Martingale Posterior Deep Neural Networks

Med et al.

January 27, 2026

1 Bayesian Linear Regression Problem

1.1 Regression problem

The regression problem aims to understand the relationship between a response variable Y and a set of predictor variables $\boldsymbol{X} = (X_1, \ldots, X_p)$. Typically, this relationship is modeled as:

$$Y = f(\boldsymbol{X}) + \epsilon$$

where f is a deterministic function and ϵ represents a random error component with zero mean. The predictor variables $\boldsymbol{X}$ are assumed to be observed without error.

The main goal is to estimate f, which represents the conditional expectation of Y given $\boldsymbol{X} = \boldsymbol{x}$, i.e.,

$$f(\boldsymbol{x}) = E(Y \mid \boldsymbol{X} = \boldsymbol{x}).$$

This function captures the true relationship between the response Y and the predictors $\boldsymbol{X}$.

Since the exact form of f is unknown, it must be approximated using observed data:

$$\mathcal{D} = \{(y_i, \boldsymbol{x}_i)\}_{i=1}^n,$$

where y_i are observed responses and $\boldsymbol{x}_i$ are corresponding predictor values.

1.2 Basis function representation in Regression

The linear regression model is given by:

$$g(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

More generally, it can be expressed as:

$$g(x) = \sum_{i=1}^{k} \beta_i B_i(x), \quad x \in X,$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)'$ are the coefficients corresponding to the basis functions $B = (B_1, \ldots, B_k)$.

The standard linear model is a special case of this framework, where $k = p + 1$, with basis functions defined as $B_1(\boldsymbol{x}) = 1$ and $B_i(\boldsymbol{x}) = x_{i-1}$ for $i = 2, \ldots, p + 1$. For now, we can simply view each basis function B_i as a mapping from the predictor space X to the real line $\mathbb{R}$.

1.3 Bayesian model

The Bayesian approach to model fitting and inference follows three key steps:

1. Assign prior distributions to all unknown parameters.
2. Specify the likelihood function of the data given the parameters.
3. Use Bayes' Theorem to compute the posterior distribution of the parameters given the data.

When approximating f by g, the model is given by:

$$y_i = \sum_{i=1}^{k} \beta_i B_i(x_i) + \epsilon_i, \quad i = 1, \ldots, n,$$

which, in matrix notation, can be written as:

$$\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where:

- $\boldsymbol{Y} = (y_1, \ldots, y_n)'$ is the response vector,
- $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)'$ represents the error terms,
- $\boldsymbol{B}$ is the design matrix of the regression, given by:

$$\boldsymbol{B} = \begin{bmatrix} B_1(\boldsymbol{x}_1) & \cdots & B_k(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ B_1(\boldsymbol{x}_n) & \cdots & B_k(\boldsymbol{x}_n) \end{bmatrix}.$$

1.3.1 The Likelihood

The likelihood function of a model, up to proportionality, is the joint probability of observing the data as a function of the model parameters. For a linear model with a fixed design matrix $\boldsymbol{B}$, this is written as:

$$p(\mathcal{D} \mid \boldsymbol{\beta}, \sigma^2) \quad \text{or} \quad p(Y \mid \boldsymbol{X}, \boldsymbol{\beta}, \sigma^2).$$

Assuming normally distributed errors, $\boldsymbol{\epsilon} \sim N(0, \sigma^2 I)$, where I is the $n \times n$ identity matrix, we express the errors as:

$$\boldsymbol{\epsilon} = \boldsymbol{Y} - \boldsymbol{B}\boldsymbol{\beta}.$$

Thus, the likelihood follows a multivariate normal distribution:

$$p(\mathcal{D} \mid \boldsymbol{\beta}, \sigma^2) = N(\boldsymbol{B}\boldsymbol{\beta}, \sigma^2 I),$$

which expands to:

$$p(\mathcal{D} \mid \boldsymbol{\beta}, \sigma^2) = (2\pi\sigma^2)^{-n/2} \exp\left\{-\frac{(\boldsymbol{Y} - \boldsymbol{B}\boldsymbol{\beta})'(\boldsymbol{Y} - \boldsymbol{B}\boldsymbol{\beta})}{2\sigma^2}\right\}.$$

1.4 The posterior

By Bayes' Theorem, the posterior distribution of the model parameters is given by:

$$p(\boldsymbol{\beta}, \sigma^2 \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\beta}, \sigma^2)p(\boldsymbol{\beta}, \sigma^2)}{p(\mathcal{D})}.$$

Since the denominator $p(\mathcal{D})$ is a normalizing constant independent of $\boldsymbol{\beta}$ and σ^2, the posterior distribution can be determined up to proportionality by multiplying the likelihood and prior. Using the identity:

$$\begin{aligned}(\boldsymbol{Y} - \boldsymbol{B}\boldsymbol{\beta})'(\boldsymbol{Y} - \boldsymbol{B}\boldsymbol{\beta}) + (\boldsymbol{\beta} - \boldsymbol{m})'\boldsymbol{V}^{-1}(\boldsymbol{\beta} - \boldsymbol{m}) + 2b \\ \equiv (\boldsymbol{\beta} - \boldsymbol{m}^*)'(\boldsymbol{V}^*)^{-1}(\boldsymbol{\beta} - \boldsymbol{m}^*) + 2b^*,\end{aligned}$$

the posterior distribution takes the form:

$$p(\boldsymbol{\beta}, \sigma^2 \mid \mathcal{D}) \propto (\sigma^2)^{-(a^* + k/2 + 1)} \exp\left\{-\frac{(\boldsymbol{\beta} - \boldsymbol{m}^*)'(\boldsymbol{V}^*)^{-1}(\boldsymbol{\beta} - \boldsymbol{m}^*) + 2b^*}{2\sigma^2}\right\}.$$

The posterior parameters are given by:

$$\begin{aligned}\boldsymbol{m}^* &= (\boldsymbol{V}^{-1} + \boldsymbol{B}'\boldsymbol{B})^{-1}(\boldsymbol{V}^{-1}\boldsymbol{m} + \boldsymbol{B}'\boldsymbol{Y}), \\ \boldsymbol{V}^* &= (\boldsymbol{V}^{-1} + \boldsymbol{B}'\boldsymbol{B})^{-1}, \\ a^* &= a + \frac{n}{2}, \\ b^* &= b + \frac{\boldsymbol{m}'\boldsymbol{V}^{-1}\boldsymbol{m} + \boldsymbol{Y}'\boldsymbol{Y} - (\boldsymbol{m}^*)'(\boldsymbol{V}^*)^{-1}\boldsymbol{m}^*}{2}.\end{aligned}$$

The posterior distribution in (2.16) retains the same functional form as the prior in (2.12), with updated parameters. Thus,

$$p(\boldsymbol{\beta}, \sigma^2 \mid \mathcal{D}) = \text{NIG}(\boldsymbol{m}^*, \boldsymbol{V}^*, a^*, b^*),$$

where the normalizing constant, found by comparison with (2.12), is

$$\frac{(b^*)^{a^*}}{(2\pi)^{k/2}|\boldsymbol{V}^*|^{1/2}\Gamma(a^*)}.$$

This standard Bayesian updating result for the normal-inverse-gamma (NIG) model is well-documented in Bayesian statistics literature (e.g., Bernardo & Smith, 1994; Broemeling, 1985; O'Hagan, 1994). It provides a straightforward way to obtain the posterior distribution of $\boldsymbol{\beta}$ and σ^2.

The updated parameters have intuitive interpretations:

- $\boldsymbol{m}^*$ represents the posterior mean estimate of $\boldsymbol{\beta}$, with its variance governed by $\boldsymbol{V}^*$.
- The term $\boldsymbol{Y}'\boldsymbol{Y} - (\boldsymbol{m}^*)'(\boldsymbol{V}^*)^{-1}\boldsymbol{m}^*$ corresponds to the sum of squared residuals when using the posterior mean.
- In the limiting case where $a, b \to 0$ and $V \to 0$, the ratio b^*/a^* approximates the residual sum of squares.

2 Example

Consider a dataset $\{(x_i, y_i)\}_{i=1}^n$ where x_i are the predictors and y_i are the responses. The linear regression model assumes:

$$y_i = \theta^T x_i + \epsilon_i,$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ are independent and identically distributed Gaussian noise terms.

2.1 Likelihood Function

The likelihood function for the data given θ and σ^2 is:

$$L(\theta, \sigma^2 | \{x_i, y_i\}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}\right).$$

Taking the natural logarithm, we obtain the log-likelihood function:

$$\log L(\theta, \sigma^2 | \{x_i, y_i\}) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta^T x_i)^2.$$

2.2 Maximum Likelihood Estimation

To estimate θ, we maximize the log-likelihood. The maximum likelihood estimate (MLE) of θ is obtained by minimizing the sum of squared residuals:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n}(y_i - \theta^T x_i)^2.$$

This leads to the well-known Ordinary Least Squares (OLS) solution:

$$\hat{\theta} = (X^T X)^{-1} X^T y,$$

where X is the design matrix with rows x_i^T and y is the vector of observed responses.

The MLE of σ^2 is given by:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\theta}^T x_i)^2.$$

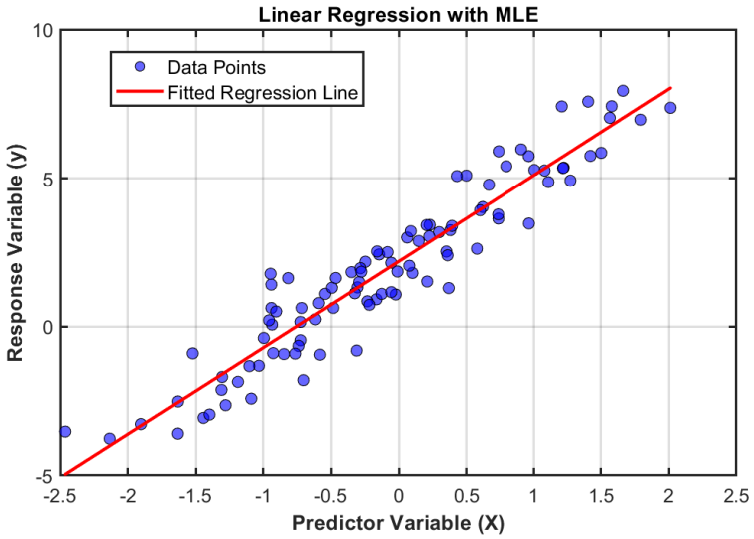

Figure 1: MLE for Linear Regression.

3 Martingale Posterior for LR

3.1 Model Formulation

We consider the standard linear regression model:

$$y_n = X_n\beta + \epsilon_n, \quad \epsilon_n \sim N(0, \sigma^2), \tag{1}$$

where:

- $y_n \in \mathbb{R}$ is the response variable at time n.
- $X_n \in \mathbb{R}^{1\times p}$ is the design matrix, assumed to be a Gaussian random vector.
- $\beta \in \mathbb{R}^p$ is the unknown regression coefficient.
- $\epsilon_n \sim N(0, \sigma^2)$ is the noise term with variance σ^2.

The design matrix follows:

$$X_n \sim N(\mu_X, \Sigma_X), \tag{2}$$

where:

- $\mu_X \in \mathbb{R}^p$ is the unknown mean vector of the design matrix.
- $\Sigma_X \in \mathbb{R}^{p\times p}$ is the unknown covariance matrix of the design matrix.

We assume that we estimate μ_X and Σ_X sequentially as new observations arrive.

3.2 Score Function Computation

The likelihood function for a single observation is given by:

$$p(y_n|X_n, \beta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - X_n\beta)^2}{2\sigma^2}\right). \tag{3}$$

Taking the logarithm, we obtain the log-likelihood:

$$\log p(y_n|X_n, \beta, \sigma^2) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y_n - X_n\beta)^2}{2\sigma^2}. \tag{4}$$

The score functions are computed as follows:

Score Function for β: Taking the derivative with respect to β:

$$\frac{\partial}{\partial\beta}\log p(y_n|X_n, \beta, \sigma^2) = \frac{1}{\sigma^2}X_n^\top(y_n - X_n\beta). \tag{5}$$

Thus, the score function for β is:

$$s(y_n, X_n, \beta) = X_n^\top(y_n - X_n\beta).$$

Score Function for σ^2: Taking the derivative with respect to σ^2:

$$\frac{\partial}{\partial\sigma^2}\log p(y_n|X_n, \beta, \sigma^2) = -\frac{1}{2\sigma^2} + \frac{(y_n - X_n\beta)^2}{2\sigma^4}. \tag{6}$$

Thus, the score function for σ^2 is:

$$s(y_n, X_n, \sigma^2) = \frac{(y_n - X_n\beta)^2}{\sigma^2} - 1.$$

Score Function for μ_X: Since $X_n \sim N(\mu_X, \Sigma_X)$, its log-likelihood is:

$$\log p(X_n|\mu_X, \Sigma_X) = -\frac{1}{2}(X_n - \mu_X)^\top\Sigma_X^{-1}(X_n - \mu_X).$$

Taking the derivative with respect to μ_X:

$$s(X_n, \mu_X) = \Sigma_X^{-1}(X_n - \mu_X).$$

Score Function for Σ_X: Taking the derivative with respect to Σ_X:

$$s(X_n, \Sigma_X) = \frac{1}{2}\left[\Sigma_X^{-1}(X_n - \mu_X)(X_n - \mu_X)^\top\Sigma_X^{-1} - \Sigma_X^{-1}\right].$$

3.3 Martingale Posterior Updates

Using the score functions, we define sequential updates with step size $\epsilon_n = 1/(n+1)$:

Update for β:

$$\beta_{n+1} = \beta_n + \epsilon_n X_n^\top (y_n - X_n \beta_n).$$

Update for σ^2:

$$\sigma^2_{n+1} = \sigma^2_n + \epsilon_n \left((y_n - X_n \beta_n)^2 - \sigma^2_n \right).$$

Update for μ_X:

$$\mu_{X,n+1} = \mu_{X,n} + \epsilon_n (X_n - \mu_{X,n}).$$

Update for Σ_X:

$$\Sigma_{X,n+1} = \Sigma_{X,n} + \epsilon_n \left[(X_n - \mu_{X,n})(X_n - \mu_{X,n})^\top - \Sigma_{X,n} \right].$$

4 Numerical Experiments

In this section, we conduct a numerical experiment to evaluate the performance of the martingale posterior updates for linear regression. We simulate a dataset and apply the sequential updating rules derived in the previous section to estimate the unknown parameters.

4.1 Experimental Setup

We consider a linear regression model with $p = 3$ predictors and a total of $n = 1000$ observations. The true parameter values are set as follows:

- True regression coefficients: $\beta^* = [1, -2, 3]^T$.
- True noise variance: $\sigma^{2*} = 1$.
- True mean of the design matrix: $\mu_X^* = [0, 0, 0]^T$.
- True covariance matrix of the design matrix: $\Sigma_X^* = I_p$.

The design matrix X_n is generated from a multivariate normal distribution:

$$X_n \sim N(\mu_X^*, \Sigma_X^*), \tag{7}$$

and the response variable is generated as:

$$y_n = X_n \beta^* + \epsilon_n, \quad \epsilon_n \sim N(0, \sigma^{2*}). \tag{8}$$

4.2 Sequential Estimation Procedure

We initialize the estimates for β, σ^2, μ_X, and Σ_X as follows:

$$\beta_0 = \mathbf{0}, \quad \sigma_0^2 = 1, \quad \mu_{X,0} = \mathbf{0}, \quad \Sigma_{X,0} = I_p. \tag{9}$$

The estimates are updated sequentially using the following recursive formulas:

$$\beta_{n+1} = \beta_n + \epsilon_n X_n^T (y_n - X_n \beta_n), \tag{10}$$

$$\sigma_{n+1}^2 = \sigma_n^2 + \epsilon_n \left((y_n - X_n \beta_n)^2 - \sigma_n^2 \right), \tag{11}$$

$$\mu_{X,n+1} = \mu_{X,n} + \epsilon_n (X_n - \mu_{X,n}), \tag{12}$$

$$\Sigma_{X,n+1} = \Sigma_{X,n} + \epsilon_n \left[(X_n - \mu_{X,n})(X_n - \mu_{X,n})^T - \Sigma_{X,n} \right]. \tag{13}$$

where $\epsilon_n = \frac{1}{n+1}$ is the step size.

4.3 Results

To assess the convergence of the estimates, we track their evolution over iterations. The estimated values of β, σ^2, μ_X, and Σ_X are plotted against the number of observations. Figures 2, 3, and 4 display the estimation history of these parameters.

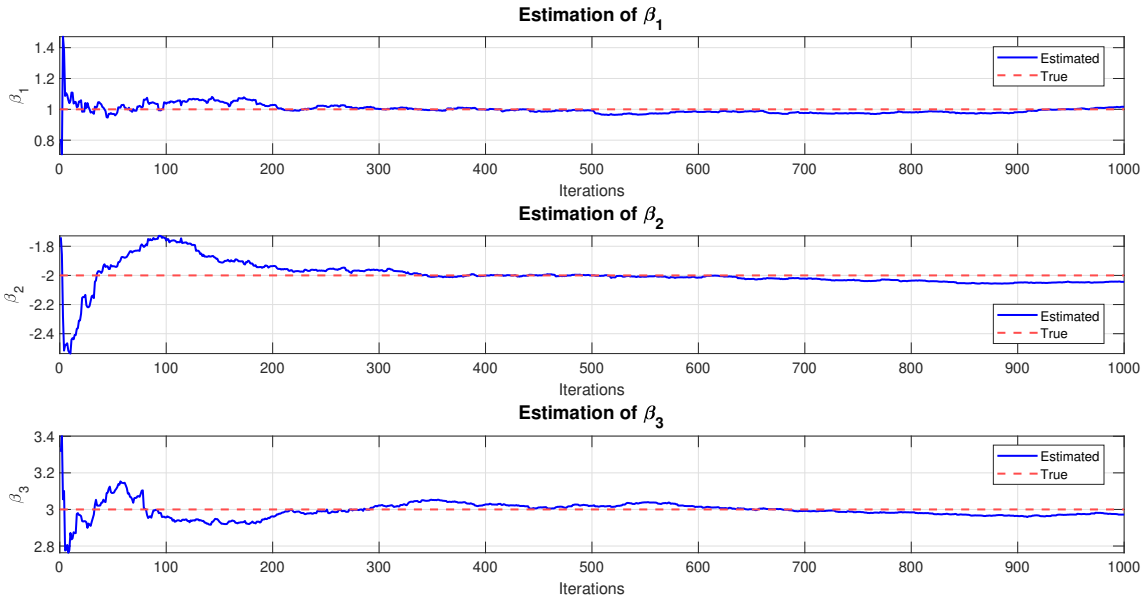

Figure 2: Convergence of β estimates over iterations.

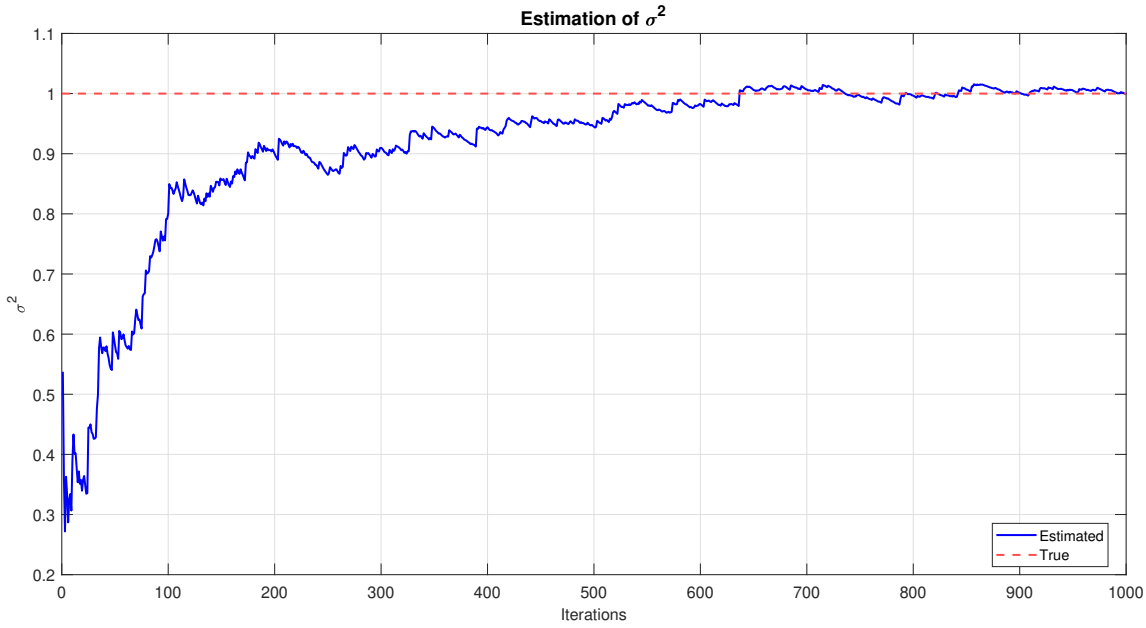

Figure 3: Convergence of σ^2 estimates over iterations.

From the results, we observe that the estimates gradually converge to their true values, validating the effectiveness of the martingale posterior approach for sequential estimation in linear regression.

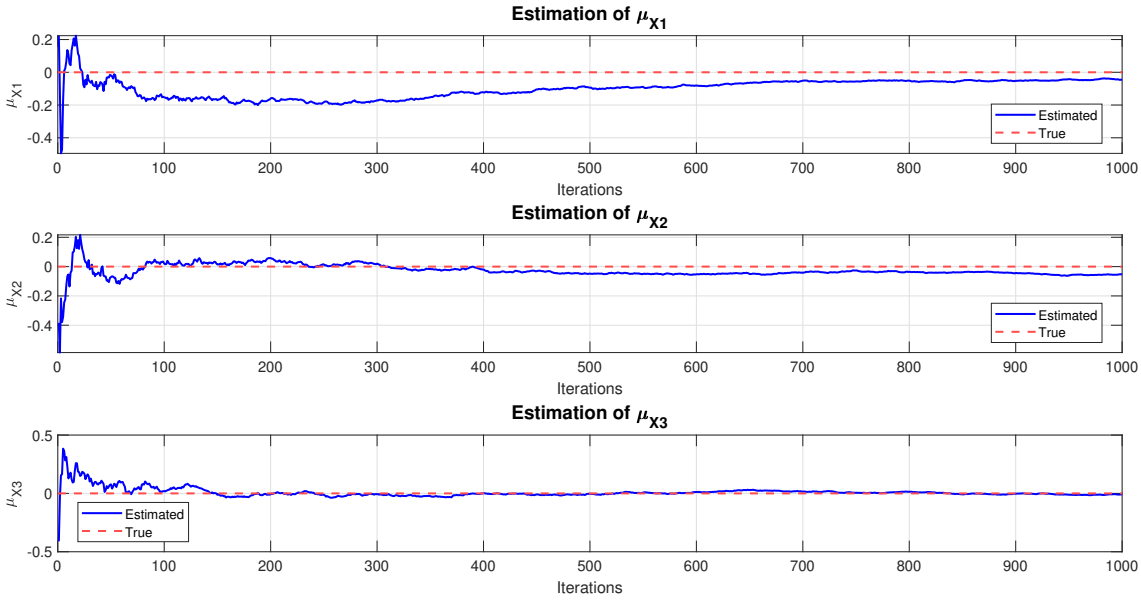

Figure 4: Convergence of μ_X estimates over iterations.

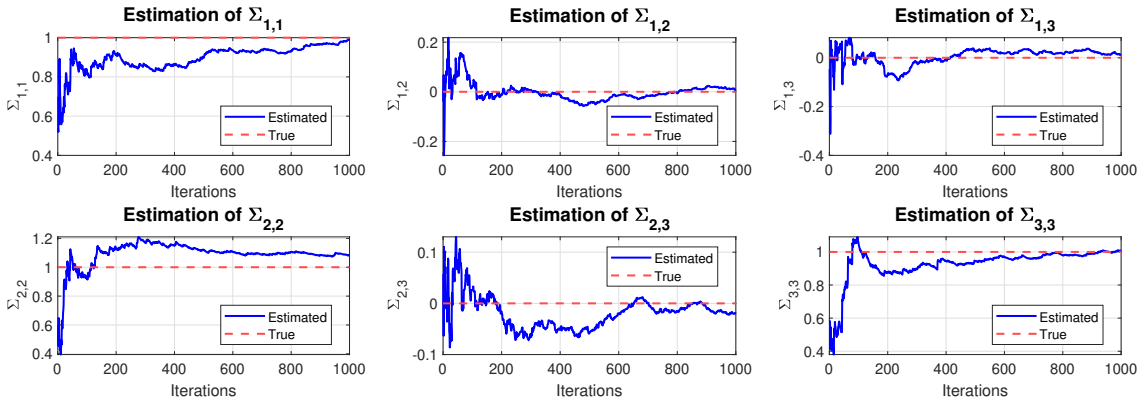

Figure 5: Convergence of μ_X estimates over iterations.

5 Numerical Results with High Dimension

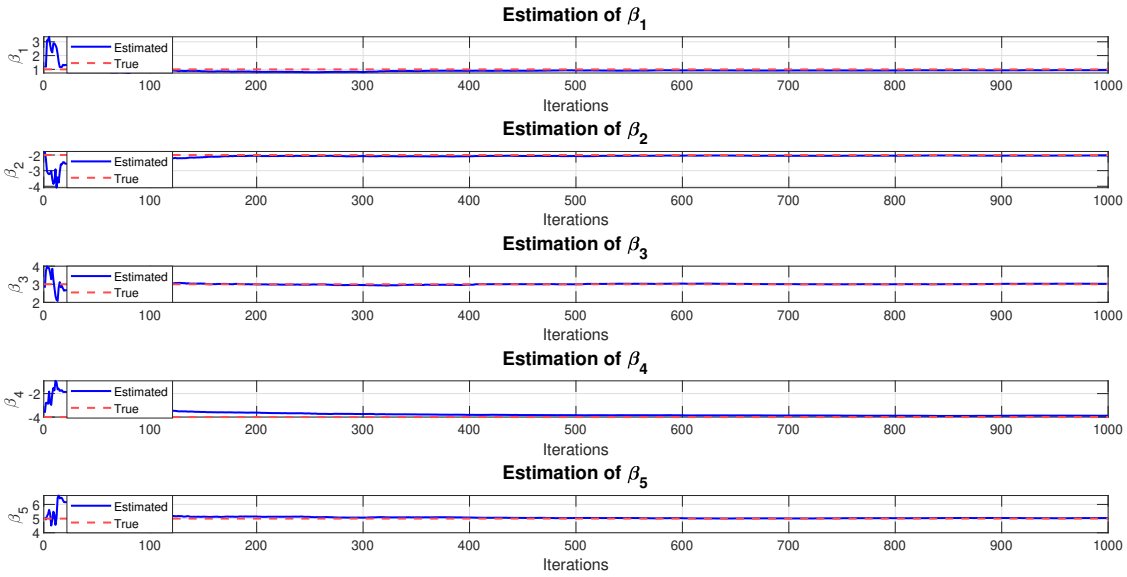

Figure 6: Convergence of β estimates over iterations.

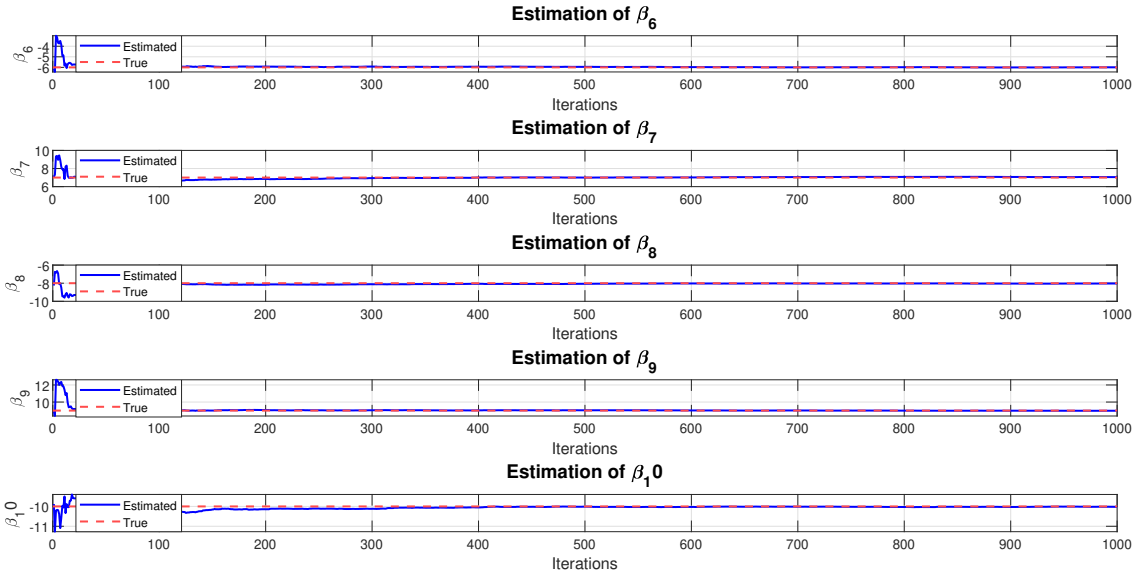

Figure 7: Convergence of β estimates over iterations.

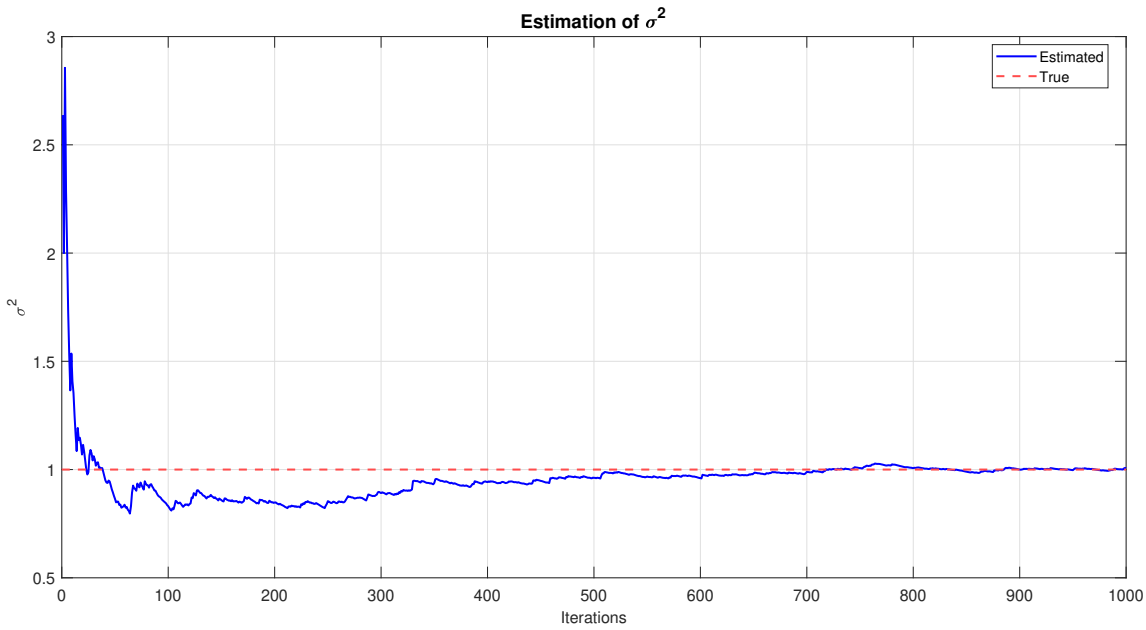

Figure 8: Convergence of σ^2 estimates over iterations.

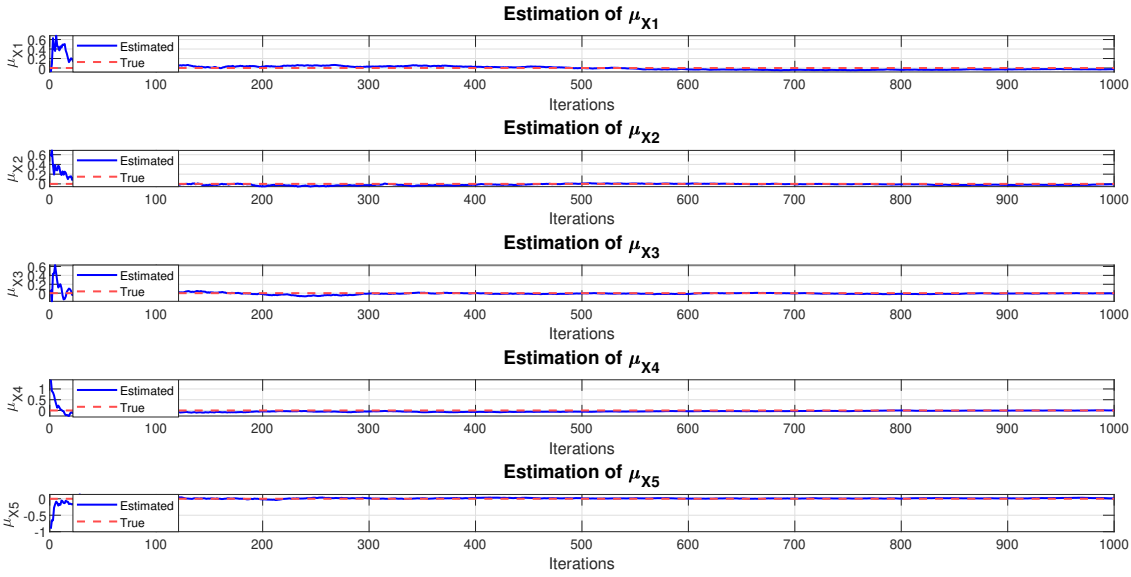

Figure 9: Convergence of μ_X estimates over iterations.

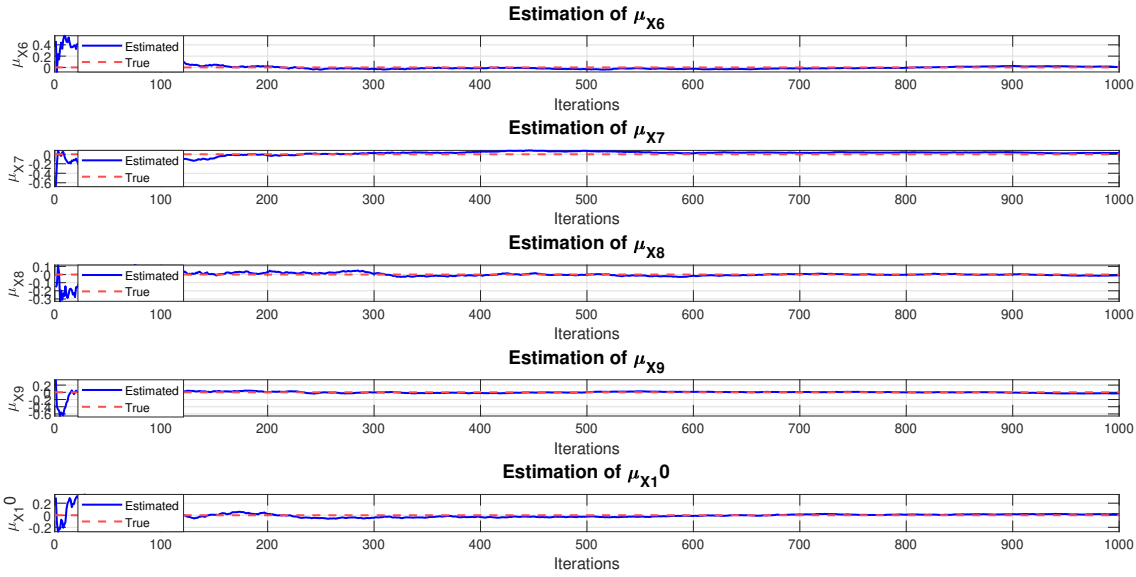

Figure 10: Convergence of μ_X estimates over iterations.

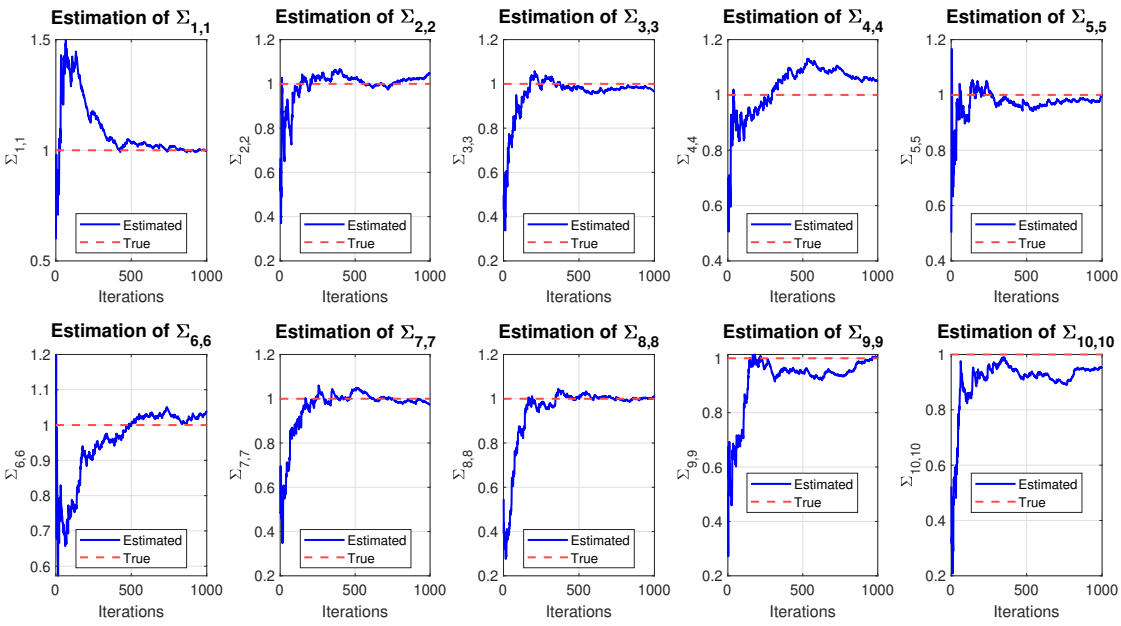

Figure 11: Convergence of Σ estimates over iterations.

6 Comparison of Martingale Posterior Updates and Exact Posterior in Bayesian Linear Regression

In this work, we compare the Martingale Posterior Updates (sequential updates) for the parameters in a Bayesian Linear Regression model with the exact posterior mean and covariance.

We focus on the following parameters:

- β: Regression coefficients (to be estimated),
- σ^2: The noise variance,
- μ_X: Mean of the predictors,
- Σ_X: Covariance matrix of the predictors.

The comparison involves running sequential updates with Martingale Posterior and comparing these estimates with the exact posterior (analytical solution).

6.1 Exact Posterior for Linear Regression

The Bayesian posterior distribution of the parameters in Linear Regression has the following form:

Posterior Mean for β: The exact posterior mean for the regression coefficients β is given by:

$$\hat{\beta}_n = (X^T X + \lambda I)^{-1} X^T y \tag{14}$$

where:

- X is the matrix of predictors,
- y is the vector of observed responses,
- λ is the regularization parameter, which is assumed to be zero for a non-informative prior.

Posterior Covariance for β: The posterior covariance for β is:

$$\Sigma_{\beta|n} = \sigma^2 (X^T X)^{-1} \tag{15}$$

where:

- σ^2 is the noise variance,
- X is the matrix of predictors.

Posterior Variance for σ^2: The posterior variance for the noise variance σ^2 is:

$$\sigma_n^2 = \frac{(y - X\hat{\beta}_n)^T(y - X\hat{\beta}_n)}{n} \tag{16}$$

7 Martingale Posterior Updates

In the Martingale Posterior Updates, we update the parameters sequentially as new data points arrive. The updates are done using the following formulas:

Update for β: The sequential update for β is given by:

$$\beta_n = \beta_{n-1} + \epsilon_n(x_n(y_n - x_n^T\beta_{n-1})) \tag{17}$$

where:

- x_n is the n-th predictor vector,
- y_n is the n-th response value,
- ϵ_n is the adaptive step size.

Update for σ^2: The sequential update for σ^2 is:

$$\sigma_n^2 = \sigma_{n-1}^2 + \epsilon_n((y_n - x_n^T\beta_n)^2 - \sigma_{n-1}^2) \tag{18}$$

Update for μ_X: The sequential update for the mean of the predictors μ_X is:

$$\mu_X = \mu_X + \epsilon_n(x_n - \mu_X) \tag{19}$$

Update for Σ_X: The sequential update for the covariance matrix of the predictors Σ_X is:

$$\Sigma_X = \Sigma_X + \epsilon_n((x_n - \mu_X)(x_n - \mu_X)^T - \Sigma_X) \tag{20}$$

This update enforces symmetry in Σ_X to avoid numerical drift.

7.1 Comparison of Estimates

For each iteration, we compare the Martingale posterior estimates with the exact posterior mean and covariance. This is done by calculating the following:

Exact Posterior Mean for β: The exact posterior mean is given by:

$$\hat{\beta}_n = (X_n^T X_n)^{-1} X_n^T y_n \tag{21}$$

Exact Posterior Covariance for β: The exact posterior covariance is given by:

$$\Sigma_{\beta|n} = \sigma^2(X_n^T X_n)^{-1} \tag{22}$$

Exact Posterior Variance for σ^2: The exact posterior variance for σ^2 is:

$$\sigma_n^2 = \frac{(y_n - X_n\hat{\beta}_n)^T(y_n - X_n\hat{\beta}_n)}{n} \tag{23}$$

7.2 Results and Visualization

We plot the following to compare the Martingale Posterior with the exact posterior:

- Estimation history for β over iterations.
- Estimation history for σ^2.
- Estimation history for elements of Σ_X: For the covariance matrix difference, the Frobenius norm error is:

$$\|\mathbf{\Sigma}_{\text{martingale}} - \mathbf{\Sigma}_{\text{exact}}\|_F = \sqrt{\sum_{i,j}(\Sigma_{ij}^{\text{martingale}} - \Sigma_{ij}^{\text{exact}})^2} \tag{24}$$

Iteration	$(\beta_1^{\text{martingale}} - \beta_1^{\text{exact}})^2$	$(\beta_2^{\text{martingale}} - \beta_2^{\text{exact}})^2$	$(\beta_3^{\text{martingale}} - \beta_3^{\text{exact}})^2$	$(\sigma^2_{\text{martingale}} - \sigma^2_{\text{exact}})^2$
100	0.010429	0.001361	0.009693	0.113569
200	0.003858	0.000417	0.002209	0.023083
300	0.001720	0.000116	0.001058	0.009327
400	0.000967	0.000107	0.000676	0.004924
500	0.000681	0.000072	0.000535	0.003257
600	0.000530	0.000026	0.000424	0.002341
700	0.000412	0.000020	0.000340	0.001732
800	0.000304	0.000015	0.000304	0.001312
900	0.000261	0.000008	0.000271	0.001059
1000	0.000197	0.000012	0.000198	0.000780

Table 1: Absolute Squared Errors for $\beta_1, \beta_2, \beta_3$, and σ^2 at each iteration

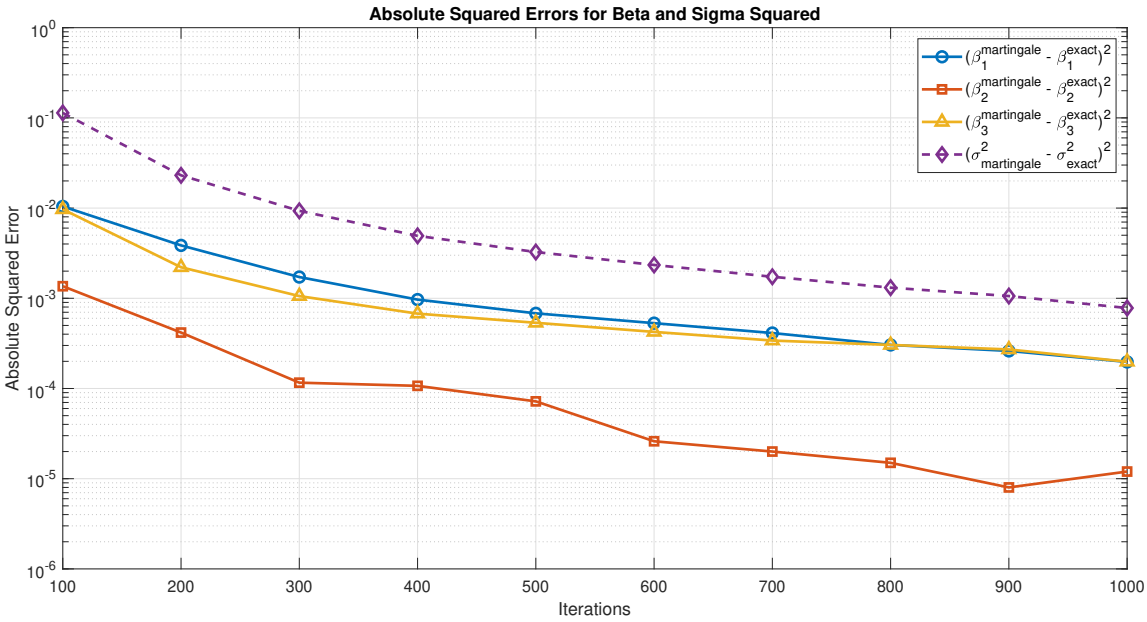

Figure 12: Absolute Squared Errors for $\beta_1, \beta_2, \beta_3$, and σ^2 at each iteration.

Iteration	Frobenius Norm Error
100	2.2421
200	1.9242
300	1.8892
400	1.7190
500	1.5468
600	1.5265
700	1.5131
800	1.5112
900	1.5061
1000	1.5043

Table 2: Frobenius Norm Error for Covariance Matrix at each iteration

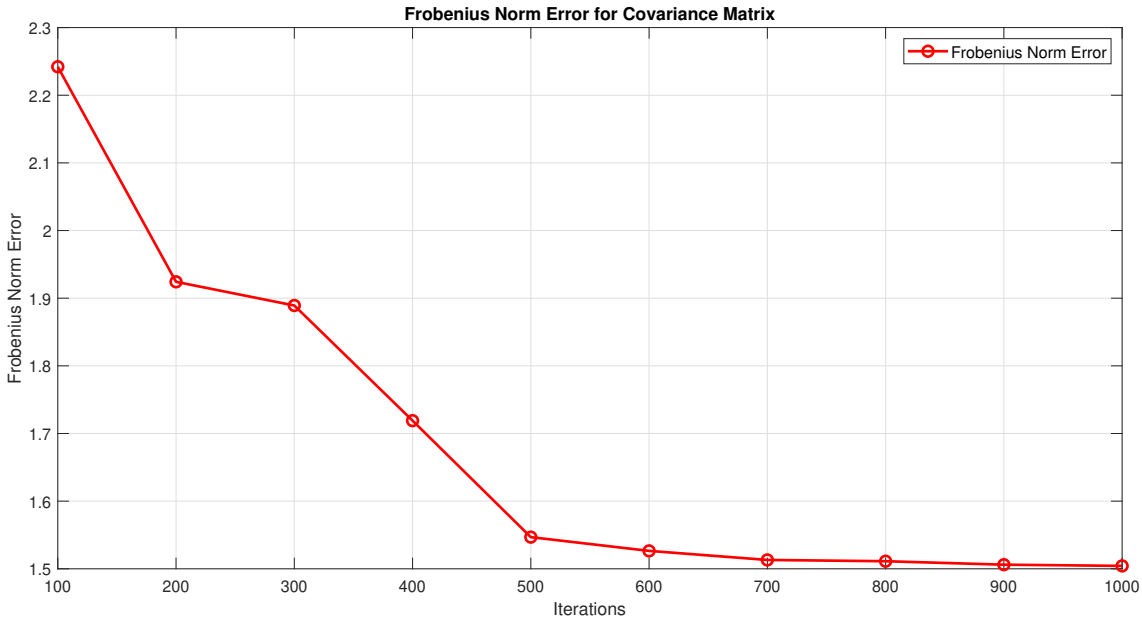

Figure 13: Frobenius Norm Error for Covariance Matrix at each iteration.

Stochastic Gradient Descent (SGD)

Objective

SGD is primarily an optimization algorithm used to minimize a loss function (e.g., mean squared error in regression) by iteratively updating parameters based on a subset of the data.

Update Rule

The parameters are updated using the gradient of the loss function with respect to the parameters, computed from a sample (or mini-batch) of data. The update rule is given by:

$$\beta = \beta - \eta \cdot \nabla L(\beta)$$

where η is the learning rate, and $\nabla L(\beta)$ is the gradient of the loss function.

Data Dependence

SGD can be sensitive to the choice of learning rate and may require tuning. It utilizes only a small subset of the data at each iteration, making it suitable for large datasets.

Martingale Posterior Score Functions

Objective

The Martingale Posterior Score Function approach is rooted in Bayesian statistics. It focuses on updating the parameters based on the likelihood of the observed data, interpreted through the lens of score functions.

Update Rule

The parameters are updated based on the score function, which is the gradient of the log-likelihood function. The update can be viewed as a form of iterative parameter adjustment based on likelihood maximization:

$$\beta = \beta - \eta \cdot \text{score}$$

where the score is typically computed from the entire dataset or a relevant subset, reflecting the underlying probabilistic model.

Data Dependence

The Martingale approach integrates the idea of likelihoods and may not rely on the same mini-batch strategy as SGD. It can be seen as a broader framework for parameter updating that incorporates the statistical properties of the model.

Key Differences

- **Foundation**: SGD is an optimization technique, while Martingale Posterior Score Functions are rooted in statistical inference.
- **Update Basis**: SGD updates parameters based on minimizing a loss function, whereas the Martingale approach updates based on maximizing the likelihood via score functions.
- **Interpretation**: The Martingale approach provides a probabilistic interpretation of updates, whereas SGD focuses on convergence to an optimal parameter value.

8 Neural Networks

Suppose we have data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where for $i \in \{1, \ldots, N\}$, $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. The objective is to infer a predictive model $f : \mathcal{X} \to \mathcal{Y}$ based on the data. One way to do this is with a parametric model of the form $f : \mathcal{X} \times \Theta \to \mathcal{Y}$, with $\Theta \subseteq \mathbb{R}^{d_\theta}$. It is assumed that $x_{1:N}$ are deterministic.

If $\mathcal{Y} = \mathbb{R}^m$, then we assume that for $i \in \{1, \ldots, N\}$,

$$y_i = f(x_i, \theta) + \epsilon_i, \quad \epsilon_i \overset{\text{ind}}{\sim} \mathcal{N}_m(0, \Sigma_i),$$

where ϵ_i are independent and $\mathcal{N}_m(\mu, \Sigma)$ denotes the m-dimensional Gaussian distribution with mean μ and covariance Σ. Hence,

$$y_i \mid \theta \sim \mathcal{N}_m(f(x_i, \theta), \Sigma_i).$$

If $\mathcal{Y} = \{1, \ldots, m\}$, for some $m \in \mathbb{N}$, let $f : \mathcal{X} \times \Theta \to \mathbb{R}^m$ with $f(x, \theta) = (f_1(x, \theta), \ldots, f_m(x, \theta))$, and define the softmax function as

$$h_k(x_i, \theta) := \frac{\exp\{f_k(x_i, \theta)\}}{\sum_{j=1}^m \exp\{f_j(x_i, \theta)\}}, \quad k \in \mathcal{Y}.$$

We assume that $y_i \sim h(x_i, \theta)$, independently for $i \in \{1, \ldots, N\}$, where $h(x, \theta) = (h_1(x, \theta), \ldots, h_m(x, \theta))$ denotes a categorical distribution on m outcomes given input x.

For deep neural networks (DNNs), consider the following architecture. Let $D \in \mathbb{N}$ and $(n_0, \ldots, n_D) \in \mathbb{N}^{D+1}$ be given, where $n_0 = n$ is the input layer and $n_D = m$ is the output layer. Let $A_d \in \mathbb{R}^{n_d \times n_{d-1}}$ and $b_d \in \mathbb{R}^{n_d}$ be the weights and biases for each layer $d \in \{1, \ldots, D\}$. The network parameters are $\theta := \{(A_1, b_1), \ldots, (A_D, b_D)\} \in \Theta = \bigotimes_{d=1}^D \{\mathbb{R}^{n_d \times n_{d-1}} \times \mathbb{R}^{n_d}\}$.

Define the network functions as follows:

$$\begin{aligned} g_0(x, \theta) &:= A_1 x + b_1, \\ g_d(x, \theta) &:= A_d \sigma_{n_{d-1}}(g_{d-1}(x)) + b_d, \quad d \in \{1, \ldots, D-1\}, \\ f(x, \theta) &:= A_D \sigma_{n_{D-1}}(g_{D-1}(x)) + b_D, \end{aligned}$$

where $f(x, \theta)$ is the output of the final layer of the DNN. For a given $f(x, \theta)$ and parameter space Θ, we place a prior $\bar{\pi}$ on Θ. The posterior distribution is given by

$$\pi(\theta \mid y_{1:N}) \propto p(y_{1:N} \mid \theta) \bar{\pi}(\theta),$$

where the likelihood function for regression is

$$p(y_{1:N} \mid \theta) = \prod_{i=1}^N \phi_m(y_i; f(x_i, \theta), \Sigma_i),$$

and for classification, the likelihood function is

$$p(y_{1:N} \mid \theta) = \prod_{i=1}^N \prod_{k=1}^m h_k(x_i, \theta)^{\mathbb{I}[y_i = k]},$$

where $\mathbb{I}[y_i = k]$ is the indicator function that is 1 if $y_i = k$ and 0 otherwise.

8.1 Deep Neural Network Architecture

We now describe the architecture of the deep neural network (DNN) used in our model. The network consists of several layers, each performing an affine transformation followed by an activation function. The general structure of the network is outlined as follows:

- **Input Layer:** The input layer consists of $n_0 = n$ neurons, where n is the dimension of the input data x. Thus, the input $x \in \mathbb{R}^n$.
- **Hidden Layers:** The network contains D hidden layers. Each hidden layer d (for $d \in \{1, \ldots, D\}$) consists of n_d neurons. The dimensions of the layers are given by the sequence $(n_0, n_1, \ldots, n_D)$, where $n_0 = n$ is the input layer and $n_D = m$ is the output layer.
- **Affine Transformation and Activation:** Each hidden layer performs an affine transformation followed by an activation function. Specifically:

$$g_0(x, \theta) = A_1 x + b_1, \tag{25}$$

$$g_d(x, \theta) = A_d \sigma_{n_{d-1}}(g_{d-1}(x)) + b_d, \quad d \in \{1, \ldots, D-1\}, \tag{26}$$

  where $A_d \in \mathbb{R}^{n_d \times n_{d-1}}$ is the weight matrix for the d-th layer, $b_d \in \mathbb{R}^{n_d}$ is the bias vector, and $\sigma_{n_{d-1}}$ is the activation function applied element-wise to the output of the previous layer. A common activation function is the Rectified Linear Unit (ReLU), defined as $\sigma(z) = \max(0, z)$.
- **Output Layer:** The final layer of the network computes the output:

$$f(x, \theta) = A_D \sigma_{n_{D-1}}(g_{D-1}(x)) + b_D, \tag{27}$$

  where $A_D \in \mathbb{R}^{m \times n_{D-1}}$ is the weight matrix for the output layer, and $b_D \in \mathbb{R}^m$ is the bias vector. The output $f(x, \theta) \in \mathbb{R}^m$ represents the predicted values (for regression) or the logits for classification.
- **Parameters:** The parameters of the model are the weight matrices $A_1, A_2, \ldots, A_D$ and bias vectors $b_1, b_2, \ldots, b_D$ for all layers. These parameters are collectively denoted as $\theta = \{(A_1, b_1), \ldots, (A_D, b_D)\} \in \Theta$.

8.2 Martingale Posterior for DNN

The Martingale Posterior update is a term from Bayesian learning in which the posterior distribution of model parameters (such as the weights of a neural network) is updated using score functions over time. The term "Martingale" here refers to a process where the expected value of a future state, given the current state, is equal to the present value (this has applications in financial modeling and is adapted in some machine learning algorithms).

In the context of your problem, we can reinterpret the Martingale Posterior update for frequentist regression as an iterative update process where you adjust the parameters (weights and biases) based on the error signal and a gradient-based update rule, but it does not involve explicitly defining a prior distribution over weights as a typical Bayesian neural network does.

Model Setup: We consider a deep neural network (DNN) model with parameters:

$$\theta := \{(A_1, b_1), \ldots, (A_D, b_D)\}, \tag{28}$$

where $A_d \in \mathbb{R}^{n_d \times n_{d-1}}$ and $b_d \in \mathbb{R}^{n_d}$ are the weight matrices and bias vectors for layer d, respectively. The output of the network is given by:

$$f(x, \theta) = A_D \sigma(g_{D-1}(x, \theta)) + b_D, \tag{29}$$

where σ represents the activation function.

8.2.1 Likelihood Functions

Regression Model: If $\mathcal{Y} = \mathbb{R}^m$, we assume

$$y_i = f(x_i, \theta) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}_m(0, \Sigma_i), \tag{30}$$

and the likelihood function is given by:

$$p(y_{1:N} \mid \theta) = \prod_{i=1}^{N} \phi_m(y_i; f(x_i, \theta), \Sigma_i). \tag{31}$$

Classification Model: If $\mathcal{Y} = \{1, \ldots, m\}$, we define the softmax function:

$$h_k(x_i, \theta) = \frac{\exp\{f_k(x_i, \theta)\}}{\sum_{j=1}^{m} \exp\{f_j(x_i, \theta)\}}, \quad k \in \mathcal{Y}, \tag{32}$$

and the likelihood function is:

$$p(y_{1:N} \mid \theta) = \prod_{i=1}^{N} \prod_{k=1}^{m} h_k(x_i, \theta)^{\mathbb{I}[y_i = k]}. \tag{33}$$

Martingale Posterior Using Score Functions: Instead of using a prior, we generate a sequence of posteriors using a martingale construction based on the score function:

$$\theta_{m+1} = \theta_m + \epsilon_m \cdot s(y_{m+1}, x_{m+1}, \theta_m), \tag{34}$$

where $s(y, x, \theta)$ is the score function:

$$s(y, x, \theta) = \nabla_\theta \log p(y \mid x, \theta). \tag{35}$$

The step size is chosen as $\epsilon_m = \frac{1}{m+1}$ to ensure convergence.

Score Function Computation: For regression:

$$s(y, x, \theta) = \Sigma^{-1}(y - f(x, \theta)) \nabla_\theta f(x, \theta). \tag{36}$$

For classification:

$$s(y, x, \theta) = \sum_{k=1}^{m} \mathbb{I}[y = k] \frac{\nabla_\theta h_k(x, \theta)}{h_k(x, \theta)}. \tag{37}$$

Algorithm Implementation:

1. Initialize θ_n using MLE or another estimator.
2. For each new data point (x_m, y_m):
   (a) Compute the score function $s(y_m, x_m, \theta_m)$.
   (b) Update parameters using:

$$\theta_{m+1} = \theta_m + \frac{1}{m+1} s(y_m, x_m, \theta_m). \tag{38}$$

3. Repeat for all $m \geq n$ to generate a sequence of posterior-like estimates.

Conclusion: This methodology provides a computationally efficient way to perform Bayesian-like inference on deep neural networks without requiring priors or MCMC. By leveraging martingale sequences and score functions, we obtain a posterior distribution that naturally quantifies uncertainty while being parallelizable and scalable.

9 Algorithm Overview

The paper presents a novel algorithm for constructing martingale posteriors using score functions. The method avoids Markov Chain Monte Carlo (MCMC) and instead relies on stochastic gradient-based updates to generate posterior samples.

9.1 Step-by-Step Breakdown

9.1.1 Step 1: Initialization

- Start with collected data $X_1, \ldots, X_n$, where n is the sample size.
- Define a parametric model $p(x|\theta)$, where θ is the parameter to estimate.
- Use an initial estimator θ_n, such as the maximum likelihood estimator (MLE).
- Define the score function:

$$s(x, \theta) = \frac{\partial}{\partial \theta} \log p(x|\theta). \tag{39}$$

- Set a step-size function ϵ_m, commonly chosen as:

$$\epsilon_m = \frac{1}{m+1}. \tag{40}$$

9.1.2 Step 2: Iterative Updates

For each iteration $m \geq n$:

1. **Generate new data point** X_{m+1}:

$$X_{m+1} \sim p(x|\theta_m). \tag{41}$$

2. **Update the parameter** θ_m using the score function:

$$\theta_{m+1} = \theta_m + \epsilon_m s(X_{m+1}, \theta_m). \tag{42}$$

9.1.3 Step 3: Stopping and Posterior Approximation

Repeat until a maximum number of iterations T is reached. Store the final values θ_T from different parallel runs as posterior samples.

9.2 Parallel Computation

Unlike MCMC, this method allows multiple martingales to run independently, making it computationally efficient.

9.3 Pseudocode

Algorithm 1 Martingale Posterior Sampling Algorithm

Input: Observed data $X_{1:n}$, likelihood $p(x|\theta)$, initial estimator θ_n, score function $s(x, \theta)$, step size function ϵ_m, number of iterations T, number of parallel runs D
Initialize: $\theta_{samples} \leftarrow []$
for $d \in \{1, \ldots, D\}$ **do** ▷ Parallel Runs
    $\theta_m \leftarrow \theta_n$
    for $m \in \{n, \ldots, T\}$ **do**
        Sample $X_{m+1} \sim p(x|\theta_m)$
        Update $\theta_{m+1} = \theta_m + \epsilon_m s(X_{m+1}, \theta_m)$
    end for
    Append θ_m to $\theta_{samples}$
end for
Output: $\theta_{samples}$ as posterior samples

9.4 Example: Normal Distribution with Known Variance

For a normal model $X \sim N(\theta, \sigma^2)$, the update formula simplifies to:

$$\theta_{m+1} = \theta_m + \epsilon_m(X_{m+1} - \theta_m). \tag{43}$$

Since the score function for a normal likelihood is:

$$s(x, \theta) = \frac{x - \theta}{\sigma^2}, \tag{44}$$

This update is simply a weighted average.

9.5 Advantages of This Algorithm

- No priors needed, unlike traditional Bayesian methods.
- Avoids computationally expensive MCMC.
- Can be implemented in parallel, improving efficiency.
- Uses gradient-based updates, similar to stochastic gradient descent (SGD).

10 Numerical Simulations

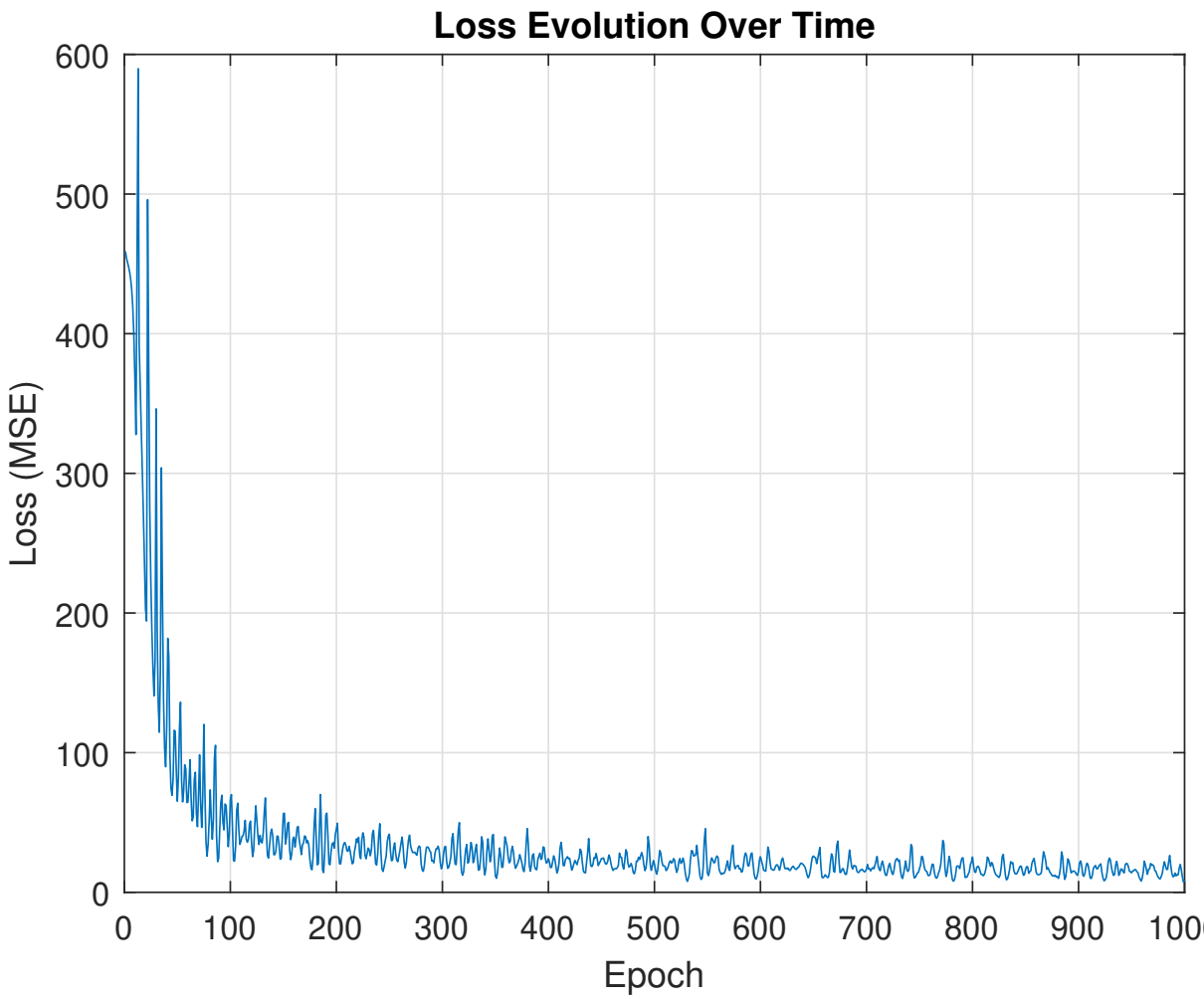

Figure 14: Loss.

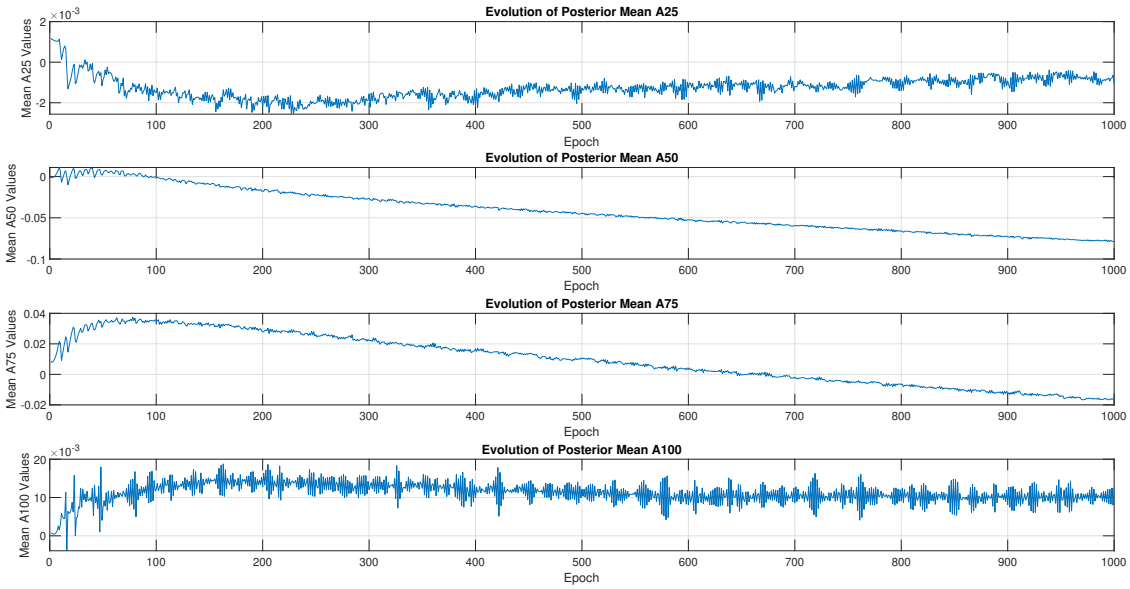

Figure 15: Weights estimation.

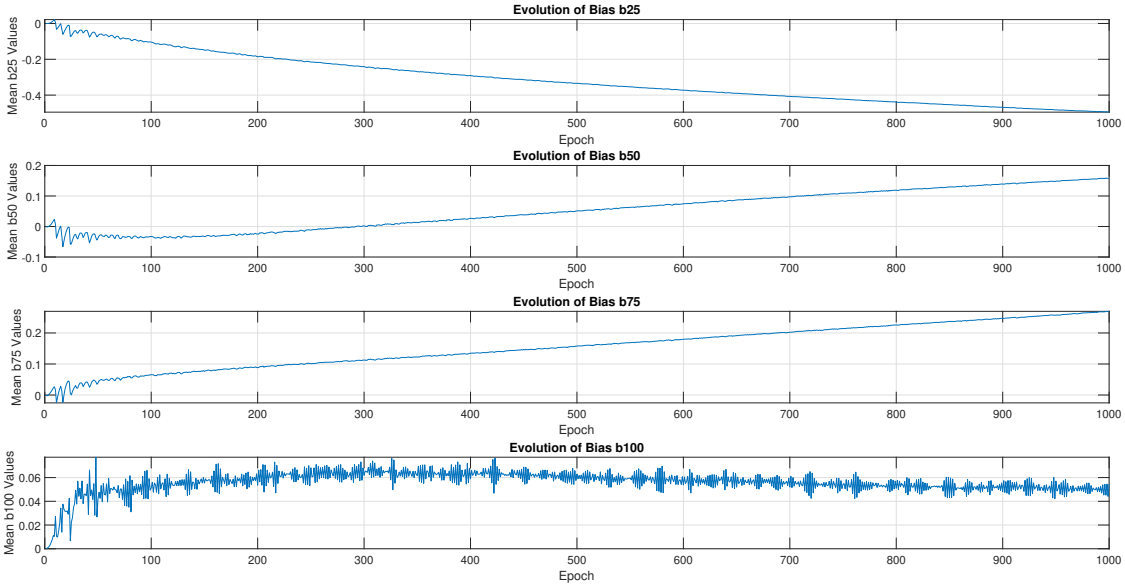

Figure 16: bias estimation.

11 Comparison between Martingale Posterior Deep Neural Networks and Sequential Monte Carlo Methods using MNIST Datasets

In this section, we present a detailed comparative analysis between Martingale Posterior Deep Neural Networks (MP-DNNs) and Sequential Monte Carlo (SMC) methods on the MNIST dataset. Both techniques represent powerful probabilistic approaches for uncertainty quantification in deep learning, yet they rely on fundamentally different principles. MP-DNNs incorporate martingale-based posterior inference to capture epistemic uncertainty within neural networks, allowing for principled decision-making under uncertainty. On the other hand, SMC methods approximate the posterior distribution using a set of weighted samples (particles), which evolve over time through importance sampling, resampling, and propagation. The MNIST dataset, a benchmark in handwritten digit recognition, serves as a suitable platform to evaluate these models in terms of predictive accuracy, calibration, robustness to noise, and computational efficiency. We discuss the implementation details, performance metrics, and experimental findings that shed light on the strengths and limitations of each approach. Ultimately, this comparison aims to provide insights into which method is more suitable for uncertainty-aware learning in classification tasks and how each can be adapted or extended for more complex, real-world scenarios.

11.1 MNIST Dataset

The MNIST (Modified National Institute of Standards and Technology) dataset is a benchmark dataset widely used for evaluating image classification algorithms, particularly in the context of handwritten digit recognition. It consists of 70,000 grayscale images of handwritten digits ranging from 0 to 9, with 60,000 images designated for training and 10,000 for testing. Each image in the dataset is a 28×28 pixel square, resulting in 784-dimensional input vectors for machine learning models. The simplicity and standardization of MNIST make it a popular starting point for developing and testing deep learning models, especially convolutional neural networks (CNNs).

Figure **??** displays sample images from the MNIST dataset, each representing a handwritten digit. The variety in handwriting styles presents a non-trivial challenge for classification models, making MNIST a useful benchmark for assessing generalization capabilities. Despite its age, MNIST remains a fundamental dataset in the field of machine learning due to its accessibility, balanced classes, and well-established performance baselines.