POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Differential Prediction-Enhanced Monte Carlo: A Fast Machine Learning-Aided Variance Reduction Method

Laurea Magistrale in Mathematical Engineering - Ingegneria Matematica

Author: Luca Amadori

Advisor: Prof. Daniele Marazzina

Academic year: 2024-2025

1. Introduction

Monte Carlo simulations are a powerful tool for pricing financial instruments in the absence of a closed-form pricing formula. Their use stems from the theoretical guarantee of unbiasedness of the Monte Carlo estimator and the precise error quantification by confidence intervals. However, the biggest limitation of the method is that obtaining accurate estimates when dealing with nested simulations or path-dependent payoffs is computationally expensive, due to its $O(1/\sqrt{M})$ convergence rate after M replications. A possible solution is to use a variance reduction technique (e.g., the control variates technique), which can achieve the same level of estimator variance with fewer simulations. Furthermore, replacing traditional Monte Carlo steps with a machine learning-based prediction model can drastically accelerate the evaluation process.

The Prediction-Enhanced Monte Carlo (PEMC) framework [6] combines these concepts: it employs a machine learning-based model to construct a control variate in a fast and parallelizable way. By executing the computationally intensive data generation and model training offline, PEMC exploits the evaluation speed of machine learning models in the online phase. This approach allows for estimating the control variate's mean as the average of a large number of rapid simulations, bypassing the need for analytical formulas and extending the applicability of the control variates technique to virtually any practical scenario. At the same time, the statistical guarantee of unbiasedness of the estimator is preserved.

Despite these significant theoretical advantages offered by PEMC, the theory itself (Lemma 6 and Appendix in [6]) indicates that PEMC's variance reduction over Monte Carlo is guaranteed only when using a large number of training samples. As a result, the data generation and training processes become computationally demanding tasks, necessitating their offline execution within the original PEMC framework. However, recent literature has introduced a preprocessing method based on the combination of Principal Component Analysis (PCA) and differential PCA [4, 5], which exploits partial derivatives of the label with respect to the features to obtain a small group of very informative variables.

This thesis proposes integrating this data preparation algorithm into PEMC in the Differential PEMC approach. By leveraging the relevance of the transformed features, the method achieves the same level of variance reduction using a dras-

tically lower number of training samples (e.g., 128 vs. 1.28 million in the best-case scenario). The consequent reduction in training time is a significant innovation that could enable a fully online use of the PEMC method, allowing for more frequent model retraining to better adapt to evolving market conditions.

2. Theoretical Framework

The theoretical concepts necessary to fully understand the problem and its implementation choices include the control variates method, skip connections in deep learning, automatic differentiation, and differential PCA. The first is a fundamental theoretical concept for the PEMC approach, while the use of skip connections is pivotal in the neural network architectures used in the practical applications of the PEMC approach. The last two concepts constitute the theoretical basis for the Differential PEMC implementation.

Control Variates

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and Y a square-integrable random variable. The standard Monte Carlo estimator for $\theta = \mathbb{E}[Y]$ is given by:

$$\hat{\theta}_{MC} = \frac{1}{n}\sum_{i=1}^{n} Y_i,$$

where n is the number of samples. The control variates method is a variance reduction technique which exploits an auxiliary square-integrable random variable X correlated with Y and with a known mean $\mathbb{E}[X]$ to obtain an estimator with lower variance than the Monte Carlo one [2]. The control variates estimator is defined as:

$$\hat{\theta}_{CV} = \frac{1}{n}\sum_{i=1}^{n} \big(Y_i - \alpha(X_i - \mathbb{E}[X])\big),$$

with $X_i, i = 1, ..., n$ a set of i.i.d. copies of X, with known mean $\mathbb{E}[X]$. This estimator remains unbiased for θ for any value of α [2]. In addition, by appropriately choosing X and α, it is possible to achieve variance reduction with respect to Monte Carlo. In particular, for the optimal choice of α:

$$\frac{Var(\hat{\theta}_{CV})}{Var(\hat{\theta}_{MC})} = 1 - \rho^2,$$

with ρ the correlation between X and Y. Therefore, by choosing the optimal α and any X such that $Cov(X, Y) \neq 0$ and $Var(X) \neq 0$, it is possible to achieve variance reduction over Monte Carlo. Moreover, the higher the correlation between X and Y, the better the variance reduction.

Skip Connections in Deep Learning

In deep learning, skip connections [3] are a residual learning tool that helps prevent the vanishing gradient problem.
During the training of a neural network, its parameters are updated to minimize a loss function L, usually the mean squared error, according to the rule:

$$w_i' = w_i - \lambda \frac{\partial L}{\partial w_i},$$

where λ is the learning rate, w_i the value of the i-th parameter before the update and w_i' the new parameter value. The backpropagation mechanism exploits the chain rule of differentiation to compute these partial derivatives starting from the last layer and going backwards. Especially for deep networks, the multiplication of these gradients often results in a very small product, meaning that the parameters of the early layers are updated by a very small amount, leading to a loss of predictive power ("vanishing gradient problem"). Skip connections solve this problem by offering an alternative path for the gradient to flow during backpropagation. The method consists of skipping some layers and adding the output of one layer to the output of a deeper layer, as shown in Figure 1.

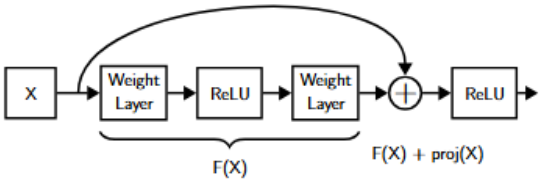

Figure 1: Skip connections in ResNet-style

In the figure above, F(X) is the residual function and proj(X) is the linear projection of X that matches the dimension of X with the dimension of F(X). In practice, F(X) could contain any number of layers, even though it is usually composed of one to three layers.

Computing Gradients with Automatic Differentiation

The path-wise method [2] produces unbiased estimators of price sensitivities without perturbing input parameters. Formally, let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered probability space with the filtration $\mathbb{F} = \{\mathcal{F}_n\}_{n=0,\dots,N}$ and $\theta \in \Theta \subseteq \mathbb{R}^k$ the vector of parameters with respect to which the sensitivities are computed. The discounted payoff function at the final state is $f(X_N(\theta))$, where $f : \mathbb{R}^d \to \mathbb{R}$ and $X = \{X_n\}_{n=0,\dots,N}$ an $\mathbb{F}$-adapted process with values in $\mathbb{R}^d$. In practice, X could be, for example, a vector of underlying assets' prices or a vector of forward rates.
Let $V(\theta) = \mathbb{E}[f(X_N(\theta))]$, the path-wise method estimates the vector of price sensitivities $\nabla_\theta V$ as:

$$\nabla_\theta f(X_N) = \left(\frac{dX_N}{d\theta}\right)^T \nabla_x f(X_N),$$

where $\nabla_x f(X_N)$ is the gradient of the discounted payoff with respect to the final state X_N, $\frac{dX_N}{d\theta}$ is the Jacobian matrix of the sensitivity of the final state with respect to θ and the equality comes from the chain rule of differentiation [2]. While the forward method calculates this estimate moving forward in time with a matrix recursion, the adjoint method executes the calculation backwards. It uses a vector recursion, reducing the computational complexity per step from $O(d^2k)$ to $O(d^2 + kd)$. This characteristic makes the adjoint method highly beneficial when computing sensitivities with respect to multiple parameters, since the computational cost scales better with the state dimension d compared to the forward method [1].

Differential PCA

Reducing the initial set of features of the training dataset to a subset of significant ones can significantly improve machine learning models' performance. Standard Principal Component Analysis (PCA) performs an orthonormal transformation to filter data based strictly on variation, defined as $\mathbb{E}[(X \cdot u_i)^2]$ along the normalized eigenvector u_i [5]. However, this metric is often inadequate for derivatives pricing and risk management, since a large variation along an irrelevant axis may not affect the price or payoff, while a small variation along a highly relevant axis can [4].

To overcome these limitations, differential PCA introduces supervision and filters inputs based on relevance to the payoff rather than variation. The strong relevance of an axis u is defined as $\mathbb{E}[(u \cdot Z)^2]$, where Z is the path-wise estimator of the vector of price sensitivities [5]. The procedure performs the eigenvalue decomposition of the noncentral covariance matrix of input path-wise differentials to eliminate axes that have negligible influence on the payoff.

3. Problem Description and Implementation

3.1. Problem Description

Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered probability space with the filtration $\mathbb{F} = \{\mathcal{F}_t\}_{t\geq 0}$. In derivatives pricing applications, the objective is to find an unbiased estimator of the initial value of the option, defined as:

$$V_0(\theta) = \mathbb{E}_\theta^{\mathbb{Q}}[f_\theta(Y)|\mathcal{F}_0], \tag{1}$$

where:

- $Y \in \mathbb{R}^d$ is a random vector representing the underlying asset path or terminal value;
- $\mathbb{Q}$ denotes the risk-neutral measure;
- $f_\theta : \mathbb{R}^d \to \mathbb{R}$ is the discounted payoff function;
- $\theta \in \Theta \subseteq \mathbb{R}^d$ is the parameter vector (e.g., volatility, strike, maturity) from which both f and $\mathbb{Q}$ may depend.

For notational simplicity, the conditional expectation $\mathbb{E}_\theta^{\mathbb{Q}}[\cdot|\mathcal{F}_0]$ is denoted as $\mathbb{E}^{\mathbb{Q}}[\cdot]$ in the following sections.
In practice, approximating the expectation in Equation (1) via standard Monte Carlo requires evaluating the payoff $f_\theta(Y_i)$ over M independent paths, yielding the standard estimator $\hat{V}_0(\theta) = \frac{1}{M}\sum_{i=1}^{M} f_\theta(Y_i)$. As previously discussed, this approach becomes computationally unfeasible in nested setups. To address this bottleneck, the Prediction-Enhanced Monte Carlo framework [6] produces a machine learning-based control variate for $f_\theta(Y)$. The PEMC estimator is of the form:

$$\begin{aligned}\mathrm{PEMC}(\theta) :=& \frac{1}{n}\sum_{i=1}^{n}\big(f_\theta(Y_i) - g(\theta, X_i)\big) \\ &+ \frac{1}{N}\sum_{j=1}^{N} g(\theta, \tilde{X}_j)\end{aligned} \tag{2}$$

where $N = 10n$, $X = X(\theta)$ is a variable that depends on $Y = Y(\theta)$ and $g(\theta, x)$ is the estimate of f_θ obtained by the neural network. Moreover, each pair (X_i, Y_i) is simulated together, whereas each $\tilde{X}_j$ has the same marginal distribution as X and is simulated independently from all (X_i, Y_i). In particular, X is chosen as the sum of Brownian increments:

$$X := \sum_{i=1}^{T/\Delta t} \Delta W_{i\Delta t},$$

where $\Delta W_{i\Delta t} \sim N(0, \Delta t), \forall i = 1, ..., \frac{T}{\Delta t}$. This choice is a consequence of the fact that X is a low-dimensional transformation of Y, it is a reasonable predictor for $f_\theta(Y)$ in the case of stochastic processes with discretized dynamics that contain Brownian incremements and its Gaussian distribution allows cheap and parallelizable simulations [6].
Like the control variates method, the PEMC estimator offers two primary statistical advantages over standard Monte Carlo simulations: it remains unbiased and provides significant variance reduction. A comprehensive analysis regarding unbiasedness, variance reduction capabilities and cost effectiveness of the PEMC framework relative to Monte Carlo methods is provided in [6].

3.2. PEMC Implementation

Data Generation

The core of the implementation relies on an "on-the-fly" data generation process, which exploits lazy evaluation to generate small batches of training samples that are immediately discarded after use. This mechanism eliminates the storage overhead associated with the large volumes of data required for the PEMC approach. On the other hand, the two validation datasets, one for early-stopping and the other for hyperparameter tuning, are generated and stored in their entirety, a choice justified by their reduced scale (approximately 10% of N_{train}) which allows them to fit within system memory.

Neural Network Training

The neural network is trained to minimize the Mean Squared Error (MSE), as its mathematical minimizer corresponds to the conditional expectation $\mathbb{E}^{\mathbb{Q}}[f_\theta(Y(\theta))|\theta, X(\theta)]$. Furthermore, the early-stopping mechanism is guided by a modified Mean Absolute Relative Error (MARE):

$$\frac{\left|\frac{1}{N}\sum_{i=1}^{N} g(\theta_i, X(\theta_i)) - \frac{1}{N}\sum_{i=1}^{N} f(Y(\theta_i))\right|}{\left|\frac{1}{N}\sum_{i=1}^{N} f(Y(\theta_i))\right|}.$$

This metric is specifically chosen for its ability to measure the alignment between $\mathbb{E}^{\mathbb{Q}}[g]$ and the true target $\mathbb{E}^{\mathbb{Q}}[f]$, prioritizing the minimization of systematic bias over the fitting of statistical noise.

Computing the PEMC estimator

Finally, the PEMC estimator is evaluated online to ensure real-time pricing capabilities with Equation (2). Its performance is quantified through RMSE across multiple independent experiments; due to the unbiased nature of the PEMC approach, this metric serves as a direct indicator of variance reduction compared to standard Monte Carlo and control variate methods. The ground truth value for the RMSE calculation is computed using a large number of Monte Carlo simulations ($5 \cdot 10^7$ or $2 \cdot 10^9$).

4. Applications and Results

The PEMC approach is applied to five realistic scenarios in order to show its practical effectiveness and versatility.

PEMC Variance Reduction

The initial application addresses arithmetic Asian call options under the Black&Scholes model, where the PEMC approach is compared to standard Monte Carlo and control variates techniques. To evaluate the impact of feature engineering, two versions of the estimator are considered, utilizing either a 1-dimensional or a 14-dimensional vector X. The prediction model is trained on $N_{train} = 1.28 \cdot 10^6$ samples and it employs a dual-branch architecture: the "θ Network Branch", which processes θ through 2 fully connected layers with batch normalization and ReLU activation, each followed by a dropout layer; the "X Network Branch", using a sequence of 2 fully connected layers followed by dropout layers to transform X. The concatenation of the outputs of both branches is fed into the "Combined Network", composed of a

couple of fully connected layers with batch normalization and ReLU activation, each followed by a dropout layer. Finally, a fully connected layer produces the result. A skip connection [3] is used in the "Combined Network" to prevent the vanishing gradient problem.
The PEMC estimators are evaluated through RMSE across 300 experiments with sample size $n \in \{1000, 4000, 9000\}$. Both the PEMC estimator with dim(X)= 1 and the one with dim(X)= 14 achieve variance reduction compared to the Monte Carlo estimator. In particular, the former reduces Monte Carlo's RMSE by $40-43\%$, while the latter attains an impressive $66-69\%$ reduction in RMSE over Monte Carlo. The superior performance of the estimator with a higher-dimensional X stems from the path-dependent nature of Asian option payoffs: a 14-dimensional vector provides higher granularity, allowing the neural network to more accurately capture the underlying asset's temporal trajectory and approximate the payoff. However, the traditional control variate (CV) estimator remains the most effective method, yielding a significantly lower RMSE than the neural network models. This outcome is expected, as the CV approach relies on exact mathematical relationships that provide stronger theoretical guarantees than purely data-driven approximations.

Boost PEMC Variance Reduction

To further enhance performance in scenarios where a control variate is available, the Boost PEMC estimator is introduced by incorporating the control variate's closed-form solution directly into the training label, which is set to $\hat{f}_\theta(\{S_t\}_t) = f_\theta(\{S_t\}_t) - P_G(\theta, \{S_t\}_t)$, i.e., the difference between the arithmetic and the geometric Asian call option payoff, with $\{S_t\}_t$ the underlying asset's path. By training the network on a dataset with $N_{train} = 1.28 \cdot 10^6$ samples and the new label, the model can further reduce the estimator's variance. In particular, the estimator's formula becomes:

$$\text{Boost PEMC}(\theta) := \frac{1}{n}\sum_{i=1}^{n}\left(\hat{f}_\theta(\{S_t^{(i)}\}_t) - g(\theta, X_i(\theta))\right) + \frac{1}{N}\sum_{j=1}^{N} g(\theta, \tilde{X}_j(\theta)) + P_G^{exact}(\theta),$$

where $P_G^{exact}(\theta)$ is the closed-form expected payoff of a geometric Asian option with discrete monitoring under the Black&Scholes model.
This formulation maintains the architecture of the previous experiment while exploiting the fact that the geometric Asian call option is a well-known control variate for its arithmetic counterpart. Thanks to this boost, the PEMC estimator significantly outperforms the control variates one, achieving a $36-43\%$ reduction in RMSE across 300 experiments with sample size $n \in \{1000, 4000, 9000\}$. These results show that the PEMC approach consistently enhances its underlying baseline (whether Monte Carlo or control variates) and proves to be a substantial upgrade when a control variate is already available.

Variance Swaps Under Heston Model

The methodology is extended to the pricing of variance swaps under the Heston model, characterized by the simultaneous modeling of stochastic volatility and asset dynamics through correlated Brownian motions. The dynamics are simulated by Euler scheme with full truncation, and the neural network is designed as a sequence of 2 Multi-Layer Perceptron (MLP) blocks connected through skip connections [3] to mitigate the vanishing gradient problem. The model is trained on $N_{train} = 3 \cdot 10^6$ samples. In this context, the feature vector X is defined by the terminal values of the underlying Brownian motions.
The PEMC estimator is evaluated on 200 independent experiments with sample size $n \in \{1000, 2000, 4000, 6000, 8000, 10000, 20000\}$, achieving a $31-45\%$ RMSE reduction compared to the Monte Carlo estimator, which is similar to the results achieved by the estimator in the Asian option example. This consistency shows the versatility of the PEMC approach: despite the Heston model's increased complexity, the neural network effectively maps the non-linear relationships between parameters and payoff. Crucially, this substantial variance reduction is highly beneficial for computationally expensive frameworks, as it significantly lowers the number of simulated paths required to reach a target accuracy.

Variance Swaps Under Stochastic Local Volatility Model

The use of a Stochastic Local Volatility (SLV) model introduces a higher degree of complexity by requiring the processing of a discretized 2D volatility surface grid. The model's SDEs are:

$$dS_t = rS_t dt + \sigma(S_t, t)e^{v_t} S_t dW_t^S,$$
$$dv_t = \kappa(\eta_t - v_t)dt + \delta dW_t^v,$$

with $\eta_t := -\frac{\delta^2}{2\kappa}(1 + e^{-2\kappa t})$, $\langle dW_t^S, dW_t^v \rangle = \rho dt$ and r, κ, δ, ρ constant parameters. Furthermore, $\sigma(\cdot,\cdot) : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}^+$ is a 2D function representing the local volatility surface and e^{v_t} is a stochastic multiplier such that $e^{v_0} = 1$ and $\mathbb{E}^{\mathbb{Q}}[e^{2v_t}] = 1$. The dynamics are simulated by Euler scheme and X is the vector (W_T^S, W_T^v).
The corresponding neural network architecture integrates a VGG-style convolutional branch for grid-based feature extraction with a fully connected branch for θ and X. The former is composed of two couples of bi-dimensional convolutional layers with ReLU activation, followed by a MaxPool2D operation and a flattening that leads to a fully connected layer; the latter contains a fully connected layer with batch normalization and ReLU activation, followed by a dropout layer and a final fully connected layer. Furthermore, the results of these two branches are concatenated and fed into the "Synthesizer" module, composed of a fully connected layer with ReLU activation, followed by a dropout layer and a fully connected layer.
In this task, the PEMC estimator proves highly effective despite the increased mathematical complexity of the underlying dynamics. The evaluation of the PEMC estimator with the same number of independent experiments and the same set of sample sizes as the Heston model example shows a $33 - 51\%$ reduction in root mean squared error compared to the standard Monte Carlo baseline. As highlighted by these results, the approach demonstrates a slight performance improvement and confirms its versatility, even when handling full volatility surfaces.

Swaptions Under Heath-Jarrow-Morton Model

In the most computationally demanding application, swaption pricing is addressed under the Heath-Jarrow-Morton (HJM) framework, necessitating the discretization of forward rate curves and volatility structures across both time and maturity axes. In this context, the 2-dimensional vector X is constructed by summing the Brownian increments over the first and second halves of the simulated time horizon leading up to the swaption's maturity.
The architecture evolves into a sophisticated three-branch system, featuring specialized 2D and 1D function encoders alongside a vector feature branch to process the forward rate dynamics and volatility grids. The model comprises the following components:

- 2D Function Encoder: this branch processes the 2D volatility grid. It utilizes a CNN architecture with two 2D convolutional layers, each with batch normalization and ReLU activation. An average pooling layer and a subsequent flattening operation conclude this branch, producing an embedding.
- 1D Function Encoder: it handles the 1D forward rate grid using a 1D equivalent of the convolutional architecture described above to produce a corresponding embedding.
- Vector Feature Branch: the third branch is dedicated to the parameters $(\theta, X(\theta))$, processing them through two fully connected layers with batch normalization and ReLU activations. To facilitate gradient flow, it incorporates a single-layer skip connection [3].
- Synthesizer Module: the three generated embeddings are concatenated with the original input $(\theta, X(\theta))$ and fed into the "Synthesizer", a sequence of two fully connected layers with batch normalization, ReLU, and single-layer skip connections. Finally, a fully connected layer outputs the model's prediction.

In this final application, the RMSEs of both the Monte Carlo and the PEMC method are evaluated on 200 independent experiments with $n \in \{1000, 3000, 5000, 7000, 9000, 11000\}$. Even though the HJM framework introduces an additional layer of complexity by requiring the simultaneous management of both volatility structures and forward rate curves, the PEMC estimator certifies its high level of performance. By delivering a $45 - 50\%$ reduction in root mean squared error compared to the Monte Carlo

baseline, the method confirms the efficiency of integrating 1D convolutional layers for sequential data with the proven effectiveness of 2D convolutional layers for grid-structured data. In addition, these results provide further evidence of the versatility and robust variance reduction of the PEMC approach [6].

5. Extensions

A primary limitation of the PEMC method is its theoretical requirement of a large training dataset (N_{train}) to guarantee variance reduction compared to the Monte Carlo estimator, which leads to computationally expensive training procedures. As established in [6], in an ideal scenario where the predictor g perfectly matches the conditional expectation $\mathbb{E}^{\mathbb{Q}}[f_\theta(Y)|\theta, X]$, the variance ratio between PEMC and Monte Carlo is approximated by $r(\rho, c)$, which depends on the correlation $\rho = corr(f, g)$ and the relative computational cost $c = \frac{c_g}{c_f}$. In practice, however, g is only an approximation, introducing an error ϵ_{total} bounded by:

$$\epsilon_{total} \leq 2\epsilon_a^{\mathcal{G}} + 2\epsilon_e^{N_{train}}.$$

While the approximation error $\epsilon_a^{\mathcal{G}}$ can be mitigated by employing sufficiently complex neural network architectures, the statistical error $\epsilon_e^{N_{train}}$ typically decreases at a rate of $O(1/\sqrt{N_{train}})$. Consequently, a massive dataset is generally required to minimize ϵ_{total} and drive the variance ratio toward its optimal value.
Principal Component Analysis (PCA) combined with differential PCA [4] can be utilized as a preprocessing tool to decrease N_{train} while maintaining optimal variance reduction. By transforming the neural network's input into lower-dimensional, noise-free data, this technique reduces the overall complexity of the problem, lowering the statistical error for a fixed N_{train}. This dimensional reduction provides advantages on two distinct levels:

- It eliminates the need to manually define the feature vector X, replacing it with a systematically derived quantity.
- By reducing the dimensional complexity of the problem, the statistical error is lowered for any given N_{train}. This allows the model to achieve the desired variance reduction using a significantly smaller training set. The resulting decrease in training time enables more frequent model updates with narrower parameter intervals, allowing the pricing tool to adapt easily to evolving market conditions.

The subsequent sections demonstrate the practical implementation and numerical validation of this preprocessing mechanism across three specific real-world pricing scenarios selected among the five employed for the standard PEMC estimator.

Differential PEMC Variance Reduction

The pricing of an arithmetic Asian call option under the Black&Scholes model serves as the first application case of the preprocessing pipeline. Two versions of the data preparation procedure are employed to evaluate the estimator: the "split" preprocessing applies the dimensionality reduction method to the market parameters θ and the Brownian increments separately, feeding the neural network described in the PEMC variance reduction example with the resulting features; conversely, the "full" preprocessing concatenates θ and the Brownian increments into a single matrix, applying the data preparation algorithm to the concatenated input to feed the "Combined Network" branch of the standard PEMC method's application to the Asian options.
The benefits of performing differential PCA on top of standard PCA are significant: the input complexity is reduced to just 2 or 3 principal components that explain at least 99.9% of the total relevance. Thanks to this optimization, both methodologies can employ a training dataset of just $N_{train} = 128$ samples.

Method	RMSE Reduction
PEMC (dim(X) = 1)	40-43%
PEMC (dim(X) = 14)	66-69%
Diff. PEMC ("split")	66-71%
Diff. PEMC ("full")	59-64%

Table 1: RMSE reduction over Monte Carlo: comparison between standard PEMC and Differential PEMC in the Asian option example

As shown in Table 1, the Differential PEMC estimator with "split" preprocessing matches the performance of the standard PEMC one trained on the original $1.28 \cdot 10^6$ samples, while the

"full" preprocessing also delivers strong results. Its slightly worse performance compared to the "split" version is coherent with the fact that the relationship between θ and the Brownian increments in the payoff function is multiplicative and exponential. Because the "full" preprocessing feeds the network with linear combinations of these variables, the model must isolate their individual contributions to learn the true relationship, a task simplified by the "split" version, which maintains a clear distinction between the market parameters and the simulated path.
The superiority of the Differential PEMC method becomes particularly evident when standard PEMC is trained on the same reduced dataset of 128 samples; in that scenario, the standard estimator's best performance is a $36-44\%$ variance reduction compared to Monte Carlo. Even though this result is worse than the ones obtained by the Differential PEMC estimator with the same number of training samples, it is still interesting to notice that with just 128 training samples the standard PEMC method outperformed Monte Carlo by a significant amount. This result is additional confirmation of the standard PEMC approach's power in rather simple scenarios.
Beyond accuracy, the most direct advantage of operating with a drastically reduced N_{train} is a significantly shorter training time than the standard PEMC with $1.28 \cdot 10^6$ training samples, highlighted by Table 2.

Method	**Training time [s/epoch]**
PEMC (dim(X) = 14)	3.84
Diff. PEMC ("split")	0.046
Diff. PEMC ("full")	0.016

Table 2: Comparison of average training time per epoch on a similar number of epochs: standard PEMC and Differential PEMC in the Asian option example

To sum up, even though the "full" methodology's training time per epoch is lower than its "split" counterpart due to a lighter architecture, they are both much faster than the standard PEMC method. Since the primary focus of the approach is variance reduction, the "split" procedure is preferable, as it achieves slightly better results in this sense.

Differential Boost PEMC Variance Reduction

As observed in the previous section, since the geometric Asian call option is a well-known control variate for its arithmetic counterpart, the Differential Boost PEMC can be built upon it. In particular, both "split" and "full" preprocessing strategies are implemented identically to the previous real-world scenario, the only difference is in the training label and the estimator formula, as defined in the standard Boost PEMC framework.
In the "split" preprocessing case, the data preparation procedure extracts 2 principal components that account for at least 99% of the total relevance of θ and the Brownian increments. On the other hand, the "full" preprocessing approach identifies 5 principal components as the ones which capture 99% of the data's relevance. Using a training dataset composed of just $N_{train} = 1.28 \cdot 10^4$ samples, the results in Table 3 are obtained.

Method	**RMSE Reduction**
Boost PEMC	36-43%
Diff. Boost PEMC ("split")	52-56%
Diff. Boost PEMC ("full")	58-60%

Table 3: RMSE reduction over control variates estimator: comparison between standard Boost PEMC and differential Boost PEMC in the Asian option example

The results of the differential Boost PEMC ("split") estimator in Table 3 represent a substantial improvement over the performance of the standard Boost PEMC, trained on a dataset of $1.28 \cdot 10^6$ samples, in terms of variance reduction over the geometric control variate baseline. However, its "full" counterpart achieves slightly better results. This behavior is opposite to the Differential PEMC's one, but this is likely related to the nature of the training label: since the geometric control variate already accounts for the dependencies from θ and the Brownian increments alone, the training label is likely driven by complex interactions between the two. The "full" method captures these interactions

more effectively than the "split" thanks to the fact that it applies the data preparation algorithm to the concatenation of θ and the Brownian increments. Furthermore, both versions of the differential Boost PEMC estimator outperform the standard Boost PEMC trained on a reduced dataset ($1.28 \cdot 10^4$ samples), which only delivers a $11 - 21\%$ variance reduction.

Method	Training time [s/epoch]
Boost PEMC	2.33
Diff. Boost PEMC ("split")	0.64
Diff. Boost PEMC ("full")	0.36

Table 4: Comparison of average training time per epoch on a similar number of epochs: standard Boost PEMC and differential Boost PEMC in the Asian option example

The reduction in training data directly translates to enhanced computational efficiency, significantly dropping the average training time per epoch compared to the standard Boost PEMC (trained on the full dataset) for a similar number of training epochs, as detailed in Table 4. In particular, the "full" method further accelerates the training process compared to the "split" one, requiring even less time per epoch. This speedup, which results in a reduction by a factor of roughly 6.5 compared to the standard model, is due to the lighter architecture employed by the "full" approach.
To conclude, the "full" preprocessing methodology is preferable for the Differential Boost PEMC estimator. Unlike the previous application, the "full" method here delivers both superior variance reduction and a lower training time per epoch, making it the optimal choice.

Differential PEMC for Swaptions Under Heath-Jarrow-Morton Model

To rigorously validate the Differential PEMC approach, it is applied to the most challenging real-world scenario considered in this work: pricing swaptions under the Heath-Jarrow-Morton (HJM) framework. Since this model involves both volatility structures and forward rate curves, the "split" and "full" preprocessing methods must be adapted from previous examples. In the "split" methodology, the data preparation algorithm is applied to the concatenation of the scalar parameters (θ) and the Brownian increments, feeding these transformed features into the "Vector Features Encoder" which is employed in the HJM scenario of standard PEMC, while the volatility surface and initial forward rates are fed directly into their respective CNN branches. Conversely, the "full" preprocessing approach flattens the 2D volatility surface and the 1D initial forward rate grid, concatenates them with θ and the Brownian increments and applies the dimensional reduction technique to the concatenated matrix before feeding it to the final "Synthesizer" branch.
The implementation of the "split" preprocessing identifies 5 principal components that explain over 99.99% of the relevance of θ and the Brownian increments. On the other hand, the "full" preprocessing approach extracts 4 principal components to capture 99.9% of the total relevance across all flattened inputs. The network is trained on a dataset of $N_{train} = 3 \cdot 10^4$ samples, against the $3 \cdot 10^6$ of the standard PEMC.

Method	RMSE Reduction
PEMC	45-50%
Diff. PEMC ("split")	47-52%
Diff. PEMC ("full")	3-13%

Table 5: RMSE reduction over Monte Carlo: comparison between standard PEMC and Differential PEMC in the HJM example

Despite the reduced number of training samples, the Differential PEMC ("split") estimator achieves very similar results to standard PEMC in terms of RMSE reduction over Monte Carlo (see Table 5). The true power of the preprocessing pipeline becomes more evident when the standard PEMC model is trained on the reduced dataset of $3 \cdot 10^4$ samples: under these conditions, standard PEMC fails, yielding an RMSE roughly equivalent to Monte Carlo. This performance is likely a consequence of the combination of a high statistical error and the high complexity of the scenario. However, the fact that the introduction of the data preparation algorithm transformed the model into an advantage compared to Monte Carlo is remarkable. Table 5 also shows that the results of the "full" preprocessing estimator are significantly worse than those

achieved by the "split" method. This difference highlights a crucial point: by keeping the inputs separated in the "split" approach, the CNN branches can efficiently extract spatial and sequential patterns. On the other hand, the "full" approach forces the network to retrieve the same complex information from a single transformed vector.
Furthermore, because the "full" approach applies PCA to large grids, its computational overhead is higher than that of the "split" method, resulting in a training time of 9.39 seconds per epoch (roughly four times slower than the "split" method). However, as highlighted in Table 6, both the differential estimators save a significant amount of training time compared to the standard PEMC implementation.

Method	**Training time [s/epoch]**
PEMC	289.89
Diff. PEMC ("split")	2.09
Diff. PEMC ("full")	9.39

Table 6: Comparison of average training time per epoch on a similar number of epochs: standard PEMC and Differential PEMC in the HJM example

Summing up, the "split" approach emerges as the superior methodology within the HJM framework, offering higher variance reduction and faster training times.

6. Conclusions

This thesis optimized the Prediction-Enhanced Monte Carlo (PEMC) framework by introducing the Differential PEMC method, where a preprocessing pipeline leveraging PCA and differential PCA is applied to the PEMC approach. By isolating features based on their relevance to the target label, the data preparation technique extract few very informative features, allowing for the use of significantly smaller training datasets, thereby accelerating training times without compromising the estimator's variance reduction capabilities. Testing two versions of the method in three realistic scenarios revealed that both the Differential PEMC estimators outperformed the standard PEMC one. Furthermore, a "full" preprocessing configuration (concatenating features prior to transformation) is optimal to enhance the performance of an existing control variate, whereas a "split" configuration (processing distinct feature groups separately) is superior in the absence of one.

References

[1] Mike Giles and Paul Glasserman. Smoking adjoints: Fast monte carlo greeks. *Risk*, 19(1):88–92, 2006.

[2] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer, 2013.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Brian Huge and Antoine Savine. Differential machine learning. *arXiv preprint arXiv:2005.02347*, 2020.

[5] Brian Huge and Antoine Savine. Axes that matter: Pca with a difference. *arXiv preprint arXiv:2503.06707*, 2025.

[6] Fengpei Li, Haoxian Chen, Jiahe Lin, Arkin Gupta, Xiaowei Tan, Honglei Zhao, Gang Xu, Yuriy Nevmyvaka, Agostino Capponi, and Henry Lam. Prediction-enhanced monte carlo: A machine learning view on control variate, 2025.