{y}, \boldsymbol{c})^\top \mathcal{H}_{\boldsymbol{y}} \cdot\right). \tag{5}$$

Using standard arguments, the solution to the problem (4) satisfies the following Hamilton-Jacobi-Bellman (HJB) equation:

$$\sup_{\boldsymbol{c}\in A} \left(U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}} V(\mathring{\boldsymbol{x}})\right) = 0 \quad \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}, \tag{6}$$

$$V(\bar{\boldsymbol{x}}) = U_2(\bar{\boldsymbol{x}}) \quad \bar{\boldsymbol{x}} \in \bar{\mathcal{X}}. \tag{7}$$

We solve this problem with the policy iteration (PI) algorithm. The PI algorithm yields an intuitive approach to optimal control by generating a sequence of controls $\boldsymbol{c}^{(n)} = \left(\boldsymbol{c}_t^{(n)}\right)_{t\geq 0}$, which improve the payoffs $J(\boldsymbol{x}, \boldsymbol{c}^{(n)})$, at each iteration. Let us assume that after $(n-1)^{th}$ steps, we have obtained a policy $\boldsymbol{c}^{(n-1)}$ and let us denote by

$$V^{(n-1)}(\boldsymbol{x}) = J(\boldsymbol{x}, \boldsymbol{c}^{(n-1)}) \quad , \quad \forall x \in \mathcal{X}$$

the corresponding payoff. We first update the control as follows

$$\boldsymbol{c}^{(n)} = \arg\sup_{\boldsymbol{c}\in A} \left(U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}} V^{(n-1)}(\mathring{\boldsymbol{x}})\right), \forall \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}. \tag{8}$$

[2]Notice that $\boldsymbol{c}$ either refers to the map in $\mathcal{A}$ or to the control value in A depending upon the context.

$\boldsymbol{c}^{(n)}$ is called an improvement of $\boldsymbol{c}^{(n-1)}$. Once the $n^{th}-$optimal control is determined, we calculate $V^{(n)}(\boldsymbol{x})$ solving

$$\begin{cases} \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n)}(\mathring{\boldsymbol{x}}) = -U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) & \forall \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}\,, \\ V^{(n)}(\bar{\boldsymbol{x}}) = U_2(\bar{\boldsymbol{x}}) & \forall \bar{\boldsymbol{x}} \in \bar{\mathcal{X}}\,. \end{cases} \tag{9}$$

We explain in the next section how to find an approximate solution with a Gaussian process regression. In later developments of this section, we assume that the exact solution exists and show that each iteration improves the value function.

It is also interesting to mention that from Eq. (9), $\mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n)}(\mathring{\boldsymbol{x}}) \leq 0$ as $U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) \geq 0$. The next proposition is an intermediate result that is used later to demonstrate that $V^{(n)}(\boldsymbol{x})$ is improved at each iteration.

Proposition 1. *$\forall \boldsymbol{x} \in \mathcal{X}$, the following inequality holds:*

$$\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}} \left(V^{(n)}(\boldsymbol{x}) \right) \;\leq\; \mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}} \left(V^{(n-1)}(\boldsymbol{x}) \right) . \tag{10}$$

The proof is provided in Appendix C. To avoid any confusion in later developments, we temporarily denote by $\left(\boldsymbol{Y}_t^{(n)}\right)_{t\geq 0}$, the process controlled by $\left(\boldsymbol{c}_t^{(n)}\right)_{t\geq 0}$:

$$d\boldsymbol{Y}_t^{(n)} \;=\; \boldsymbol{\mu}_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t^{(n)}, \boldsymbol{c}_t^{(n)})dt + \Sigma_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t^{(n)}, \boldsymbol{c}_t^{(n)})d\boldsymbol{B}_t\,. \tag{11}$$

and $\boldsymbol{X}_t^{(n)} = (t, \boldsymbol{Y}_t^{(n)})$. The next proposition states that each iteration of the PI algorithm increases the value function, under mild assumptions.

Proposition 2. *Under the assumption that the difference of two consecutive value functions converge in L^1 to a non-negative random variable:*

$$\lim_{t\to T} \left(V^{(n)}(\boldsymbol{X}_t^{(n)}) - V^{(n-1)}(\boldsymbol{X}_t^{(n)}) \right) \overset{L_1}{=} Z \geq 0 \quad , \tag{12}$$

and that $\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}} V^{(n)}(\boldsymbol{x})$ is well defined, the value function increases at each iteration:

$$V^{(n)}(\boldsymbol{x}) \;\geq\; V^{(n-1)}(\boldsymbol{x}) \quad , \quad \forall x \in \mathcal{X}.$$

The proof is available in Appendix C. In most cases the assumption 12 is satisfied: for a problem with a finite time horizon, $V^{(n)}(\boldsymbol{X}_T^{(n)}) = V^{(n-1)}(\boldsymbol{X}_T^{(n)}) = U_2(T, \boldsymbol{Y}_T)$ and $Z = 0$. Under additional regularity assumptions, Jacka and Mijatovic (2017) prove the convergence of the controls and the value function. We also refer the reader to Fleming (1963) and Puterman (1981) for an analysis of the convergence of policy iterations for controlled diffusions in a more general setting. The next section develops the method based on constrained Gaussian process regression for approximating the solution of Eq. (9).

3 The constrained Gaussian process regression with policy iterations (PI-CGPR)

We start by briefly reviewing the general principles of the CGPR. We consider a series of Gaussian processes $\{g^{(n)}(\boldsymbol{x})\,,\, \boldsymbol{x} \in \mathcal{X}\}$, where $n \in \mathbb{N}$. By definition, a Gaussian process is a collection such that for any $d \in \mathbb{N}$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d \in \mathcal{X}$, the vector $\left(g^{(n)}(\boldsymbol{x}_1), \ldots, g^{(n)}(\boldsymbol{x}_d)\right)$ is a multivariate Gaussian random variable. Without loss of generality, the mean is set to zero and the covariance function is defined by a kernel $k(\boldsymbol{x}, \boldsymbol{x}')$:

$$\mathbb{C}\left(g^{(n)}(\boldsymbol{x}), g^{(n)}(\boldsymbol{x}')\right) = k(\boldsymbol{x}, \boldsymbol{x}') \quad \forall (\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{X} \times \mathcal{X}\,.$$

A necessary and sufficient condition for the function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to be a valid covariance kernel is that the $d \times d$ matrix of $(k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\dots,d}$ is positive semi-definite for all possible samples.

The PI-CGPR provides an approximation $\widehat{V^{(n)}}(\boldsymbol{x})$, of the objective function $V^{(n)}(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ after n iterations. We will see that $\widehat{V^{(n)}}(\boldsymbol{x})$ is defined as the a posteriori conditional expectation of $g^{(n)}(\boldsymbol{x})$. Let us detail this. In the PI-CGPR framework, the functional form of $V^{(n)}(\boldsymbol{x})$ is unknown, but its values are observed at certain points of $\bar{\mathcal{X}}$ and are ruled by the PDE of Eq. (9) at other points of $\mathring{\mathcal{X}}$. For this reason, we consider two random samples. We define a first set with d samples of state variables at expiry T, denoted as $\bar{\boldsymbol{X}} = \{\bar{\boldsymbol{x}}_i\}_{i=1,\dots,d}$ and corresponding noised terminal utility, $\boldsymbol{h} = (h_i)_{i=1,\dots,d}$. The $\bar{\boldsymbol{x}}_i$'s are randomly sampled from $\bar{\mathcal{X}}$ and $h_i = U_2(\bar{\boldsymbol{x}}_i) + \bar{\epsilon}$ are noised by $\bar{\epsilon} \sim \mathcal{N}(0, \sigma^{*2})$. The second training set contains m samples of state variables before expiry, denoted as $\mathring{\boldsymbol{X}}\mathring{\boldsymbol{X}} = \{\mathring{\boldsymbol{x}}_i\}_{i=1,\dots,m}$ with $\mathring{\boldsymbol{x}}_i \in \mathring{\mathcal{X}}$. The noised values of the PDE of Eq. (9) are grouped in a m-vector, $\boldsymbol{z}^{(n)} = \left(z_i^{(n)}\right)_{i=1,\dots,m}$ with $z_i^{(n)} = -U_1(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) + \mathring{\epsilon}$ and $\mathring{\epsilon} \sim N(0, \sigma^{*2})$.

Assuming noisy observations of $U_2(\bar{\boldsymbol{x}}_i)$ for $\bar{\boldsymbol{x}}_i \in \bar{\boldsymbol{X}}$ and $\mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} V^{(n)}(\mathring{\boldsymbol{x}}_i)$ for $\mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}$, is a way to introduce a regularization term via a linear shrinkage of the covariance matrix $k(\boldsymbol{X}, \boldsymbol{X})$, analogous to Ridge estimators. We will come back to this point later. At each iteration $n \in \mathbb{N}$, we fit a Gaussian process $g^{(n)}$ approximating the value function $V^{(n)}$ over $\bar{\boldsymbol{X}}$ and $\mathring{\boldsymbol{X}}$. The Gaussian process satisfies the following relations on the two training sets:

$$\begin{cases} \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} g^{(n)}(\mathring{\boldsymbol{x}}_i) = z_i^{(n)} & \forall \mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}, \\ g^{(n)}(\bar{\boldsymbol{x}}_i) = h_i & \forall \bar{\boldsymbol{x}}_i \in \bar{\boldsymbol{X}}. \end{cases} \tag{13}$$

Eq. (13) is a discrete noised version of the system (9). We recognize the structure of a constrained regression problem: we fit $g^{(n)}$ such that $\mathbb{E}\left(g^{(n)}(\bar{\boldsymbol{x}})\right) = U_2(\bar{\boldsymbol{x}})$ for $\bar{\boldsymbol{x}} \in \bar{\boldsymbol{X}}$, under the constraint that $\mathbb{E}\left(\mathcal{L}_{\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} g^{(n)}(\mathring{\boldsymbol{x}})\right) = -U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)})$ for $\mathring{\boldsymbol{x}} \in \mathring{\boldsymbol{X}}$. We denote by $\boldsymbol{H}$ and $\boldsymbol{Z}^{(n)}$, the random d- and m-vectors whose realizations are $\boldsymbol{h}$ and $\boldsymbol{z}^{(n)}$, the right-hand term in Eq. (13).

To summarize, we a priori encode our belief that instances of $g^{(n)}(\boldsymbol{x})$, satisfying Eq. (13), are drawn from a Gaussian process with null mean and covariance $k(\boldsymbol{x}, \boldsymbol{x}')$, prior to taking into account observations $\boldsymbol{h}$ and $\boldsymbol{z}^{(n)}$ of terminal utility and HJB PDE. In this Bayesian approach, the estimator of $V^{(n)}(\boldsymbol{x})$ is the a posteriori expectation of $g^{(n)}(\boldsymbol{x})$ conditioned by realizations of $\boldsymbol{Z}^{(n)}$ and $\boldsymbol{H}$:

$$\widehat{V^{(n)}}(\boldsymbol{x}) = \mathbb{E}\left(g^{(n)}(\boldsymbol{x}) \mid \boldsymbol{Z}^{(n)} = \boldsymbol{z}^{(n)}, \boldsymbol{H} = \boldsymbol{h}\right).$$

To estimate $g^{(n)}$, we develop it as a sum a basis functions weighted by random normal weights. This is feasible because the function $g^{(n)}(\boldsymbol{x})$ belongs to a reproducing kernel Hilbert space spanned by eigenfunctions of the kernel (see e.g. Chapter 6 of Rasmussen and Williams, 2006). Without loss of generality, we consider kernels defined by a finite number of eigenfunctions. This assumption is not restrictive as any kernel can be approximated by a finite combination of eigenfunctions for a given accuracy. This implies that the following results asymptotically hold for any kernel with an infinite number of eigenfunctions. In this case, there exist q basis functions $\boldsymbol{\varphi}(\boldsymbol{x}) = (\varphi_j(\boldsymbol{x}))_{j=1,\dots,q}$ such that

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\varphi}(\boldsymbol{x})^\top \boldsymbol{\varphi}(\boldsymbol{x}'), \ \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}.$$

As $g^{(n)}(\boldsymbol{x})$ is a-priori normal with null mean and $\mathbb{C}\left(g^{(n)}(\boldsymbol{x}), g^{(n)}(\boldsymbol{x}')\right) = k(\boldsymbol{x}, \boldsymbol{x}')$, $g^{(n)}(\boldsymbol{x})$ can be written a sum of q functions $\varphi_j(.)$ weighted by a vector $\boldsymbol{w}^{(n)} \sim N(0, I_q)$:

$$g^{(n)}(\boldsymbol{x}) = \sum_{j=1}^{q} w_j^{(n)} \varphi_j(\boldsymbol{x}) = \boldsymbol{w}^{(n)\top} \boldsymbol{\varphi}(\boldsymbol{x}). \tag{14}$$

We define the function $\boldsymbol{\varphi}_{\mathcal{L}^{(n)}}(\boldsymbol{x}) = \left(\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}}\varphi_j(\boldsymbol{x})\right)_{j=1,\dots,q}$ from $\mathcal{X} \to \mathbb{R}^q$. Using the linearity of the operator (5), $\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}}g^{(n)}(\boldsymbol{x})$ is the scalar product:

$$\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}}g^{(n)}(\boldsymbol{x}) = \sum_{j=1}^{q} w_j^{(n)} \left(\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}}\varphi_j(\boldsymbol{x})\right) = \boldsymbol{w}^{(n)\top}\boldsymbol{\varphi}_{\mathcal{L}^{(n)}}(\boldsymbol{x})\,.$$

Next, we denote by $\mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}}$ the $m \times q$ matrix of $(\mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}})_{i,j} = \mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}\varphi_j(\mathring{\boldsymbol{x}}_i)$ where $\mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}$ and by $\bar{\boldsymbol{\varphi}}$, the $d \times q$ matrix of $(\bar{\boldsymbol{\varphi}})_{i,j} = \varphi_j(\bar{\boldsymbol{x}}_i)$ with $\bar{\boldsymbol{x}}_i \in \bar{\boldsymbol{X}}$. Conditionally to $\boldsymbol{w}^{(n)}$, the joint distribution of $(\boldsymbol{Z}^{(n)}, \boldsymbol{H})$, the random vectors of right-hand terms in Eq. (13) is a multivariate normal:

$$\left.\begin{pmatrix} \boldsymbol{Z}^{(n)} \\ \boldsymbol{H} \end{pmatrix}\right| \boldsymbol{w}^{(n)} \sim \mathcal{N}\left(\begin{pmatrix} \mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}}\boldsymbol{w}^{(n)} \\ \bar{\boldsymbol{\varphi}}\boldsymbol{w}^{(n)} \end{pmatrix}, \begin{pmatrix} \sigma^{*2}I_m & \boldsymbol{0} \\ \boldsymbol{0} & \sigma^{*2}I_d \end{pmatrix}\right). \tag{15}$$

From the properties of the multivariate normal distribution, the regression weights $\boldsymbol{w}^{(n)}$, conditionally to $\boldsymbol{z}^{(n)}$ and $\boldsymbol{h}$, are realizations of a normal distribution $\boldsymbol{W}^{(n)}$:

$$\boldsymbol{W}^{(n)}|\boldsymbol{z}^{(n)}, \boldsymbol{h} \;\sim\; \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{z}^{(n)},\boldsymbol{h}}\,;\, \Sigma_{\boldsymbol{z}^{(n)},\boldsymbol{h}}\right),$$

where $\boldsymbol{\mu}_{\boldsymbol{z}^{(n)},\boldsymbol{h}}$, and $\Sigma_{\boldsymbol{z}^{(n)},\boldsymbol{h}}$ are respectively a q-vector and a $(q \times q)$-matrix equal to

$$\begin{cases} \Sigma_{\boldsymbol{z}^{(n)},\boldsymbol{h}} &= \left(I_q + \sigma^{*-2}\mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}}^{\top}\mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}} + \sigma^{*-2}\bar{\boldsymbol{\varphi}}^{\top}\bar{\boldsymbol{\varphi}}\right)^{-1}, \\ \boldsymbol{\mu}_{\boldsymbol{z}^{(n)},\boldsymbol{h}} &= \sigma^{*-2}\Sigma_{\boldsymbol{z}^{(n)},\boldsymbol{h}}\left(\mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}}^{\top}\boldsymbol{z}^{(n)} + \bar{\boldsymbol{\varphi}}^{\top}\boldsymbol{h}\right). \end{cases} \tag{16}$$

In following developments, we do not need the exact expression of basis functions. All results depend on their product that is equal to the kernel. We adopt for this reason the following notations:

$$\begin{cases} k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \boldsymbol{x}\right) &= \mathring{\boldsymbol{\varphi}}_{\mathcal{L}^{(n)}}\boldsymbol{\varphi}(\boldsymbol{x}) = \left(\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)\right)_{i=1,\dots,m}, \\ k\left(\bar{\boldsymbol{X}}, \boldsymbol{x}\right) &= \left(k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right)\right)_{j=1,\dots,d}, \\ k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) &= \mathring{\boldsymbol{\varphi}}_{\mathcal{L}}\bar{\boldsymbol{\varphi}}^{\top} = \left(\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k(\bar{\boldsymbol{x}}_j, \mathring{\boldsymbol{x}}_i)\right)_{i=1,\dots,m,j=1,\dots,d}, \\ k_{\mathcal{L}^{2}{}^{(n)}}\left(\mathring{\boldsymbol{X}}, \mathring{\boldsymbol{X}}\right) &= \mathring{\boldsymbol{\varphi}}_{\mathcal{L}}\mathring{\boldsymbol{\varphi}}_{\mathcal{L}}^{\top} = \left(\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}\mathcal{L}_{\mathring{\boldsymbol{x}}_j,\boldsymbol{c}^{(n)}}k(\mathring{\boldsymbol{x}}_i, \mathring{\boldsymbol{x}}_j)\right)_{i,j=1,\dots,m}, \\ k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) &= \bar{\boldsymbol{\varphi}}\bar{\boldsymbol{\varphi}}^{\top} = \left(k(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{x}}_j)\right)_{i,j=1,\dots,d}. \end{cases}$$

The analytical expression of the above equations will be developed in the next section when the kernel is Gaussian. The following two propositions are directly adapted from Hainaut and Vrins (2024) (Propositions 4 and 5), and we refer the reader to this article for the proofs.

Proposition 3. *Let us define $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$, the $(m+d)$-vector:*

$$\boldsymbol{\beta}^{(n)}(\boldsymbol{x}) = \begin{pmatrix} k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \boldsymbol{x}\right) \\ k\left(\bar{\boldsymbol{X}}, \boldsymbol{x}\right) \end{pmatrix}, \tag{17}$$

and the Gram matrix

$$C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \;=\; \sigma^{*2}I_{m+d} + \begin{pmatrix} k_{\mathcal{L}^{2}{}^{(n)}}\left(\mathring{\boldsymbol{X}}, \mathring{\boldsymbol{X}}\right) & k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \\ k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{\top} & k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \end{pmatrix}. \tag{18}$$

The approximate solution $\widehat{V^{(n)}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{W}^{(n)}|\boldsymbol{z}^{(n)},\boldsymbol{h}}\left(g^{(n)}(\boldsymbol{x})\right)$ of the HJB Eq. (9) is given by

$$\widehat{V^{(n)}}(\boldsymbol{x}) \;=\; \boldsymbol{\beta}^{(n)}(\boldsymbol{x})^{\top}C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}. \tag{19}$$

As previously mentioned, the Gaussian noises added to h_i's and z_i's introduce a regularization term, that is the diagonal matrix $\sigma^{*2} I_{m+d}$, in the Gram matrix (18). This term stabilizes a possibly ill-conditioned matrix. The standard deviation σ^* may then be viewed as a parameter tuning the numerical robustness of the PI-CGPR. Of course, this affects the accuracy of the solution but we will see in the numerical illustrations that its impact is limited. The next proposition provides the conditional variance of the estimator. For a proof, we again refer the reader to Proposition 5 of Hainaut and Vrins (2024).

Algorithm 1 PI-CGPR : policy iterations algorithm with CGPR

1. Choose an initial admissible control $\boldsymbol{c}^{(0)}$ and set $\widehat{V^{(0)}}(\boldsymbol{x}) = J(\boldsymbol{x}, \boldsymbol{c}^{(0)})$.
2. Sample m points $\mathring{\boldsymbol{x}}_i = (\mathring{t}_i, \mathring{\boldsymbol{y}}_i) \in \mathring{\mathcal{X}}$.
3. Sample d points $\bar{\boldsymbol{x}}_i = (T, \bar{\boldsymbol{y}}_i) \in \bar{\mathcal{X}}$ and set $(h_i = U_2(\bar{\boldsymbol{x}}_i) + \epsilon)_{i=1\dots d}$ where $\epsilon \sim N(0, \sigma^*)$.
4. Loop on $n \in \mathbb{N}$
   a) Update of $\boldsymbol{z}^{(n)} = \left(z_1^{(n)}, ..., z_d^{(n)}\right)^\top$:

$$z_i^{(n)} = -U_1(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) + \epsilon \,,\ \epsilon \sim N(0, \sigma^*)\,.$$

   b) Compute the Gram matrix, and its inverse:

$$C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) = \sigma^{*2} I_{m+d} + \begin{pmatrix} k_{\mathcal{L}^{2\,(n)}}\left(\mathring{\boldsymbol{X}}, \mathring{\boldsymbol{X}}\right) & k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \\ k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^\top & k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \end{pmatrix} \tag{20}$$

   Calculate $\boldsymbol{\beta}^{(n)}\left(\mathring{\boldsymbol{x}}_i\right) = \begin{pmatrix} k_{\mathcal{L}^{(n)}}\left(\mathring{\boldsymbol{X}}, \mathring{\boldsymbol{x}}_i\right) \\ k\left(\bar{\boldsymbol{X}}, \mathring{\boldsymbol{x}}_i\right) \end{pmatrix}$ and intermediate value functions

$$\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}_i) = \boldsymbol{\beta}^{(n)}(\mathring{\boldsymbol{x}}_i)^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1} \begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}. \tag{21}$$

   c) find the improvement of $\boldsymbol{c}^{(n)}$

$$\boldsymbol{c}^{(n+1)} = \sup_{\boldsymbol{c} \in A}\left(U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}) + \mathcal{L}_{\mathring{\boldsymbol{x}}, \boldsymbol{c}} \widehat{V^{(n)}}(\mathring{\boldsymbol{x}})\right) \ \forall \mathring{\boldsymbol{x}} \in \mathring{\boldsymbol{X}}. \tag{22}$$

   d) Stop if $\frac{1}{m}\sum_{i=1}^m \left(\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}_i) - \widehat{V^{(n-1)}}(\mathring{\boldsymbol{x}}_i)\right)^2 \leq \delta$, for a small $\delta \in \mathbb{R}^+$.

Proposition 4. *Let $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ be the $(m+d)$ vector defined in Eq. (17). The conditional variance of $g^{(n)}(\boldsymbol{x})$ is equal to*

$$\mathbb{V}_{\boldsymbol{W}^{(n)}|, \boldsymbol{z}^{(n)}, \boldsymbol{h}}\left(g^{(n)}(\boldsymbol{x})\right) = k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{\beta}^{(n)}(\boldsymbol{x})^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1} \boldsymbol{\beta}^{(n)}(\boldsymbol{x})\,. \tag{23}$$

The conditional variance is a useful metric for assessing the accuracy of predicted value functions. Unlike alternative methods such as backward PDE solvers or Physics-Informed Neural Networks (PINNs), CGPR provides explicit information about the uncertainty associated with its predictions. The full algorithm for solving the HJB equation using policy iterations and constrained Gaussian process regressions (PI-CGPR) is presented in Frame 1 presents the full

algorithm for solving the HJB equation by policy iterations and constrained Gaussian process regressions (PI-CGPR algorithm).

After the completion of the algorithm, we can measure the errors on the inner and boundary domains by computing the average residuals of Equations (9):

$$\mathring{Err} = \frac{1}{m}\sum_{\mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}} \left| \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} \widehat{V^{(n)}}(\mathring{\boldsymbol{x}}_i) + U_1(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) \right| , \tag{24}$$

$$\bar{Err} = \frac{1}{d}\sum_{\bar{\boldsymbol{x}}_i \in \bar{\boldsymbol{X}}} \left| \widehat{V^{(n)}}(\bar{\boldsymbol{x}}_i) - U_2(\bar{\boldsymbol{x}}_i) \right| . \tag{25}$$

4 The PI-CGPR with Gaussian kernels

The computation of estimates requires applying the differential operator $\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(n)}}$ twice to the kernel function. Many alternatives, see e.g. Beckers (2021), exist but we opt for a Gaussian kernel (also called radial basis function) in our numerical illustrations. This kernel is easy to differentiate and allows us to infer fully analytical expressions for the vector $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ and sub-blocks of the Gram matrix $C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)$. Let us denote $\boldsymbol{x} = (t, \boldsymbol{y})$, $\boldsymbol{y} = (y_1, ..., y_{p-1})$, $\tilde{\boldsymbol{x}} = (\tilde{t}, \tilde{\boldsymbol{y}})$ and $\tilde{\boldsymbol{y}} = (\tilde{y}_1, ..., \tilde{y}_{p-1})$. The Gaussian kernel has the general form:

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(\eta_0 - \frac{(t-\tilde{t})^2}{2\,\eta_t^2} - \sum_{k=1}^{p-1} \frac{(y_k - \tilde{y}_k)^2}{2\,\eta_k^2}\right) \quad \forall \boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X} , \tag{26}$$

where $\boldsymbol{\eta} = (\eta_0, \eta_t, \eta_1, ..., \eta_{p-1})^\top$is a $(p+1)$-vector in $\mathbb{R}_0^{p+1}$ of hyper-parameters. The coefficients $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ of the CGPR have, in this case, a closed-form expression. The last d elements of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ are equal to $(k(\bar{\boldsymbol{x}}_i, \boldsymbol{x}))_{i=1,...,d}$. The first m items of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ require the calculation of $\mathcal{L}_{\tilde{\boldsymbol{x}},\boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})$. For this purpose, we introduce the following p-vector:

$$\boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \begin{pmatrix} (y_1 - \tilde{y}_1)/\eta_1^2 \\ \vdots \\ (y_{p-1} - \tilde{y}_{p-1})/\eta_{p-1}^2 \end{pmatrix} , \tag{27}$$

and the $(p-1)\times(p-1)$ matrix:

$$\boldsymbol{H}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \begin{pmatrix} \frac{(y_1-\tilde{y}_1)^2-\eta_1^2}{\eta_1^4} & \frac{(y_1-\tilde{y}_1)(y_2-\tilde{y}_2)}{\eta_1^2\eta_2^2} & \cdots & \frac{(y_1-\tilde{y}_1)(y_{p-1}-\tilde{y}_{p-1})}{\eta_1^2\eta_{p-1}^2} \\ \frac{(y_1-\tilde{y}_1)(y_2-\tilde{y}_2)}{\eta_1^2\eta_2^2} & \ddots & \ddots & \frac{(y_2-\tilde{y}_2)(y_{p-1}-\tilde{y}_{p-1})}{\eta_2^2\eta_{p-1}^2} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{(y_1-\tilde{y}_1)(y_{p-1}-\tilde{y}_{p-1})}{\eta_1^2\eta_{p-1}^2} & \frac{(y_2-\tilde{y}_2)(y_{p-1}-\tilde{y}_{p-1})}{\eta_2^2\eta_{p-1}^2} & \cdots & \frac{(y_{p-1}-\tilde{y}_{p-1})^2-\eta_{p-1}^2}{\eta_{p-1}^4} \end{pmatrix} . \tag{28}$$

With a Gaussian kernel, the vector $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ involved in Algorithm 1, admits a closed-form expression as stated in the next proposition.

Proposition 5. *For all $\boldsymbol{x} = (t, \boldsymbol{y}) \in \mathcal{X}$, the approximated value function after n iterations of algorithm 1 is $\widehat{V^{(n)}}(\boldsymbol{x}) = \boldsymbol{\beta}^{(n)}(\boldsymbol{x})^\top \times C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1} (\boldsymbol{z}^{(n)}, \boldsymbol{h})^\top$. With a Gaussian kernel (26), the $(m+d)$-vector $\boldsymbol{\beta}^{(n)}(\boldsymbol{x}) = \left(\boldsymbol{\beta}_1^{(n)}(\boldsymbol{x}), \boldsymbol{\beta}_2^{(n)}(\boldsymbol{x})\right)^\top$ has the following components:*

$$\begin{cases} \boldsymbol{\beta}_1^{(n)}(\boldsymbol{x}) = \Big(\big((t - \mathring{t}_i)/\eta_t^2 - \alpha + \boldsymbol{\mu}_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i) \\ \quad + \frac{1}{2} tr\left(\Sigma_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{H}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)\right)\big)\, k(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)\Big)_{i=1,...,m} , \\ \boldsymbol{\beta}_2^{(n)}(\boldsymbol{x}) = \left(e^{\eta_0 - (\bar{t}_i - t)^2/(2\,\eta_t^2) - \sum_{k=1}^{p-1} (\bar{y}_{i,k} - t)^2/(2\,\eta_k^2)} \right)_{i=1,...,d} . \end{cases}$$

The sketch of the proof is provided in Appendix C. The construction of the Gram matrix, $C^{(n)}\left(\mathring{\bar{\boldsymbol{X}}}, \bar{\boldsymbol{X}}\right)$, requires to compute the sub-blocks: $k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)$, $k_{\mathcal{L}^{(n)}}\left(\mathring{\bar{\boldsymbol{X}}}, \bar{\boldsymbol{X}}\right)$ and $k_{\mathcal{L}^{2(n)}}\left(\mathring{\bar{\boldsymbol{X}}}, \mathring{\bar{\boldsymbol{X}}}\right)$. When the kernel is Gaussian, they admit general closed-form expressions.

Proposition 6. *The sub-blocks $k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)$ and $k_{\mathcal{L}^{(n)}}\left(\mathring{\bar{\boldsymbol{X}}}, \bar{\boldsymbol{X}}\right)$ are computed with the following analytical formulas*

$$\left\{\begin{array}{l} k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) = \left(e^{\eta_0 - (\bar{t}_i - \bar{t}_j)^2/\left(2\,\eta_t^2\right) - \sum_{k=1}^{p-1}\left(\bar{y}_{i,k} - \bar{y}_{j,k}\right)^2/\left(2\,\eta_k^2\right)}\right)_{i,j=1,\ldots,d}, \\ k_{\mathcal{L}^{(n)}}\left(\mathring{\bar{\boldsymbol{X}}}, \bar{\boldsymbol{X}}\right) = \left(\left(\left(\bar{t}_j - \mathring{t}_i\right)/\eta_t^2 - \alpha + \boldsymbol{\mu}_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\bar{\boldsymbol{x}}_j, \mathring{\boldsymbol{x}}_i)\right.\right. \\ \left.\left. + \frac{1}{2} tr\left(\Sigma_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{H}_{\mathring{\boldsymbol{y}}_i}(\bar{\boldsymbol{x}}_j, \mathring{\boldsymbol{x}}_i)\right)\right)\, k(\bar{\boldsymbol{x}}_j, \mathring{\boldsymbol{x}}_i)\right)_{i=1,\ldots,m\, j=1,\ldots,d}. \end{array}\right.$$

The expression of $k\left(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)$ is a direct consequence of the kernel choice. The second equation comes from the symmetry of the kernel and Equation (64) in Appendix C. Let us denote by $\boldsymbol{e}_k$'s the canonical vectors of $\mathbb{R}^{p-1}$. We define the following matrix:

$$\boldsymbol{H}_{\boldsymbol{y}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \boldsymbol{H}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) + \begin{pmatrix} 2/\eta_1^2 & 0 & 0 \\ 0 & \ldots & 0 \\ 0 & 0 & 2/\eta_{p-1}^2 \end{pmatrix}.$$

Proposition 7. *The sub-block $k_{\mathcal{L}^{2(n)}}\left(\mathring{\bar{\boldsymbol{X}}}, \mathring{\bar{\boldsymbol{X}}}\right)$ of $C^{(n)}\left(\mathring{\bar{\boldsymbol{X}}}, \bar{\boldsymbol{X}}\right)$ is the matrix of $\mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} \mathcal{L}_{\mathring{\boldsymbol{x}}_j, \boldsymbol{c}^{(n)}} k(\mathring{\boldsymbol{x}}_i, \mathring{\boldsymbol{x}}_j)$ for $i, j = 1, ..., m$ where*

$$\begin{aligned} \mathcal{L}_{\boldsymbol{x}, \boldsymbol{c}^{(n)}} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) &= \partial_t \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \\ &- \alpha \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) + \boldsymbol{\mu}_{\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{c}^{(n)})^\top \nabla_{\boldsymbol{y}} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \\ &+ \frac{1}{2} tr\left(\Sigma_{\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{c}^{(n)})^\top \mathcal{H}_{\boldsymbol{y}} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right). \end{aligned} \tag{29}$$

Using Equation (64), the partial derivative in (29) is given by

$$\partial_t \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \frac{1}{\eta_t^2} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) - \left(\mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right) \frac{(t - \tilde{t})}{\eta_t^2}. \tag{30}$$

while the gradient $\nabla_{\boldsymbol{y}} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \left(\partial_{y_k} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right)_{k=1,\ldots,p-1}$ is a $(p-1)$ vector filled by :

$$\begin{aligned} \partial_{y_k} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right) &= \left[\boldsymbol{e}_k^\top \Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)})^\top \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right. \\ &\left. + \boldsymbol{e}_k^\top \boldsymbol{\mu}_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)})\right] \frac{k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)}{\eta_k^2} - \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)\, \boldsymbol{e}_k^\top \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}). \end{aligned} \tag{31}$$

Finally The Hessian $\mathcal{H}_{\boldsymbol{y}} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \left(\partial_{y_l} \partial_{y_k} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right)_{k,l=1,\ldots,p-1}$ in Equation (29) is a $(p-1) \times (p-1)$ matrix with elements:

$$\begin{aligned} \partial_{y_l} \partial_{y_k} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right) &= \left(\boldsymbol{e}_k^\top \Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)})^\top \boldsymbol{e}_l\right) \frac{k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)}{\eta_k^2 \eta_l^2} \\ &- \left(\partial_{y_k} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)\right) \boldsymbol{e}_l^\top \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) - \left(\partial_{y_l} \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)\right) \boldsymbol{e}_k^\top \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \\ &- \mathcal{L}_{\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right) \left(\boldsymbol{e}_l^\top \left(\boldsymbol{H}_{\boldsymbol{y}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right) \boldsymbol{e}_k\right). \end{aligned} \tag{32}$$

In the numerical illustrations, we consider optimization problems for which the optimal intermediate controls, solving Eq. (8), admit an explicit formulation in terms of the intermediate approximated payoff and its first and second order derivatives, i.e.

$$\boldsymbol{c}^{(n)} = \boldsymbol{c}^{(n)}\left(\boldsymbol{x}, \partial_t \widehat{V^{(n-1)}}(\boldsymbol{x}), \nabla_{\boldsymbol{y}} \widehat{V^{(n-1)}}(\boldsymbol{x}), \mathcal{H}_{\boldsymbol{y}} \widehat{V^{(n-1)}}(\boldsymbol{x})\right).$$

The gradient and Hessian of $\widehat{V^{(n-1)}}(\boldsymbol{x})$ also admits analytical expressions inferred from Eq. (31) and (32). We provide them in Appendix A.

To conclude this section, we remark that Eq. (19) and (23) correspond to the conditional expectation and variance of $g^{(n)}(\boldsymbol{x}) \mid \boldsymbol{z}^{(n)}, \boldsymbol{h}$. All variables being Gaussian, the triplet $\left(g^{(n)}(\boldsymbol{x}), \boldsymbol{Z}^{(n)}, \boldsymbol{H}\right)$ where n is the last iteration of Algorithm 1, has a multivariate normal distribution:

$$\begin{bmatrix} g^{(n)}(\boldsymbol{x}) \\ \boldsymbol{Z}^{(n)} \\ \boldsymbol{H} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x},\boldsymbol{x}) & \boldsymbol{\beta}^{(n)}(\boldsymbol{x})^\top \\ \boldsymbol{\beta}^{(n)}(\boldsymbol{x}) & C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right) \end{bmatrix}\right).$$

In theory, this observation would allows us to optimize hyper-parameters $\boldsymbol{\eta}$ by maximizing the marginal log-likelihood of $(\boldsymbol{Z}^{(n)}, \boldsymbol{H})$:

$$\begin{aligned} \log L(\boldsymbol{z}^{(n)}, \boldsymbol{h}) = & \frac{(m+d)}{2}\log 2\pi - \frac{1}{2}\log\left|C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)\right| \\ & - \frac{1}{2}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}. \end{aligned} \tag{33}$$

In practice, this maximization is time-consuming and prone to numerical instabilities. First, computing the log-determinant of a large Gram matrix is computationally demanding. Second, during the search for optimal hyperparameters, inaccuracies in determinant calculations can lead to bandwidth values that render the Gram matrix nearly ill-conditioned. For these reasons, we consider an alternative efficient method in which the hyper-parameters minimize the errors of approximation on the inner and boundary sample sets. We denote by $V^{(n^*,\boldsymbol{\eta})}(.)$ the optimal value function obtained after n^* iterations of Algorithm 1. The inner and terminal total errors are defined as follows:

$$\begin{aligned} \mathring{TErr}(\boldsymbol{\eta}) &= \sum_{\mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}} \left|\mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}}\widehat{V^{(n^*,\boldsymbol{\eta})}}\left(\mathring{\boldsymbol{x}}_i\right) - z_i^{(n^*)}\right|, \\ \bar{TErr}(\boldsymbol{\eta}) &= \sum_{\bar{\boldsymbol{x}}_i \in \bar{\boldsymbol{X}}} \left|\widehat{V^{(n^*)}}\left(\bar{\boldsymbol{x}}_i\right) - h_i\right|. \end{aligned}$$

The total error is the sum of inner and terminal errors:

$$TErr(\boldsymbol{\eta}) = \mathring{TErr}(\boldsymbol{\eta}) + \bar{TErr}(\boldsymbol{\eta}). \tag{34}$$

We run a standard gradient descent procedure and update hyper-parameters $\boldsymbol{\eta}^{(k)}$ at each iteration:

$$\boldsymbol{\eta}^{(k)} = \boldsymbol{\eta}^{(k-1)} - \rho\, \nabla_{\boldsymbol{\eta}^{(k-1)}} TErr\left(\boldsymbol{\eta}^{(k-1)}\right), \tag{35}$$

where ρ is a learning rate. The gradient in this last expression does not admit any simple analytical expression and is, for this reason, numerically computed. We initialize the gradient descent with $\eta_0^{(0)} = 0$, and $\eta_t^{(0)}$, $\left(\eta_j^{(0)}\right)_{j=1,\dots,p-1}$ are respectively set to the standard deviations of $\left\{\mathring{t}_i\right\}_{i=1,\dots,m}$ and $\{\mathring{\boldsymbol{y}}_i\}_{i=1,\dots,m}$. A variant to accelerate the optimization process is to compute, at each iteration, only the partial derivative with respect to a single bandwidth, rather than the full gradient. As with the log-likelihood approach, the total error is not a convex function of the hyperparameters, and the algorithm may converge to a local minimum depending on the initial values. In our numerical illustrations, we initialize the bandwidths using a multiple of the standard deviations of the state variables in the inner training set, which yields good results. An alternative strategy is to run the estimation procedure multiple times with different initial values, as implemented in the Python library Scikit-Learn for Gaussian Process Regression.

5 Numerical illustration

We test the capacity of the PI-CGPR algorithm to solve the linear-quadratic regulator and the consumption-investment problems. As they admit closed-form solutions for the value function and the controls, we can accurately measure the errors of approximation induced by the PI-CGPR algorithm. We also compare the PI-CGPR to the PDE solving method and to a Physics-Informed Neural Network (PINN).

5.1 Optimal investment-consumption problem

In this section, we focus on the optimal consumption-allocation problem. An investor has an initial wealth Y_0, invested in a portfolio of cash, earning the risk-free rate $r \in \mathbb{R}^+$, and v stocks, $\boldsymbol{S}_t = \left(S_t^{(1)}, ..., S_t^{(v)}\right)$. The stock prices are ruled by a geometric Brownian motion:

$$\frac{d\boldsymbol{S}_t}{\boldsymbol{S}_t} = \boldsymbol{\mu}\, dt + \Sigma\, d\boldsymbol{B}_t\,,$$

where $\boldsymbol{\mu} = (\mu_1, ..., \mu_v) \in \mathbb{R}^v$ and $\Sigma \in \mathbb{R}^{v\times v}$ and $\frac{d\boldsymbol{S}_t}{\boldsymbol{S}_t}$ is an element-wise division. At time t, the fractions of the fund invested in stocks are stored in a vector denoted $\boldsymbol{\pi}_t = \left(\pi_t^{(1)}, \ldots, \pi_t^{(v)}\right)^\top$. The investor also withdraws a percentage b_t of Y_t for its own consumption. The wealth process is therefore ruled by the following SDE:

$$\begin{aligned} dY_t &= Y_t \boldsymbol{\pi}_t^\top \frac{d\boldsymbol{S}_t}{\boldsymbol{S}_t} + Y_t\Big(1 - \boldsymbol{\pi}_t^\top \mathbf{1}_v\Big) r\, dt - b_t Y_t\, dt \\ &= \Big(r + (\boldsymbol{\mu} - r)^\top \boldsymbol{\pi}_t - b_t\Big)\, Y_t dt + Y_t \boldsymbol{\pi}_t^\top \Sigma\, d\boldsymbol{B}_t\,, \end{aligned} \tag{36}$$

where $\mathbf{1}_v$ is the unit vector of dimension v. In this problem, we have a single risk process[3] $(Y_t)_{t\geq 0}$ whereas the control is a $(v+1)$ vector $\boldsymbol{c}_t = (b_t, \boldsymbol{\pi}_t)^\top$ ($p = 2$ and $l = v+1$). In this setting, the drift and volatility matrix of risk processes are:

$$\boldsymbol{\mu_Y}(t, Y_t, \boldsymbol{c}_t) = \Big(r + (\boldsymbol{\mu} - r)^\top \boldsymbol{\pi}_t - b_t\Big)\, Y_t\ ,\ \Sigma_{\boldsymbol{Y}}(t, Y_t, \boldsymbol{c}_t) = Y_t \boldsymbol{\pi}_t^\top \Sigma\,.$$

The running utility of consumption and the utility of the terminal wealth, are concave and invertible functions denoted by $U_1(.)$ and $U_2(.)$. We choose utilities with a constant relative risk aversion (CRRA) coefficient γ:

$$U_1\left(b_t y\right) = \frac{(b_t y)^\gamma}{\gamma}\ ,\ U_2\left(y\right) = \frac{(y)^\gamma}{\gamma}\,.$$

The payoff is the sum of the discounted expected utility of consumption and terminal wealth over $[0, T]$:

$$J(t, y, \boldsymbol{c}) = \mathbb{E}\left(\int_t^T e^{-\alpha(s-t)} \frac{(b_s Y_s)^\gamma}{\gamma} ds + e^{-\alpha(T-t)} \frac{(Y_T)^\gamma}{\gamma} \,|\, Y_t = y\right)\,.$$

The linear differential operator is in this case defined by

$$\begin{aligned} \mathcal{L}_{\boldsymbol{x},\boldsymbol{c}}\cdot = \partial_t \cdot - \alpha \cdot + \Big(r + (\boldsymbol{\mu} - r)^\top \boldsymbol{\pi}_t - b_t\Big)\, y\, \nabla_{\boldsymbol{y}}\cdot \\ + \frac{1}{2} y^2 \mathrm{tr}\left(\boldsymbol{\pi}_t^\top \Sigma\, \Sigma^\top \boldsymbol{\pi}_t\, \mathcal{H}_{\boldsymbol{y}}\cdot\right)\,. \end{aligned} \tag{37}$$

[3] As $(Y_t)_{t\geq 0}$ is a scalar process, we do not use bold notation.

The inner and terminal domains are $\mathring{\mathcal{X}} = [0,T) \times \mathbb{R}^+$ and $\bar{\mathcal{X}} = T \times \mathbb{R}^+$. Noting $\boldsymbol{x} = (t,y)$, the value function, $V(\boldsymbol{x}) = \max_{c \in \mathcal{A}} J(\boldsymbol{x}, \boldsymbol{c})$, is the solution of the following HJB equation:

$$\sup_{\boldsymbol{c} \in A} \left(\frac{(b\mathring{y})^\gamma}{\gamma} + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}} V(\mathring{\boldsymbol{x}}) \right) = 0 \quad \forall \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}\,, \tag{38}$$

$$V(\bar{\boldsymbol{x}}) = \frac{(\bar{y})^\gamma}{\gamma} \quad \forall \bar{\boldsymbol{x}} \in \bar{\mathcal{X}}\,. \tag{39}$$

The optimal controls, $\boldsymbol{c}^*(t,y) = (b^*(t,y), \boldsymbol{\pi}^*(t,y))$, are obtained by canceling the derivative of Eq. (38) w.r.t. to consumption rate and asset proportions:

$$\begin{cases} b^*(t,y) & = \frac{1}{y} \left(\partial_y V(\boldsymbol{x})\right)^{\frac{1}{\gamma-1}}\,, \\ \boldsymbol{\pi}^*(t,y) & = -\left(\Sigma\Sigma^\top\right)^{-1} (\boldsymbol{\mu} - r) \frac{1}{y} \frac{\partial_y V(\boldsymbol{x})}{\partial_{yy} V(\boldsymbol{x})}\,. \end{cases}$$

The next proposition provides the analytical formula of the value function and controls.

Proposition 8. *$V(t,y) = \max_{c \in \mathcal{A}} J(t,y,\boldsymbol{c})$ is a power function of the wealth:*

$$V(t,y) \;=\; A(t) \frac{y^\gamma}{\gamma}\,, \tag{40}$$

where $A(t) : [0,T] \to \mathbb{R}^+$ is given by

$$A(t) \;=\; \left(\frac{1-\gamma}{\theta} \left(e^{\frac{\theta}{1-\gamma}(T-t)} - 1 \right) + e^{\frac{\theta}{1-\gamma}(T-t)} \right)^{1-\gamma}\,. \tag{41}$$

The optimal consumption and investment policy are respectively equal to:

$$\begin{cases} b_t^* = & A(t)^{\frac{1}{\gamma-1}}\,, \\ \boldsymbol{\pi}_t^* = & \left(\Sigma\Sigma^\top\right)^{-1} \frac{(\boldsymbol{\mu}-r)}{(1-\gamma)}\,. \end{cases} \tag{42}$$

We can check that the value function (40) satisfies the HJB equation, with the optimal control (42).

The analytical solution of Proposition 8 will serve as a benchmark for the PI-CGPR algorithm 1. For the consumption-investment problem, the step 4.b) of the algorithm becomes:

$$\begin{cases} b^{(n)}(t,y) & = \frac{1}{y} \left(\partial_y \widehat{V^{(n-1)}}(t,y) \right)^{\frac{1}{\gamma-1}}\,, \\ \boldsymbol{\pi}^{(n)}(t,y) & = -\left(\Sigma\Sigma^\top\right)^{-1} (\boldsymbol{\mu} - r) \frac{1}{y} \frac{\partial_y \widehat{V^{(n-1)}}(t,y)}{\partial_{yy} \widehat{V^{(n-1)}}(t,y)}\,, \end{cases} \tag{43}$$

where $\widehat{V^{(n-1)}}(\boldsymbol{x})$ is the CGPR approximation. In numerical tests, we consider a market with two stocks. Table 1 provides the parameters defining the asset dynamics, the risk-free rate and the utility functions. Table 2 reports the parameters of the kernel. σ^* is set to 10^{-4}. This value offers a good trade-off between accuracy and robustness. In practice, σ^* should be set to the lowest possible value for which the Gram matrix $C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)$ is not ill-conditioned. Considering very large sample sets is often the origin of numerical errors when inverting it. In this case, increasing σ^* gives more weight to the regularization term in the Gram matrix and prevents numerical instabilities during the inversion. The initial bandwidths $\eta_t^{(0)}$ and $\eta_1^{(0)}$ are set to a multiple of the standard deviation of state variables in the inner sample sets. This choice reveals in practice accelerates the convergence when estimating optimal bandwidths.

α	4.00%	γ	0.30
r	3.00%	μ_1	5.00%
μ_2	7.00%	T	1,3,5 years
$\Sigma = \begin{pmatrix} 0.20 & -0.05 \\ 0 & 0.20 \end{pmatrix}$			

Table 1: Parameters of the investment-consumption problem

$\mathring{t}$	uniform distribution $[0, T)$
$\mathring{y}$	uniform distribution $[0, 500)$
σ^*	10^{-4}
$\eta_0^{(0)}$	0
$\eta_t^{(0)}$	0.75×standard deviation of $\mathring{t}$, $\sqrt{T^2/12}$
$\eta_1^{(0)}$	0.75×standard deviation of $\mathring{y}$, $\sqrt{500^2/12}$
$c^{(0)}(t, y)$	10%
$\pi^{(1)(0)}(t, y)$	10%
$\pi^{(2)(0)}(t, y)$	10%

Table 2: Parameters of the PI-CGPR for the investment-consumption problem

Table 3 reports the mean relative errors (MREs) between approximated and exact value functions for sample sets $\mathring{\boldsymbol{X}}$ of various sizes m. The MRE is computed as:

$$\text{MRE} \quad = \quad \frac{1}{m} \sum_{\mathring{\boldsymbol{x}}_i \in \mathring{\boldsymbol{X}}} \left| \frac{\widehat{V^{(n)}(\mathring{\boldsymbol{x}}_i)} - V(\mathring{\boldsymbol{x}}_i)}{V(\mathring{\boldsymbol{x}}_i)} \right| . \tag{44}$$

The PI-CGPR algorithm is stopped when the average quadratic variation of $\widehat{V^{(n)}} - \widehat{V^{(n-1)}}$, defined in step d) of Algorithm 1, falls below δ=1%. Whatever the configuration of the problem, the MRE is smaller than 0.60%, which is an acceptable level for a numerical approximation. For a fixed sample size m, the MRE increases with the time horizon, T. Increasing the sample size does not have a significant impact on the MREs. Table 3 reveals that convergence is attained after at most three iterations.

	m and $d = m/2$					
	200	300	500	200	300	500
T	MRE (%)			# of iterations		
1.0	0.23	0.17	0.24	1.0	2.0	2.0
3.0	0.39	0.33	0.40	3.0	2.0	2.0
5.0	0.50	0.50	0.60	5.0	3.0	3.0

Table 3: Mean relative errors (MREs) in % and the number of iterations to attain convergence

The computational times[4], reported in Table 4, vary from 12.23 to 60.65 seconds. The time-consuming step is the optimization of hyper-parameters, which requires several runs of the policy iterations algorithm 1. We perform maximum 30 iterations of the gradient descent, Equation (35), with a learning rate $\rho = 0.01$ and stop when the variation of the total error, $TErr(\boldsymbol{\eta})$, falls below 0.001. As seen in Table 3, this is sufficient to achieve good overall accuracy. As $\boldsymbol{\eta}$ can be assimilated to standard deviations, the starting values in Table 2 are proportional to the

[4]Configuration: laptop with an Intel Core I7 and 32Mb of RAM.

standard deviations of $\mathring{t}$ and $\mathring{y}$. The gradient with respect to $\boldsymbol{\eta}$ in Equation (35) is computed by finite differences, with increments of $\boldsymbol{\eta}$ set to $\max(0.01\boldsymbol{\eta}\,;\,0.01)$.

	m and $d = m/2$		
	200	300	500
T	Time (seconds)		
1.0	51.83	29.25	22.79
3.0	54.24	12.23	67.20
5.0	60.55	14.43	25.32

Table 4: Computational time, optimal consumption-investment problem

Table 5 reports the average errors $\mathring{Err}$ and $\bar{Err}$ on the inner and boundary domains, as defined in Equations (24) and (25). For problems without any analytical solution, these are the only available indicators for measuring the accuracy of the PI-CGPR. Contrary to MREs, we do not observe any clear trend between T and m but the average errors are very small, regardless of the setting of the PI-CGPR.

	m and $d = m/2$					
	200	300	500	200	300	500
T	$\mathring{Err}$			$\bar{Err}$		
1.0	0.0701	0.0270	0.0313	0.0102	0.0030	0.0116
3.0	0.2521	0.0063	0.0153	0.0120	0.0032	0.0106
5.0	0.1011	0.0026	0.0137	0.0141	0.0027	0.0097

Table 5: Errors of approximation on the inner and boundary domains

To gain better insight into the PI-CGPR's capacity to provide an accurate approximation, we focus on the consumption-investment problem with $T = 5$ and m=500. Table 6 presents the evolution of inner and boundary errors, (24) and (25), for each iteration of the policy iterations algorithm. It confirms the convergence of the procedure and the reduction of errors after the update of controls.

	$\mathring{Err}$	$\bar{Err}$	$\frac{1}{m}\sum_{i=1}^{m}\left(\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}_i) - \widehat{V^{(n-1)}}(\mathring{\boldsymbol{x}}_i)\right)^2$
1	1.9083	0.0117	1275.53
2	0.0747	0.0103	25.70
3	0.0060	0.0099	0.03
4	0.0041	0.0097	0.00

Table 6: Evolution of the average inner and boundary errors by PI iterations

Figure 1 displays the absolute spreads between the exact and approximated value functions, $\left|\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}) - V(\mathring{\boldsymbol{x}})\right|$, on the domain $[0,5]\times[50,200]$. On this subset, errors are of the same order and small nearly everywhere. We observe a few small local maxima located in areas not containing any sampled points in $\mathring{\boldsymbol{X}}$. We also observe an increase in errors in the upper right part of the plot. This is due to the propagation of errors near the boundaries.

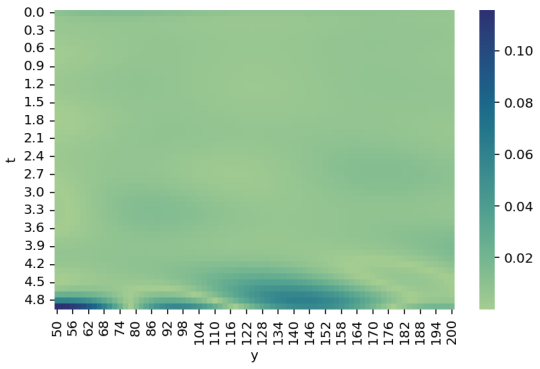

Figure 1: Inner errors for $T = 5$ years and m =500

Figure 2 compares approximated and exact value functions and controls in a single scenario where the wealth remains equal to 100, still for $T = 5$ and $m = 500$. We observe a clear similarity between results obtained with the PI-CGPR algorithm and their analytical counterparts.

To detect where the PI-CGPR is less efficient, we plot heatmaps of relative absolute errors for $V(.)$ and optimal controls, b_t^*, $\pi_t^{(1)*}$, $\pi_t^{(2)*}$ in Figure 3 and 4. We display heatmaps on the domain $[0, 5] \times [50, 200]$. These graphs confirm the accuracy of the PI-CGPR, on this subset of the training domain. The relative spread between exact and approximated value functions does not exceed 0.08%. Approximated and exact controls are also close in most configurations. The largest gaps are observed near the starting date and for the highest values of y. Other tests reveal that the accuracy of approximation declines for pairs (t, y) close to the boundaries of the training domain.

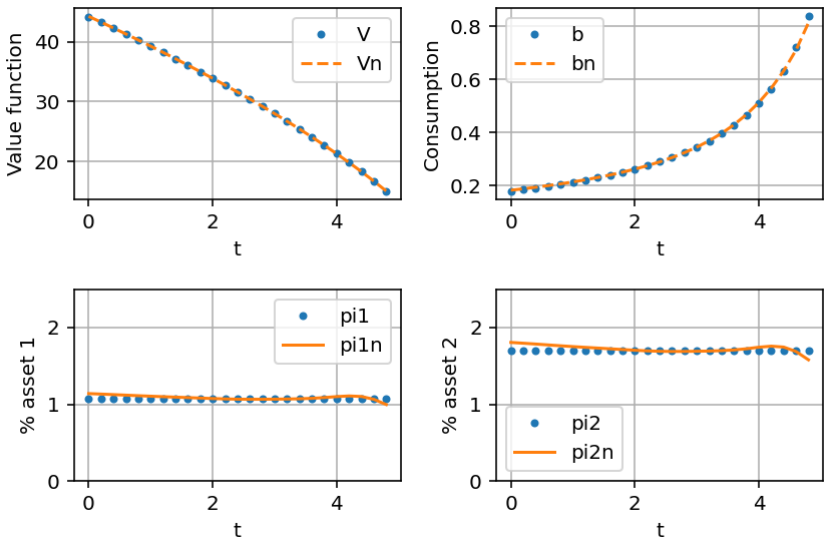

Figure 2: Comparison of exact and approximated value functions, consumption and optimal investments in one sample path: $y_t = 100$ and $T = 5$ years

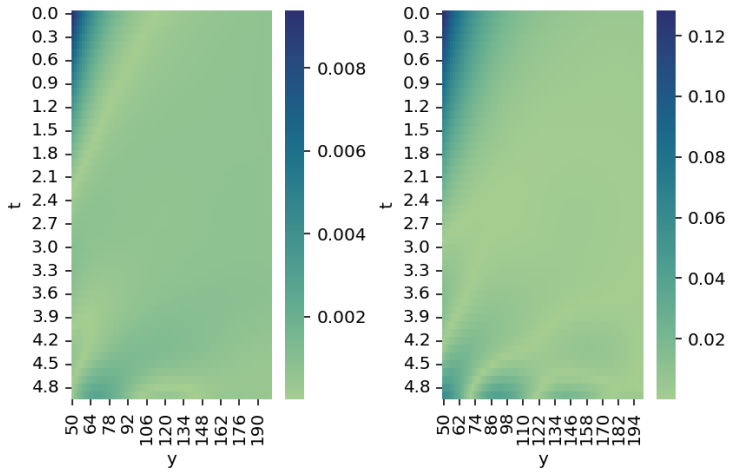

Figure 3: Left plot: heatmap of $\left|\frac{\widehat{V^{(n)}}(t,y)-V(t,y)}{V(t,y)}\right|$. Right plot: heatmap of $\left|\frac{b^{(n)}(t,y)-b^*(t,y)}{b^*(t,y)}\right|$

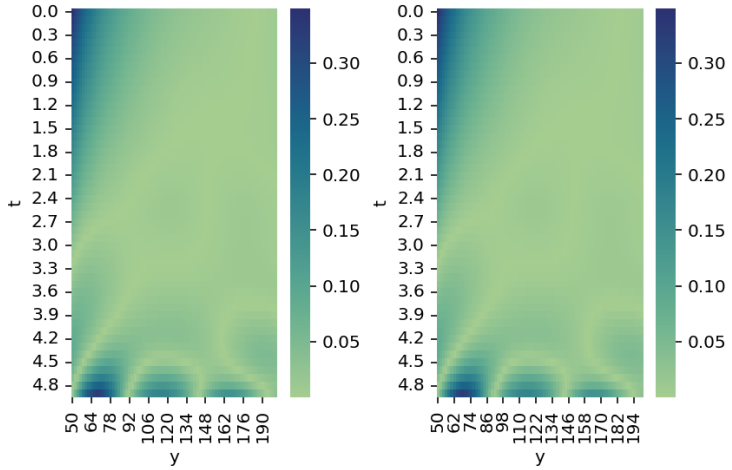

Figure 4: Left plot: heatmap of $\left|\frac{\pi_t^{(1)(n)}-\pi_t^{(1)*}}{\pi_t^{(1)*}}\right|$. Right plot: heatmap of $\left|\frac{\pi_t^{(2)(n)}-\pi_t^{(2)*}}{\pi_t^{(2)*}}\right|$

Table 7 quantifies the impact of the shrinkage parameter σ^* on modelling errors when $T = 5$ and $m = 500$. As expected, the mean relative errors (MREs) decrease as σ^* is reduced. The average error on the boundary domain follows the same trend. However, for very small values of σ^*, the inversion of the Gram matrix may fail during the bandwidth optimization process due to numerical instability.

σ^*	MRE (%)	$\mathring{Err}$	$\bar{Err}$
0.01	1.34	0.3040	0.0287
0.005	1.14	0.0075	0.0163
0.001	0.83	0.0055	0.0114
0.0005	0.73	0.0042	0.0101
0.0001	0.60	0.0137	0.0097
5e-05	0.54	0.0079	0.0064

Table 7: Impact of σ^* on errors

Physics-Informed Neural Networks (PINNs) offer an interesting alternative to the PI-CGPR framework for solving the HJB equation. This neural network-based approach is summarized in Appendix B. The results presented here were obtained using a feed-forward network architecture consisting of one rescaling layer, two hidden layers with 64 neurons each, and activation functions including hyperbolic tangent (TANH) and Gaussian Error Linear Unit (GELU). To ensure the positivity of the value function, the output layer uses the Rectified Linear Unit (RELU) activation function. This architecture was selected for its strong performance and relative simplicity. The network was trained over four phases of 500, 1000, 1000, and 1000 epochs, with respective learning rates of 0.05, 0.01, 0.005, and 0.001. As with the PI-CGPR, the accuracy of the PINN depends on the size of the inner and boundary sample sets. To ensure a fair comparison, the neural network was trained on the same sample sets as those used for the PI-CGPR.

Table 8 reports the PINN mean relative errors (MREs) on the inner domain $\mathring{\boldsymbol{X}}$, expressed in percentages. A comparison with Tables 3 and 4 highlights that the PI-CGPR method outperforms PINNs, yielding lower relative errors and significantly shorter computational times. Moreover, the PINN model is heavily over-parameterized compared to the PI-CGPR: it involves 4,417 neural weights, whereas PI-CGPR requires only three bandwidth parameters. To achieve comparable accuracy, PINNs require substantially larger training sets. For example, with 1,000 inner points and 500 boundary points, the PINN MRE for $T = 5$ drops to 1.01%, compared to 2.98% with smaller datasets. In contrast, PI-CGPR leverages strong prior assumptions through Bayesian inference, enabling it to generalize effectively from limited data—while PINNs must learn structural relationships from scratch.

	200	300	500	200	300	500
T	MRE (%)			Time		
1	2.49	1.02	0.48	167.43	177.99	209.41
3	9.75	5.92	0.92	169.16	183.30	227.85
5	6.35	4.68	2.98	165.04	182.75	238.55

Table 8: PINN : mean relative errors and computational time

We also compare the PI-CGPR method to a PDE-based approach for solving the HJB equation. In this case, the equation is solved via backward iterations using an Euler discretization of partial derivatives on a grid. The state variable ranges from $y_{min} = 0$ to $y_{max} = 200$ with unit steps. The convergence of the method is highly sensitive to the time step size. We consider 500, 1000 and 1500 time steps per year. Table 9 reports the mean relative errors computed on a random sample of 1000 points in the inner domain. Given the simplicity of the problem, the PDE method proves to be more accurate and significantly faster than PINNs. However, the PI-CGPR still achieves excellent and in some cases even superior, accuracy compared to the explicit PDE method.

time steps per year	500	1000	1500	500	1000	1500
T	MRE (%)			Time		
1	0.66	0.27	0.27	0.0684	0.1878	0.4360
3	0.61	0.39	0.49	0.0760	0.1627	0.4139
5	1.64	0.78	1.87	0.0714	0.1974	0.4292

Table 9: PDE : mean relative errors and computational time

5.2 The linear-quadratic regulator

In this section, we test the capacity of the PI-CGPR algorithm to optimize the linear regulation of a system with quadratic reward and penalty utilities. We consider a $(p-1)$ vector of controlled risk processes, $\boldsymbol{Y}_t = \left(Y_t^{(1)}, \ldots, Y_t^{(p-1)}\right)$ ruled by the SDEs:

$$d\boldsymbol{Y}_t \quad = \quad (M\boldsymbol{Y}_t + \boldsymbol{c}_t)\, dt + \Sigma\, d\boldsymbol{B}_t\,,$$

where $M : \mathbb{R}^{(p-1)\times(p-1)}$ and $\Sigma \in \mathbb{R}^{(p-1)\times(p-1)}$. The control, $\boldsymbol{c}_t$, and the vector of Brownian motions, $\boldsymbol{B}_t$, are of dimension $(p-1)$. In this setting, the drift and volatility matrix of risk processes are:

$$\boldsymbol{\mu}_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t, \boldsymbol{c}_t) = M\boldsymbol{Y}_t + \boldsymbol{c}_t\,,\ \Sigma_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t, \boldsymbol{c}_t) = \Sigma\,.$$

Over a time horizon T, we aim to minimize the running costs of controls and the terminal utility. Therefore, we consider negative running and terminal utilities which have quadratic forms

$$\begin{aligned} U_1(t, \boldsymbol{y}, \boldsymbol{c}) &= -\boldsymbol{c}^\top Q_1\, \boldsymbol{c}\,, \\ U_2(t, \boldsymbol{y}) &= -\boldsymbol{y}^\top Q_2\, \boldsymbol{y}\,, \end{aligned}$$

for some positive invertible, positive definite matrix $Q_1, Q_2 \in \mathbb{R}^{(p-1)\times(p-1)}$. The payoff is the sum of the discounted expected utilities of controls and terminal values of risk processes:

$$J(t, \boldsymbol{y}, \boldsymbol{c}) \quad = \quad \mathbb{E}\left(-\int_t^T e^{-\alpha(s-t)} \boldsymbol{c}_s^\top\, Q_1\, \boldsymbol{c}_s\, ds - e^{-\alpha(T-t)} \boldsymbol{Y}_T^\top\, Q_2\, \boldsymbol{Y}_T \,|\, \boldsymbol{Y}_t = \boldsymbol{y} \right). \tag{45}$$

The linear differential operator is in this case defined by

$$\mathcal{L}_{\boldsymbol{x},\boldsymbol{c}} \cdot = \partial_t \cdot - \alpha \cdot + (M\boldsymbol{y} + \boldsymbol{c})^\top \nabla_{\boldsymbol{y}} \cdot + \frac{1}{2} \mathrm{tr}\left(\Sigma\, \Sigma^\top\, \mathcal{H}_{\boldsymbol{y}} \cdot \right). \tag{46}$$

The inner and terminal domains are respectively $\mathring{\mathcal{X}} = [0, T) \times \mathbb{R}^{(p-1)}$ and $\bar{\mathcal{X}} = T \times \mathbb{R}^{(p-1)}$. Noting $\boldsymbol{x} = (t, \boldsymbol{y})$, the value function, $V(\boldsymbol{x}) = \max_{c\in\mathcal{A}} J(\boldsymbol{x}, \boldsymbol{c})$, is the solution of the following HJB equation:

$$\sup_{\boldsymbol{c}\in A} \left(-\boldsymbol{c}^\top\, Q_1\, \boldsymbol{c} + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}} V(\mathring{\boldsymbol{x}}) \right) = 0 \quad \forall \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}\,, \tag{47}$$

$$V(\bar{\boldsymbol{x}}) = -\bar{\boldsymbol{y}}^\top\, Q_2\, \bar{\boldsymbol{y}} \quad \forall \bar{\boldsymbol{x}} \in \bar{\mathcal{X}}\,. \tag{48}$$

By deriving Eq. (47) w.r.t. $\boldsymbol{c}$, we infer that the optimal control, $\boldsymbol{c}^*(t, \boldsymbol{y})$, is proportional to the gradient of the value function:

$$\boldsymbol{c}^*(t, \boldsymbol{y}) \quad = \quad \frac{1}{2} Q_1^{-1} \nabla_{\boldsymbol{y}} V(\boldsymbol{x})\,. \tag{49}$$

Furthermore, the value function admits a closed-form solution developed in the next proposition.

Proposition 9. *$V(t, \boldsymbol{y}) = \max_{c\in\mathcal{A}} J(t, \boldsymbol{y}, \boldsymbol{c})$ is a quadratic function of risk processes*

$$V(t, \boldsymbol{y}) \quad = \quad \boldsymbol{y}^\top A(t) \boldsymbol{y} + \boldsymbol{y}^\top B(t) + C(t)\,, \tag{50}$$

where $A(t) : [0, T] \to \mathbb{R}^{(p-1)\times(p-1)}$, $B(t) : [0, T] \to \mathbb{R}^{(p-1)}$, $C(t) : [0, T] \to \mathbb{R}$ are the solutions of a system of differential equations:

$$\begin{aligned} A(t)' \quad &= \quad -\left(2M^\top - \alpha I_{p-1} \right) A(t) - A(t)^\top Q_1^{-1\,\top} A(t)\,, \\ B(t)' \quad &= \quad -\left(M^\top - \alpha I_{p-1} \right) B(t)\,, \\ C(t)' \quad &= \quad \alpha C(t) - tr\left(\Sigma\, \Sigma^\top A(t) \right) - \frac{1}{4} B(t)^\top Q_1^{-1\,\top} B(t)\,, \end{aligned} \tag{51}$$

with the terminal conditions $A(T) = -Q_2$, $B(T) = 0$ and $C(T) = 0$.

To prove this result, it is sufficient to check that the value function (50) satisfies the HJB equation with the optimal control (49). The functions $A(t)$, $B(t)$ and $C(t)$ are computed by solving numerically the ODEs (51).

We use Proposition 9 to measure the accuracy of the approximation yielded by the PI-CGPR algorithm 1. For the linear-quadratic regulator problem, the step 4.b) of the algorithm becomes:

$$\boldsymbol{c}^{(n)}(t,\boldsymbol{y}) \;=\; \frac{1}{2}Q_1^{-1}\nabla_{\boldsymbol{y}}\widehat{V^{(n-1)}}(\boldsymbol{x})\,, \tag{52}$$

where $\widehat{V^{(n-1)}}(\boldsymbol{x})$ is the CGPR approximation of the value function.

Table 10 presents the parameters driving the risk processes of a two and four-dimensional linear-quadratic regulator. We consider maturities ranging from 1 up to 5 years. As for the consumption-investment problem, we set σ^* to 10^{-4}. Considering a lower value prevents the PI-CGPR algorithm from converging due to an ill-conditioned Gram matrix, $C^{(n)}\left(\mathring{\boldsymbol{X}},\bar{\boldsymbol{X}}\right)$. Other tuning parameters of the algorithm are provided in Table 11.

n=2 or n =4 D linear-quadratic regulator	
$M = 0.00\, I_n$	$\Sigma_{i,j} = \begin{cases} 0.25 & i = j \\ 0.08 & i \neq j \end{cases}$
$Q_1 = 1.50\, I_n$	$Q_2 = 0.25\, I_n$
$\alpha = 5\%$	
T from 1 to 5 years	

Table 10: Parameters defining the 2 risk processes of the linear-quadratic regulator problem

$\mathring{t}$	uniform distribution $[0, T)$
$\mathring{y}^{(j)}$ $j = 1, ..., 4$	uniform distribution $[-5, 5]$
σ^*	10^{-4}
$\eta_0^{(0)}$	0.00
$\eta_t^{(0)}$	0.75×standard deviation of $\mathring{t}$, $\sqrt{T^2/12}$
$\eta_j^{(0)}$ $j = 1, ..., 4$	0.75×standard deviation of $\mathring{y}^{(j)}$, $\sqrt{10^2/12}$
$c_t^{(j)(0)}$ $j = 1, ..., 4$	0.00

Table 11: Kernel and PI-CGPR parameters for the linear-quadratic regulator

Table 12 reports the MREs, as defined in Equation (44), for the sample sets $\mathring{\boldsymbol{X}}$ of sizes ranging from 250 to 750. The exact value function is computed by solving numerically Equations (51) with backward finite differences. We observe that MREs are small but rise with the time-horizon of the problem, while increasing the size of the sample set reduces the MREs. We will check later that the largest spreads between approximated and exact value functions are obtained with triplets close to the boundaries of the training set. The policy iterations algorithm needs more iterations to converge[5] than for the consumption-investment problem. The computational times, reported in Table 13, vary from 11.15s to 86.67s for the 2D problem and from 63.67 to 480.81s for the 4D problem. We run maximum 30 iterations of the gradient descent for optimizing hyper-parameters. The learning rate is $\rho = 0.01$ whereas the gradient with respect to $\boldsymbol{\eta}$ in Equation (35) is computed by finite differences with increments of $\boldsymbol{\eta}$ $\boldsymbol{\eta}$ set to $\max(0.01\boldsymbol{\eta}\,;\,0.01)$.

[5]We use $\delta = 1.00$ as stopping criterion.

Number of $y_t^{(i)}$		m and $d = m/2$					
		250	500	750	250	500	750
	T	MRE (%)			# of iterations		
2	1	0.10	0.10	0.10	1.0	1.0	1.0
	3	0.06	0.04	0.03	2.0	2.0	2.0
	5	0.15	0.08	0.06	2.0	2.0	2.0
4	1	0.57	0.08	0.04	1.0	2.0	2.0
	3	0.77	0.24	0.08	2.0	2.0	2.0
	5	1.58	0.50	0.21	2.0	2.0	2.0

Table 12: Mean relative errors (MREs) in % and number of iterations for converging

Number of $y_t^{(i)}$		m and $d = m/2$		
		250	500	750
	T	Time (seconds)		
2	1	11.15	30.21	62.72
	3	15.96	41.29	88.42
	5	15.26	40.92	86.67
4	1	114.26	67.55	162.76
	3	480.81	118.20	242.38
	5	364.92	63.67	182.72

Table 13: Computational time, PI-CGPR

Table 14 shows the errors $\mathring{Err}$ and $\bar{Err}$, defined in Equations (24) and (25), on the inner and boundary domains. The average errors remain small across all time horizons and sample sizes.

Number of $y_t^{(i)}$		m and $d = m/2$					
		250	500	750	250	500	750
	T	$\mathring{Err}$			$\bar{Err}$		
2	1	0.0133	0.0136	0.0145	0.0003	0.0002	0.0002
	3	0.0004	0.0004	0.0004	0.0002	0.0002	0.0002
	5	0.0011	0.0012	0.0012	0.0002	0.0002	0.0002
4	1	0.0204	0.0002	0.0003	0.0001	0.0002	0.0003
	3	0.0004	0.0004	0.0005	0.0002	0.0002	0.0003
	5	0.0015	0.0018	0.0021	0.0001	0.0002	0.0002

Table 14: Errors of approximation on the inner and boundary domains

We next focus on the 2D optimal regulator problem with $T = 5$ and m=750. Table 15 presents the evolution of inner and boundary errors (Equations (24); (25)) for each PI iteration. We see that the PI algorithm converges and that errors decrease at each iteration.

	$\mathring{Err}$	$\bar{Err}$	$\frac{1}{m}\sum_{i=1}^{m}\left(\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}_i) - \widehat{V^{(n-1)}}(\mathring{\boldsymbol{x}}_i)\right)^2$
1	0.8233	0.0001	19.9011
2	0.0854	0.0001	2.0742
3	0.0011	0.0001	0.0271

Table 15: Evolution of the average inner and boundary errors by PI iterations, 2 assets

Figure 5 presents the absolute spreads between the exact and approximated value functions, $\widehat{V^{(n)}}(\mathring{\boldsymbol{x}}) - V(\mathring{\boldsymbol{x}})$, on $[0,5] \times [-5,5]^2$ for various y_1 and constant $y_2 = 1.0$. We again observe local maxima, located in areas far from sampled points in $\mathring{\boldsymbol{X}}$. As with the consumption-investment problem, the largest errors are found on the boundaries of the training domain.

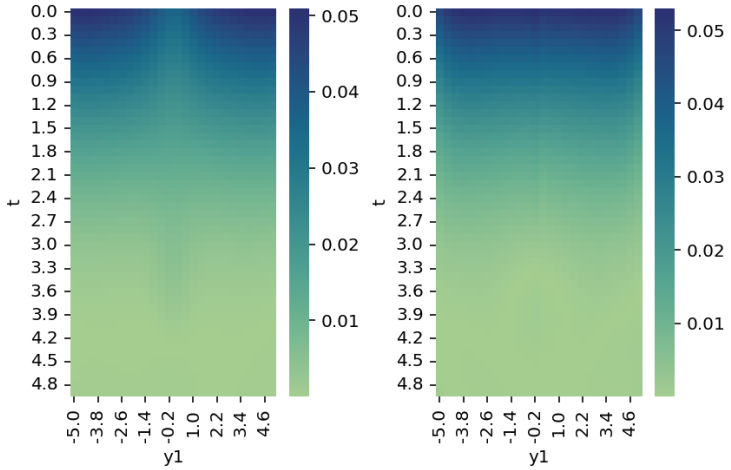

Figure 5: Left plot: heatmap of $\left|\frac{\widehat{V^{(n)}}(t,y) - V(t,y)}{V(t,y)}\right|$. Right plot: heatmap of $\left|\frac{c_t^{(1)(n)} - c_t^{(1)*}}{c_t^{(1)*}}\right|$

Figure 6 compares approximated and exact value functions and controls in a single scenario. The value function and controls are calculated with the assumption that $y_t^{(1)}$,$y_t^{(2)}$ remain equal to 2.5 . The graphs clearly demonstrate the accuracy of the PI-CGPR algorithm. However, we observe increasing deviations between the exact and approximated value functions and controls as the time to expiry increases.

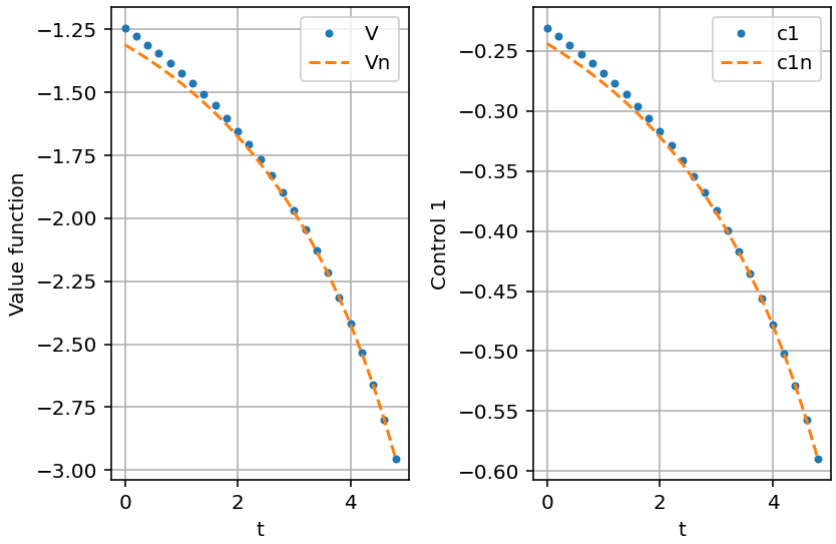

Figure 6: Comparison of exact and approximated value functions and controls in one sample path: $y_t^{(1)}, y_t^{(2)} = 3.0$ and $T = 5$ years

We next compare the PI-CGPR method with PINN and PDE approaches. For the PINN, we use a feed-forward neural network architecture consisting of one rescaling layer and two hidden layers with 64 neurons each, employing hyperbolic tangent (TANH) and Gaussian Error Lin-

ear Unit (GELU) activation functions. The output layer uses a linear activation function. The PINNs are trained on inner sample sets of the same size as those used for PI-CGPR. However, to improve accuracy, we use larger boundary datasets for PINNs ($d = m$) compared to PI-CGPR ($d = m/2$).

Table 16 reports the mean relative errors (MREs) on the inner domain. A comparison with Table 12 confirms the conclusions drawn from the previous example: PI-CGPR achieves lower errors using even fewer data points than PINNs. Additionally, PINNs generally require longer computational times. PI-CGPR is also more parsimonious, relying on only 4 to 6 bandwidth parameters, whereas the PINN models involve 4,481 and 4,609 parameters for the 2D and 4D problems, respectively.

Number	m and $d = m$	250	500	750	250	500	750
of $y_t^{(i)}$	T	MRE (%)			Time		
2	1	0.81	0.46	0.32	201.01	241.19	275.90
	3	0.96	0.66	0.45	205.81	245.47	274.09
	5	4.77	3.07	2.82	201.34	142.86	283.22
4	1	1.59	0.86	0.43	257.58	310.61	370.89
	3	2.72	1.27	0.98	259.51	314.65	375.55
	5	3.06	2.08	1.93	261.44	316.78	381.96

Table 16: mean relative errors and computational time

Table 17 presents the mean relative errors obtained using a backward PDE method. The two- and four-dimensional state variables are discretized on a grid ranging from $y_{min} = -5$ to $y_{max} = 5$ with 40 intermediate values. We consider 100 and 200 yearly steps per year. The MREs, computed on a random sample of 1,000 points in the inner domain, are generally below 1%. However, PI-CGPR consistently achieves even lower errors. For problems with only two state variables, the PDE method is significantly faster than both PI-CGPR and PINNs. However, for four-dimensional problems, its computational time exceeds that of PI-CGPR. In theory, the accuracy of the PDE method could be improved by refining the discretization steps. Yet, the size of the resulting hypercube of value functions grows rapidly, requiring memory resources that are impractical for standard personal computers.

PDE	T	MRE (%)		Time	
time steps per year		100	200	100	200
2 $y_t^{(i)}$s	1	0.58	0.59	0.0298	0.0622
	3	0.63	0.64	0.0945	0.1781
	5	0.70	0.71	0.1540	0.3033
4 $y_t^{(i)}$s	1	0.34	0.34	118.76	217.68
	3	0.50	0.51	462.95	1014.21
	5	0.65	0.65	863.35	1509.84

Table 17: PDE: mean relative errors and computational time

5.3 Constrained investment-consumption with stochastic volatility

To conclude the numerical analysis, we examine a variant of the investment-consumption problem that incorporates stochastic volatility and constrained controls. This problem, originally studied by Fleming and Hernández-Hernández (2003), poses significant challenges for numerical methods such as PINNs and PI-CGPR. These approaches may struggle to accurately capture

the dependence between the value function and volatility, as the wealth level tends to dominate the dynamics. Since this problem does not admit a closed-form analytical solution, we use numerical results from the PDE method as a reference for comparison.

An investor invests his wealth, $Y_t^{(1)}$, in a portfolio of cash, earning the risk-free rate $r \in \mathbb{R}^+$, and stock, whose the volatility, $Y_t^{(2)}$, is ruled by the Heston (1983) model. The part of the fund invested in stocks, denoted by π_t, is lower and upper bounded : $\pi_t \in [\pi_{min}, \pi_{max}]$. The consumption rate is also constrained : $b_t \in [b_{min}, b_{max}]$. In this setting, the two dimensional vector of variables, $\boldsymbol{Y}_t = \left(Y_t^{(1)}, Y_t^{(2)}\right)^\top$, controlled by $\boldsymbol{c}_t = (\pi_t, b_t)^\top$, is ruled by the SDE (1) with

$$\boldsymbol{\mu}_{\boldsymbol{Y}}(t, Y_t, \boldsymbol{c}_t) \quad = \quad \begin{pmatrix} (r + \pi_t (\mu - r) - b_t) Y_t^{(1)} \\ \kappa \left(\theta - Y_t^{(2)}\right) \end{pmatrix},$$

and

$$\Sigma_{\boldsymbol{Y}}(t, \boldsymbol{Y}_t, \boldsymbol{c}_t) \quad = \quad \begin{pmatrix} \pi_t Y_t^{(1)} \sqrt{1-\rho^2} \sqrt{Y_t^{(2)}} & \pi_t Y_t^{(1)} \rho \sqrt{Y_t^{(2)}} \\ 0 & \xi \sqrt{Y_t^{(2)}} \end{pmatrix}.$$

$\mu \in \mathbb{R}^+$ is the stock average return, $\kappa, \theta \in \mathbb{R}^+$ are the speed and the level of mean reversion of the stock variance and $\xi \in \mathbb{R}^+$ is the volatility of stock variance. We again consider utilities with a constant relative risk aversion (CRRA) coefficient γ:

$$U_1(b_t y) = \frac{(b_t y)^\gamma}{\gamma}, \; U_2(y) = \frac{(y)^\gamma}{\gamma}.$$

The payoff is the sum of the discounted expected utility of consumption and terminal wealth over $[0, T]$:

$$J(t, y, \boldsymbol{c}) \quad = \quad \mathbb{E}\left(\int_t^T e^{-\alpha(s-t)} \frac{(b_s Y_s)^\gamma}{\gamma} ds + e^{-\alpha(T-t)} \frac{(Y_T)^\gamma}{\gamma} \,|\, Y_t = y \right),$$

and the admissible set of controls is

$$\mathcal{A} = \{(\pi_t, b_t) \,|\, b_t \in [b_{min}, b_{max}], \pi_t \in [\pi_{min}, \pi_{max}] \; t \in [0, T]\}.$$

The inner and terminal domains are $\mathring{\mathcal{X}} = [0, T) \times \mathbb{R}^{2,+}$ and $\bar{\mathcal{X}} = T \times \mathbb{R}^{2,+}$. If $\boldsymbol{x} = (t, \boldsymbol{y})$, the value function is $V(\boldsymbol{x}) = \max_{c \in \mathcal{A}} J(\boldsymbol{x}, \boldsymbol{c})$. The optimal controls, $\boldsymbol{c}^*(t, \boldsymbol{y}) = (b^*(t, \boldsymbol{y}), \boldsymbol{\pi}^*(t, \boldsymbol{y}))$, are given by:

$$\begin{cases} b^*(t, \boldsymbol{y}) & = \min\left(b_{max}, \max\left(\frac{1}{y_1} \left(\partial_{y_1} V(\boldsymbol{x})\right)^{\frac{1}{\gamma - 1}}, b_{min}\right)\right), \\ \boldsymbol{\pi}^*(t, \boldsymbol{y}) & = \min\left(\pi_{max}, \max\left(\frac{(\mu - r)\partial_{y_1} V(\boldsymbol{x}) + \rho \xi y_2 \partial_{y_1 y_2} V(\boldsymbol{x})}{y_1 y_2 \partial_{y_1 y_1} V(\boldsymbol{x})}, \pi_{min}\right)\right). \end{cases}$$

Table 18 shows the parameters used in the numerical illustration. We assume that the investor consumes no more than 25% of the wealth and invests no more than 60% in the risky asset. The parameters of the Heston model are standard, with volatility reverting to a mean level of 25%. We consider three time horizons ranging from 1 to 5 years. Table 19 provides other PI-CGPR parameters. As in previous examples, the initial bandwidths are set as multiples of the standard deviations of the state variables in the inner training set.

α	4.00%	γ	0.30
r	3.00%	μ	6.00%
κ	1.5	θ	0.04
ξ	0.5	ρ	-0.7
c_{min}	0	c_{max}	0.25
π_{min}	0	π_{max}	0.60
T	1, 3, 5 years		

Table 18: Parameters of the constrained investment-consumption problem with stochastic volatility

$\mathring{t}$	uniform distribution $[0, T)$
$\mathring{y}^{(1)}$	uniform distribution $[0, 200)$
$\mathring{y}^{(2)}$	uniform distribution $[1e-6, 2.0)$
σ^*	10^{-4}
$\eta_0^{(0)}$	0
$\eta_t^{(0)}$	0.75×standard deviation of $\mathring{t}$, $\sqrt{T^2/12}$
$\eta_1^{(0)}$	0.75×standard deviation of $\mathring{y}^{(1)}$, $\sqrt{200^2/12}$
$\eta_2^{(0)}$	0.75×standard deviation of $\mathring{y}^{(2)}$, $\sqrt{(2-1e-6)^2/12}$
$c^{(0)}(t, y)$	10%
$\pi^{(0)}(t, y)$	10%

Table 19: Parameters of the PI-CGPR for the constrained investment-consumption problem with stochastic volatility

Table 20 reports the mean relative errors (MREs) between value functions obtained using the PI-CGPR and PDE methods. These MREs are computed on the same random subset of 1,000 points in the inner domain. The parameters of the PDE solver and the corresponding computation times are provided in Table 22. As in previous numerical examples, the MREs obtained with PI-CGPR remain below 1% and tend to increase with the time horizon. Additionally, increasing the size of the sample sets has a positive effect on accuracy.

	m and $d = m/2$					
	400	600	800	400	600	800
T	MRE (%)			# of iterations		
1	0.0646	0.0615	0.0808	2.0	2.0	2.0
3	0.1056	0.0824	0.0421	4.0	4.0	3.0
5	0.2253	0.0763	0.0589	6.0	9.0	4.0

Table 20: Mean relative errors (MREs) in % and number of iterations for converging

	m and $d = m/2$		
	400	600	800
T	Time (seconds)		
1	252.90	304.60	865.57
3	709.68	1266.38	896.12
5	1346.56	1266.09	914.48

Table 21: PI-CGPR computational time

Table 22 presents the mean relative errors (MREs) obtained using a neural network with three hidden layers. The activation functions used are GELU for the hidden layers and RELU for the output layer. The network is trained on the same sample sets as the PI-CGPR. Training is performed over three phases of 1,000 epochs each, with decreasing learning rates of 0.02, 0.01, and 0.001. A comparison with Table 20 confirms that PI-CGPR achieves better or comparable accuracy to PINNs. However, the computational time for PI-CGPR is longer than that of PINNs for time horizons of 3 and 5 years, though it remains similar to that of the PDE method. This final test reinforces the conclusion that PI-CGPR is a viable alternative to both PDE and PINN approaches.

	PINN			PDE
m , $d = m/2$	800		Grid size y_1, y_2	500 ,500
Layers	64-64-64		Grid size, time	$4000 \times T$
Weights	8641		y_1, y_2 bounds	[0,300] , [1e-3,10]
T	MRE (%)	Time	T	Time
1	0.1095	373.60	1	207.66
3	0.1498	364.25	3	763.84
5	0.2140	364.57	5	1140.01

Table 22: Mean relative errors (MREs) in % for PINNs and computation times

6 Conclusions

This paper presents a novel iterative algorithm that combines policy iterations with constrained Gaussian process regressions (CGPR) to solve stochastic control problems. By leveraging the strengths of Gaussian processes for regression and the iterative nature of policy iterations, our method offers a robust and efficient approach to approximating value functions and optimizing controls.

Our numerical experiments, focusing on the consumption-investment, linear-quadratic regulator problems and constrained consumption-investment with stochastic volatility, demonstrate the efficacy and accuracy of the proposed PI-CGPR algorithm. The results indicate that the algorithm consistently improves the value function at each iteration and converges to the optimal solution within a reasonable number of iterations. The analytical tractability of Gaussian kernels further enhances the computational efficiency of our method.

The PI-CGPR method offers several additional advantages. It is largely data-driven and requires the estimation of only a small number of hyperparameters, specifically, the bandwidths. In contrast, PINNs rely on more complex network architectures involving several thousand neural weights. PI-CGPR also benefits from strong prior assumptions through Bayesian inference, allowing it to generalize effectively from limited data. As demonstrated throughout this article, PINNs typically require larger datasets to learn structural relationships from scratch. Moreover, for problems involving more than two state variables, PI-CGPR is not more computationally intensive than PDE or PINN approaches.

However, the PI-CGPR algorithm is not without limitations. One drawback is its reliance on the choice of kernel and hyperparameters, which can affect both the accuracy and stability of the solution. Like PINNs, PI-CGPR requires careful tuning of the training set size and the regularization parameter to avoid numerical instabilities. Additionally, the accuracy of the approximation tends to deteriorate near the boundaries of the training domain. Another limitation is the need to invert a large Gram matrix, which poses computational challenges and restricts

the method's scalability to problems with a high number of state variables. Nonetheless, techniques such as reduced-rank or greedy approximations of the Gram matrix can help manage high-dimensional settings. For further details, we refer the reader to Chapter 8 of Rasmussen and Williams.

Despite these limitations, the PI-CGPR algorithm's ability to solve control problems makes it a valuable tool for a wide range of applications in finance, engineering, and beyond. Future research could explore the extension of this framework to other types of kernels and the incorporation of additional constraints on controls to further enhance its applicability and performance.

Appendix A, partial derivatives of $\beta^{(n)}(\boldsymbol{x})$

In the policy iterations algorithm 1, we consider optimization problems which admit explicit optimal controls in terms of the gradient and Hessian of intermediates approximated payoffs:

$$\boldsymbol{c}^{(n+1)} = \boldsymbol{c}^{(n)}\left(\boldsymbol{x}, \partial_t \widehat{V^{(n-1)}}(\boldsymbol{x}), \nabla_{\boldsymbol{y}} \widehat{V^{(n)}}(\boldsymbol{x}), \mathcal{H}_{\boldsymbol{y}} \widehat{V^{(n)}}(\boldsymbol{x})\right),$$

where $\widehat{V^{(n)}}(\boldsymbol{x}) = \boldsymbol{\beta}^{(n)}(\boldsymbol{x})^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}$. The first and second order derivatives of $\widehat{V^{(n)}}$ ruling the control are

$$\partial_t \widehat{V^{(n)}}(\boldsymbol{x}) = \left(\partial_t \boldsymbol{\beta}^{(n)}(\boldsymbol{x})\right)^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}, \tag{53}$$

$$\partial_{y_k} \widehat{V^{(n)}}(\boldsymbol{x}) = \left(\partial_{y_k} \boldsymbol{\beta}^{(n)}(\boldsymbol{x})\right)^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}, \tag{54}$$

$$\partial_{y_l}\partial_{y_k} \widehat{V^{(n)}}(\boldsymbol{x}) = \left(\partial_{y_l}\partial_{y_k} \boldsymbol{\beta}^{(n)}(\boldsymbol{x})\right)^\top C^{(n)}\left(\mathring{\boldsymbol{X}}, \bar{\boldsymbol{X}}\right)^{-1}\begin{pmatrix} \boldsymbol{z}^{(n)} \\ \boldsymbol{h} \end{pmatrix}. \tag{55}$$

The derivative of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ w.r.t. time is a $(m+d)$ vector

$$\partial_t \boldsymbol{\beta}^{(n)}(\boldsymbol{x}) = \begin{pmatrix} \left(\partial_t \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right)\right)_{i=1,\dots,m} \\ \left(\partial_t k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right)\right)_{j=1,\dots,d} \end{pmatrix},$$

where

$$\begin{cases} \partial_t \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right) & = \frac{k(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)}{\eta_t^2} - \frac{\left(t - \mathring{t}_i\right)}{\eta_t^2} \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right) \\ \partial_t k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right) & = \frac{\left(\bar{t}_j - t\right)}{\eta_t^2} k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right). \end{cases}$$

The derivative of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ w.r.t. y_k in Eq. (54) is a $(m+d)$ vector

$$\partial_{y_k} \boldsymbol{\beta}^{(n)}(\boldsymbol{x}) = \begin{pmatrix} \left(\partial_{y_k} \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right)\right)_{i=1,\dots,m} \\ \left(\partial_{y_k} k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right)\right)_{j=1,\dots,d} \end{pmatrix},$$

with elements equal to

$$\begin{cases} \partial_{y_k} \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right) & = \left[\boldsymbol{e}_k^\top \Sigma_{\boldsymbol{Y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{Y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i) \right. \\ & \quad \left. + \boldsymbol{\mu}_{\boldsymbol{Y}}(\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)})^\top \boldsymbol{e}_k\right] \frac{k(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)}{\eta_k^2} \\ & \quad - \mathcal{L}_{\mathring{\boldsymbol{x}}_i, \boldsymbol{c}^{(n)}} k\left(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i\right) \boldsymbol{e}_k^\top \boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i), \\ \partial_{y_k} k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right) & = \frac{\left(\bar{y}_{j,k} - y_k\right)}{\eta_k^2} k\left(\bar{\boldsymbol{x}}_j, \boldsymbol{x}\right). \end{cases}$$

The Hessian of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ is the $(p-1)\times(p-1)$ matrix of second order derivatives

$$\partial_{y_l}\partial_{y_k}\boldsymbol{\beta}^{(n)}(\boldsymbol{x}) \quad = \quad \begin{pmatrix} \left(\partial_{y_l}\partial_{y_k}\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k\left(\boldsymbol{x},\mathring{\boldsymbol{x}}_i\right)\right)_{i=1,...,m} \\ \left(\partial_{y_l}\partial_{y_k}k\left(\bar{\boldsymbol{x}}_j,\boldsymbol{x}\right)\right)_{j=1,...,d} \end{pmatrix},$$

where

$$\begin{cases} \partial_{y_l}\partial_{y_k}\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k\left(\boldsymbol{x},\mathring{\boldsymbol{x}}_i\right) & = \left(\boldsymbol{e}_k^\top\Sigma_{\boldsymbol{Y}}(\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)})\Sigma_{\boldsymbol{Y}}(\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)})^\top\boldsymbol{e}_l\right)\frac{k(\boldsymbol{x},\mathring{\boldsymbol{x}}_i)}{\eta_k^2\eta_l^2} \\ & \quad -\left(\partial_{y_k}\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k\left(\boldsymbol{x},\mathring{\boldsymbol{x}}_i\right)\right)\boldsymbol{e}_l^\top\boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x},\mathring{\boldsymbol{x}}_i) \\ & \quad -\left(\partial_{y_l}\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k\left(\boldsymbol{x},\mathring{\boldsymbol{x}}_i\right)\right)\boldsymbol{e}_k^\top\boldsymbol{d}_{\mathring{\boldsymbol{y}}_i}(\boldsymbol{x},\mathring{\boldsymbol{x}}_i) \\ & \quad -\mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}}k\left(\boldsymbol{x},\mathring{\boldsymbol{x}}_i\right)\left(\boldsymbol{e}_l^\top\boldsymbol{H}_{\boldsymbol{y}}(\boldsymbol{x},\mathring{\boldsymbol{x}}_i)\boldsymbol{e}_k\right), \\ \partial_{y_l}\partial_{y_k}k\left(\bar{\boldsymbol{x}}_j,\boldsymbol{x}\right) & = \left(\frac{\left(\bar{y}_{j,k}-y_k\right)\left(\bar{y}_{j,l}-y_l\right)}{\eta_k^2\eta_l^2}-\mathbf{1}_{\{k=l\}}\frac{1}{\eta_k^2}\right)k\left(\bar{\boldsymbol{x}}_j,\boldsymbol{x}\right). \end{cases}$$

Appendix B, PINN

We approximate the value function that solves the HJB equation ([eq:HJB_equation]) using a neural network. This network takes as input the vector of state variables $\boldsymbol{x}$. There is no universally accepted procedure for determining the optimal architecture of a neural network. Single-layer neural networks can approximate regular functions arbitrarily well; however, achieving reasonable accuracy may require a large number of neurons. In this work, we instead adopt a feed-forward architecture, where information flows sequentially through multiple neural layers.

Definition Let l, n_0, n_1, ..., $n_l \in \mathbb{N}$ be respectively the number of layers and neurons in each layer. $n_0 = p$ is the size of input vector. The activation function of layer $k = 1, 2, ..., l$ is noted $\phi_k(.) : \mathbb{R} \to \mathbb{R}$. Let $A_k \in \mathbb{R}^{n_k} \times \mathbb{R}^{n_{k-1}}$, $\boldsymbol{b}_k \in \mathbb{R}^{n_k}$ for $k = 1, ..., l$, be neural weights defining the input, intermediate and output layers. We define the following functions

$$\psi_k(\boldsymbol{x}) = \phi_k\left(A_k\boldsymbol{x}+\boldsymbol{b}_k\right), k = 1, ..., l$$

where activation functions $\phi_k(.)$ are applied component-wise. The neural network is a function $F : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ defined by

$$F(\boldsymbol{z}) \quad = \quad \psi_l \circ \psi_{l-1}\left(.,\boldsymbol{z}\right) ... \circ \psi_2\left(.,\boldsymbol{z}\right) \circ \psi_1(\boldsymbol{z}).$$

Once the network architecture is selected, the model is trained by minimizing a loss function that reflects the approximation error. This error is evaluated by substituting the value function $V(.)$ with $F(.)$ in the HJB equation, at randomly sampled points from the inner domain $\mathring{\mathcal{X}}$ and on the boundary $\bar{\mathcal{X}}$.

During the training phase, the error of approximation is measured for a sample of m points, $\mathring{\boldsymbol{X}} \subset \mathring{\mathcal{X}}$. The error at expiry is measured with another sample of size d, $\bar{\boldsymbol{X}} \subset \bar{\mathcal{X}}$. We denote by $\boldsymbol{\Theta}$, the vector containing all neural weights $(A_k, \boldsymbol{b}_k)_{k=1,...,l}$. At points of $\mathring{\boldsymbol{X}}$, we define for $j = 1, ..., n_{\mathcal{D}}$, the error in the HJB equation when V is replaced by the neural network:

$$\mathring{e}_j(\boldsymbol{\Theta}) = U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^*) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^*}F(\mathring{\boldsymbol{x}}), \tag{56}$$

where $\boldsymbol{c}^*$ is the optimal control expressed as function of partial derivatives of $F(.)$. We refer to $\mathring{e}_j$ as the error on the inner domain since it measures the goodness of fit before expiry. The average quadratic loss on $\mathring{\boldsymbol{X}}$ is the first component of the total loss function used to fit the neural network,

$$\mathring{\mathcal{L}}(\boldsymbol{\Theta}) = \frac{1}{d}\sum_{j=1}^{d}\mathring{e}_j(\boldsymbol{\Theta})^2. \tag{57}$$

As the error $\mathring{e}_j(\boldsymbol{\Theta})$ depends on partial derivatives with respect to state variables, special attention must be given to the accuracy of their calculation. We use automatic or algorithmic differentiation in TensorFlow. Automatic differentiation leverages the fact that every computer calculation executes a sequence of elementary arithmetic operations and elementary functions, allowing us to compute partial derivatives with accuracy up to the working precision.

On the terminal boundary sample set $\bar{\boldsymbol{X}}$, we define the error $\bar{e}_k(\boldsymbol{\Theta})$ for $k = 1, ..., m$, as the difference between the output of the neural network and the terminal utility:

$$\bar{e}_k(\boldsymbol{\Theta}) \quad = \quad F(\bar{\boldsymbol{x}}) - U_2(\bar{\boldsymbol{x}})\,. \tag{58}$$

The average quadratic loss at expiry is the second component of the total loss function.

$$\bar{\mathcal{L}}\,(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{k=1}^{m} \bar{e}_k(\boldsymbol{\Theta})^2\,. \tag{59}$$

The optimal network weights are found by minimizing the losses in $\mathring{\boldsymbol{X}}$ and $\bar{\boldsymbol{X}}$ with a gradient descent

$$\boldsymbol{\Theta}_{opt} = \arg\min_{\boldsymbol{\Theta}} \left[\mathring{\mathcal{L}}\,(\boldsymbol{\Theta}) + \bar{\mathcal{L}}\,(\boldsymbol{\Theta})\right]\,. \tag{60}$$

Appendix C Proofs

Proof of Proposition 1:

Proof. From Eq. (9), $\forall \mathring{\boldsymbol{x}} \in \mathring{\mathcal{X}}$, we know that

$$U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(k)}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(k)}} V^{(k)}(\mathring{\boldsymbol{x}}) = 0 \qquad \text{for} \quad k = n-1\,,\, n\,.$$

By subtraction, we infer the equality

$$U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) - U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n-1)}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n)}(\mathring{\boldsymbol{x}}) - \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n-1)}} V^{(n-1)}(\mathring{\boldsymbol{x}}) = 0\,. \tag{61}$$

But from Eq. (8), $\boldsymbol{c}^{(n)}$ is optimal compared to $\boldsymbol{c}^{(n-1)}$:

$$U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n-1)}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n-1)}} V^{(n-1)}(\mathring{\boldsymbol{x}}) \leq U_1(\mathring{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) + \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n-1)}(\mathring{\boldsymbol{x}})\,. \tag{62}$$

Summing up Eq. (61) and (62), we obtain the inequality:

$$\mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n)}(\mathring{\boldsymbol{x}}) - \mathcal{L}_{\mathring{\boldsymbol{x}},\boldsymbol{c}^{(n)}} V^{(n-1)}(\mathring{\boldsymbol{x}}) \leq 0\,.$$

For $\bar{\boldsymbol{x}} \in \bar{\mathcal{X}}$, we have $V^{(n)}(\bar{\boldsymbol{x}}) = V^{(n-1)}(\bar{\boldsymbol{x}}) = U_2(\bar{\boldsymbol{x}})$ and Eq. (10) holds. □

Proof of Proposition 2:

Proof. For $s \geq t$ and $k = n-1\,,\, n$, we have that $\mathbb{E}\left(dV^{(k)}(\boldsymbol{x})|\mathcal{F}_{t-}\right) = \mathcal{L}_{\boldsymbol{x},\boldsymbol{c}^{(k)}} V^{(k)}(\boldsymbol{x})\, dt$. Therefore,

$$V^{(k)}\left(\boldsymbol{X}_s^{(n)}\right) - \int_0^s \mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(k)}(\boldsymbol{X}_u^{(n)})\, du\,,$$

is a martingale, denoted $\left(M_s^{(k)}\right)_{s\geq 0}$. Using the martingale representation theorem and $X_t^{(n)} = \boldsymbol{x}$, we can rewrite $V^{(k)}\left(\boldsymbol{X}_s^{(n)}\right)$ as follows

$$V^{(k)}\left(\boldsymbol{X}_s^{(n)}\right) \quad = \quad V^{(k)}\left(\boldsymbol{x}\right) + \int_t^s \mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(k)}\left(\boldsymbol{X}_u^{(n)}\right)\, du + M_s^{(k)} - M_t^{(k)}\,. \tag{63}$$

Let us define the following process:

$$S_s^{(n)} = V^{(n)}\left(\boldsymbol{X}_s^{(n)}\right) - V^{(n-1)}\left(\boldsymbol{X}_s^{(n)}\right).$$

Using the martingale representation of $V^{(k)}$, Eq. (63), we infer that

$$\begin{aligned} S_s^{(n)} &= V^{(n)}(\boldsymbol{x}) - V^{(n-1)}(\boldsymbol{x}) + \int_t^s \left(\mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(n)} - \mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(n-1)}\right)(\boldsymbol{X}_u^{(n)})\, du \\ &\quad + M_s^{(n)} + M_s^{(n-1)} - M_t^{(n)} - M_t^{(n-1)}. \end{aligned}$$

From Proposition 1, $\mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(n)} \leq \mathcal{L}_{\boldsymbol{X}^{(n)},\boldsymbol{c}^{(n)}} V^{(n-1)}$ and the integral is non-positive. Taking expectations and letting $s \to T$, as $\lim_{s\to T} \mathbb{E}\left(S_s^{(n)}\right) \geq 0$ under Assumption (12), we conclude that

$$V^{(n)}(\boldsymbol{x}) - V^{(n-1)}(\boldsymbol{x}) \geq 0.$$

□

Proof of Proposition 5.

Proof. The last d elements of $\boldsymbol{\beta}^{(n)}(\boldsymbol{x})$ are equal to $(k(\bar{\boldsymbol{x}}_i, \boldsymbol{x}))_{i=1,\dots,d}$, directly calculable with Equation (26). By direct differentiation, we check that the gradient and Hessian of the kernel are given by

$$\begin{cases} \nabla_{\tilde{\boldsymbol{y}}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) &= \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})\, k(\boldsymbol{x}, \tilde{\boldsymbol{x}}), \\ \mathcal{H}_{\tilde{\boldsymbol{y}}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) &= \boldsymbol{H}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})\, k(\boldsymbol{x}, \tilde{\boldsymbol{x}}). \end{cases}$$

We infer from this last equation, that $\mathcal{L}_{\tilde{\boldsymbol{x}}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ depends on $\boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ and $\boldsymbol{H}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ as follows:

$$\begin{aligned} \mathcal{L}_{\tilde{\boldsymbol{x}},\boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \Big(& \left(t - \tilde{t}\right)/\eta_t^2 - \alpha + \boldsymbol{\mu}_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)})^\top \boldsymbol{d}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \\ & + \frac{1}{2}\mathrm{tr}\left(\Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)}) \Sigma_{\boldsymbol{Y}}(\tilde{\boldsymbol{x}}, \boldsymbol{c}^{(n)})^\top \boldsymbol{H}_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}, \tilde{\boldsymbol{x}})\right)\Big)\, k(\boldsymbol{x}, \tilde{\boldsymbol{x}}), \end{aligned} \tag{64}$$

Equations (64) allows us to retrieve the expression of $\boldsymbol{\beta}_1^{(n)}(\boldsymbol{x}) = \mathcal{L}_{\mathring{\boldsymbol{x}}_i,\boldsymbol{c}^{(n)}} k(\boldsymbol{x}, \mathring{\boldsymbol{x}}_i)$. □

Acknowledgement

Donatien Hainaut thanks the FNRS (Fonds de la recherche scientifique) for the financial support through the EOS project Asterisk (research grant 40007517).

Statements and declarations

Authors have seen and agree with the contents of the manuscript and there is no financial interest to report.

References

[1] Al-Aradi A., Correia A., Jardim G., de Freitas Naiff D., Saporito Y. 2022. Extensions of the deep Galerkin method. App. Math. and Computation, 430, 127287.

[2] Banerjee A., Dunson D.B., Tokdar S.T., 2013. Efficient Gaussian process regression for large datasets. Biometrika 100 (1), 75–89.

[3] Barber D., Wang Y. 2014. Gaussian processes for Bayesian estimation in ordinary differential equations. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), pages 1485-1493. JMLR Workshop and Conference Proceedings.

[4] Bertsekas D.P., 2011. Approximate policy iteration: a survey and some new methods. Journal of Control Theory and Technology, 9, 310-335.

[5] Calderhead B. , Girolami M., Lawrence N.D. 2009. Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes, in Advances in Neural Information Processing Systems, pp. 217-224.

[6] Cialenco I., Fasshauer G.E., Ye Q. 2012. Approximation of Stochastic Partial Differential Equations by a Kernel-based Collocation Method. International Journal of Computer Mathematics. Vol 89 (18), p. 2543-2561.

[7] Choi J., Son Y. , Jeong M.K. 2024. Gaussian kernel with correlated variables for incomplete data. Annals of Operational Research 341, 223–244.

[8] Cockayne J., Oates C., Ipsen I., Girolami M. 2019. A Bayesian conjugate gradient method (with discussion). Bayesian Analysis, 14(3), p. 937–1012.

[9] Deringer V., Bartok A., Bernstein N., Wilkins D., Ceriotti M., 2021. Gaussian Process Regression for Materials and Molecules. Chemical Reviews 121 (16), 10073-10141.

[10] Dupret J.L., Hainaut D., 2024. Deep learning for high-dimensional continuous-time stochastic optimal control without explicit solution. UCLouvain working paper. https://dial.uclouvain.be/pr/boreal/object/boreal:287848.

[11] Fine C.H., Porteus, E.L. 1989. Dynamic process improvement. Operations research 37 (4), 514-673.

[12] Fleming W., Hernández-Hernández D., 2003. An Optimal Consumption Model with Stochastic Volatility. Finance and Stochastics, 7, 245-262.

[13] Glau K., Wunderlich L. 2022. The deep parametric PDE method and applications to option pricing. App. Math. and Computation, 432, 127355

[14] Glau K., Wunderlich L. 2024. Neural network expression rates and applications of the deep parametric PDE method in counterparty credit risk. Annals of Operations Research 336, 331–357.

[15] Graepel T. 2003. Solving Noisy Linear Operator Equations by Gaussian Processes: Application to Ordinary and Partial Differential Equations. ICML'03: Proceedings of the Twentieth International Conference on International Conference on Machine Learning Pages 234 - 241

[16] Hainaut D., Casas A., 2024.Option pricing in the Heston model with physics inspired neural networks. Annals of finance. https://doi.org/10.1007/s10436-024-00452-7

[17] Hainaut D., Vrins F., 2024. European option pricing with model constrained Gaussian process regressions. UCLouvain ISBA discussion paper. https://dial.uclouvain.be/pr/boreal/object/boreal:292395

[18] Hainaut D. 2024. American option pricing with model constrained Gaussian process regressions. UCLouvain ISBA discussion paper. https://dial.uclouvain.be/pr/boreal/object/boreal:292665

[19] Heston, S. L. 1993. A closed-form solution for options with stochastic volatility with applications to bond and currency options. Review of Financial Studies. 6 (2), 327–343.

[20] Howard R.A. 1960. Dynamic programming and Markov processes. MIT press and J. Wiley & Sons, New-York-London.

[21] Hu M., Xu J., Chen C-H., Hu J-Q. 2025 Optimal computation budget allocation with Gaussian process regression. European Journal of Operational Research, 322 (1), 147-156.

[22] Jacka S.D., Mijatovic A. 2017. On the policy improvement algorithm in continuous time. Stochastics 89(1), 348–359.

[23] Jakubik J., Binding A., Feuerriegel S. 2021 Directed particle swarm optimization with Gaussian-process-based function forecasting. European Journal of Operational Research, 295 (1), 157-169

[24] Mlakar M., Petelin D., Tusar T., Pilipic B. 2015. GP-DEMO: Differential Evolution for Multiobjective Optimization based on Gaussian Process models. European Journal of Operational Research, 243 (2), 347-361

[25] Murphy K.P. 2012. Machine learning: a probabilistic perspective, MIT press.

[26] Nguyen N.C., Peraire J. 2015. Gaussian functional regression for linear partial differential equations. Computational methods in applied mechanics and engineering, 287, p. 69-89.

[27] Price I., Fowkes J., Hopman D. 2019. Gaussian processes for unconstrained demand. European Journal of Operational Research, 275 (2), 621-634.

[28] Pförtner, M., Steinwart I., Hennig P., Wenger J. 2022. Physics-Informed Gaussian Process Regression Generalizes Linear PDE Solvers. arXiv:2212.12474

[29] Rasmussen C.E., Kuss M. 2004. Gaussian Processes in Reinforcement Learning. Advances in neural information processing systems, 16, 751–759.

[30] Rasmussen C.E., Williams C.K.I. 2006. Gaussian processes for machine learning, MIT press.

[31] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686–707.

[32] Roberts S, Osborne M, Ebden M, Reece S, Gibson N, Aigrain S., 2013. Gaussian processes for time-series modelling. Phil Trans R Soc A 371: 20110550.

[33] Sacks J., Welch W.J., Mitchell T.J., Wynn. H.P. 1989. Design and analysis of computer experiments. Statistical Science, 409–423.

[34] Santner T.J., Williams B.J., Notz W.I. 2003. The design and analysis of computer experiments. Springer Series in Statistics.

[35] Särkkä S. 2011. Linear operators and stochastic partial differential equations in Gaussian process regression. In Artificial Neural Networks and Machine Learning - ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part II, pages 151-158. Springer, Berlin, Heidelberg.

[36] Sirignano J., Spiliopoulos K., 2018. DGM: a deep learning algorithm for solving partial differential equations, Journal of Computational Physics, 375, 1339–1364.

[37] Swiler L., Gulian M., Frankel A., Safta C., Jakeman J. 2020. A survey of constrained Gaussian process regression: approaches and implementation challenges. Journal of Machine Learning for Modeling and Computing 1(2) 119-156.

[38] Touzi, N. (2012). Optimal stochastic control, stochastic target problems, and backward SDE, volume 29. Springer Science & Business Media.

[39] Yang X., Zhang J., 2025. Gaussian Process Policy Iteration with Additive Schwarz Acceleration for Forward and Inverse HJB and Mean Field Game Problems. https://arxiv.org/abs/2505.00909