LUISS

Degree Program in Data Science and Management

Course of Advanced Statistics

Gaussian Processes: from theory to applications

Supervisor
Prof.ssa Marta Catalano

Co-Supervisor
Prof. Alessio Martino

Candidate
Leonardo Pulicati
Matricola: 786931

Academic Year 2024/2025

Index

1 Introduction

The primary objective of this thesis is to provide a comprehensive and rigorous explanation of Gaussian Processes, examined from multiple perspectives. A further aim is to investigate whether Gaussian Processes can enhance regression and classification tasks, thereby enabling more accurate analyses and deeper insights.

A balanced evaluation of Gaussian Processes is presented, considering both their advantages and their limitations, and situating them within the broader landscape of modern statistical and machine learning methods.

RoadMap The work begins with an introductory overview of Gaussian Processes, presented in Chapter 2. This section provides both definition and the corresponding mathematical formulations. Particular attention is applied to the role of kernels, which constitute the core component of these models.

The thesis then develops along two main directions. The first focuses on regression (Chapter 3). The discussion starts with a comparison between Frequentist and Bayesian Linear Regression, emphasizing the differences in inference. The analysis then transitions to Gaussian Processes Regression, highlighting its distinction from Bayesian Linear Regression. Then, a visual comparison is provided to enhance the understanding of the differences among the three inference methods. Finally, a real-world case study is presented to connect the theoretical aspects with practical applications.

The second direction addresses classification (Chapter 4). It begins with a general overview of classification, followed by an examination through linear models and then the extension to Gaussian Processes for classification. A central theme in this part is the treatment of the non-Gaussian likelihood, in particular, the differences between the Laplace approximation and Expectation Propagation are analyzed, illustrating how each method provides a Gaussian approximation through distinct approaches. This section also concludes with a real-world case study, in parallel with the structure adopted for regression.

Chapters 2–4 are primarily based on *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006), with simulations reproduced from the book but re-implemented using code. Moreover, the comparisons among methods and the applied real-world case study constitute original contributions of this thesis.

The final part of the thesis introduces a framework for validating Gaussian Processes inference, known as Simulation-Based Calibration (Chapter 5). The general methodology is first described, and its main features are discussed. The framework is then applied to Gaussian Processes models.

This chapter is based on recent works such as *Validating Bayesian Inference Algorithms with Simulation-Based Calibration* (Talts et al., 2018) and *Validating Gaussian Process Models with Simulation-Based Calibration* (McLeod & Simpson, 2021). Moreover, the comparison between Expectation Propagation and Laplace through Simulation Based Calibration constitutes original contribution of this thesis.

The thesis concludes with a summary of the main findings and insights, reflecting on both the theoretical contributions and the empirical results.

Symbols and Notation

Basic statistical concepts used throughout the thesis:

- $Z : \Omega \to R$ denotes a random variable.
- $\mu = E(Z)$ denotes the expected value of a random variable Z.
- $\sigma^2 = \mathrm{Var}(Z) = E\big[(Z - E(Z))^2\big]$ denotes the variance of a random variable Z.
- $\mathrm{Cov}(Z, Y) = E[(Z - E[Z])(Y - E[Y])]$ is the covariance between two random variables Z and Y.
- Σ is the covariance matrix of a random vector $(X_1, \ldots, X_n)$, i.e. $\Sigma_{ij} = \mathrm{Cov}(X_i, X_j)$.
- $|A|$ is the determinant of a squared matrix A.
- $A^\top$ is the transpose of a matrix A, i.e. $(A^\top)_{ij} = A_{ji}$.
- $(I_n)_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$ is the $n \times n$ identity matrix.
- $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} \mathbf{x}_i^2}$ is the Euclidean norm of a vector.
- $\phi : \mathcal{X} \to \mathcal{H}$ is the feature map that maps from input space $\mathcal{X}$ to feature space $\mathcal{H}$.
- $k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp\big(-\frac{1}{2\ell^2}(\mathbf{x}_p - \mathbf{x}_q)^2\big) + \sigma_n^2 \delta_{pq}$ is the squared exponential kernel.
- $\delta_{pq} = \begin{cases} 1 & \text{if } p = q, \\ 0 & \text{if } p \neq q. \end{cases}$ is the kronecker delta.

2 Overview on Gaussian Processes

The roots of Gaussian Processes (GPs) trace back over a century. Early foundations were laid by the Danish astronomer T. N. Thiele in the 1880s and further formalized in the 1940s by the seminal works of Kolmogorov and Wiener in the context of time series analysis. Over the decades, the method was independently developed and applied in various domains: it became known as "kriging" in geostatistics (Matheron, 1973), was applied in meteorology for spatial prediction, and entered the statistical mainstream through the work of O'Hagan (1978) who articulated its use in one-dimensional regression. In the 1990s, Williams and Rasmussen extended GPs to the machine learning context, framing them as flexible models capable of representing infinite-dimensional function spaces.

Let's delve now into the formal definition of Gaussian Processes.

GPs are a collection of random variables, one for each possible input value, any finite number of which have a joint multivariate Gaussian distribution.

A Gaussian (or normal) distribution is a continuous probability distribution over a random variable Z. In the univariate case, Z follows a Gaussian distribution with mean μ and variance σ^2 (defined in the Notation section), denoted by

$$Z \sim \mathcal{N}(\mu, \sigma^2),$$

if its probability density function (PDF) is given by

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right).$$

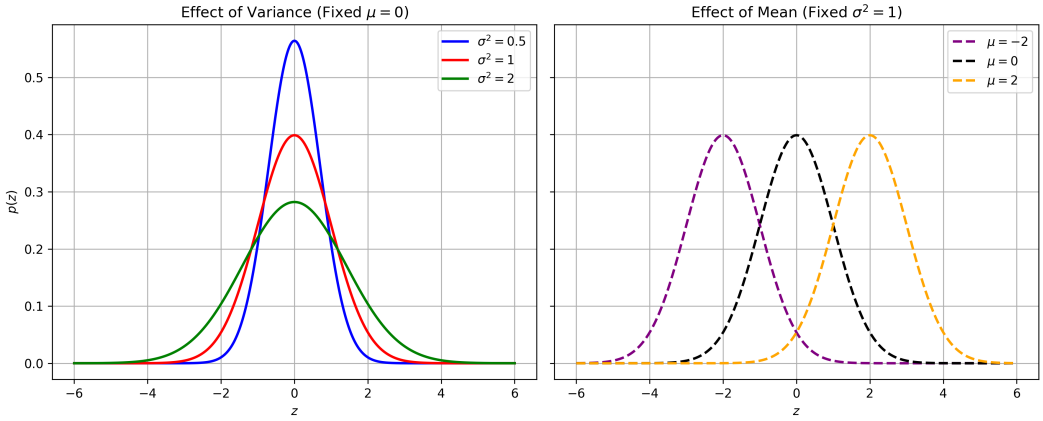


Figure 1: Effect of variance (left) and mean (right) on the probability density function of a univariate Gaussian distribution.

The multivariate Gaussian distribution generalizes the univariate case to vectors of random variables. Let $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]^\top \in R^n$ be a random vector with mean vector $\boldsymbol{\mu} \in R^n$ and covariance matrix $\Sigma \in R^{n \times n}$, then

$$\mathbf{Y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

with probability density function of $\mathbf{Y}$ given by

$$p(\mathbf{y}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{y} - \boldsymbol{\mu})\right),$$

where $\mathbf{y} \in R^n$ is a possible value of $\mathbf{Y}$, and $|\Sigma|$ denotes the determinant (defined in Notation section) of the covariance matrix.

A key property of the multivariate Gaussian distribution is that any linear combination of its components results in a univariate Gaussian distribution. More formally, the multivariate Gaussian distribution is closed under linear transformations. Let

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$

and let $\mathbf{a}$ be a deterministic vector. Then the linear combination

$$Y = \mathbf{a}^\top \mathbf{x}$$

is distributed as a univariate Gaussian random variable with mean

$$E[Y] = \mathbf{a}^\top \boldsymbol{\mu},$$

and variance

$$\mathrm{Var}(Y) = \mathbf{a}^\top \Sigma \, \mathbf{a}.$$

This result highlights the stability of the Gaussian distribution, any weighted sum of its components remains Gaussian, with parameters determined from the mean vector and covariance matrix of the original distribution. This characteristic underlies the formulation of Gaussian Processes, which define distributions over functions such that any finite collection of function values follows a joint Gaussian distribution.

To illustrate the role of the covariance structure in shaping multivariate Gaussians, Figure 2 shows contour plots of bivariate Gaussian distributions with fixed variances but different correlation coefficients. As the correlation varies, the elliptical contours change orientation and eccentricity, reflecting the dependency between the two variables.

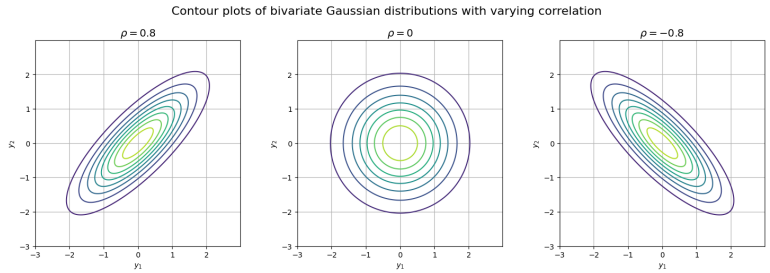


Figure 2: Contour plots of bivariate Gaussian distributions with fixed variances and varying correlation coefficient ρ. The elliptical shape and orientation of the contours reflect the structure of the covariance matrix.

After clarifying the foundational properties of the Gaussian distribution and recalling the formal definition of GPs (see at the beginning of the paragraph), the focus now shifts to the definition of Gaussian Processes.

Definition. *For any finite set of inputs* $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$*, the corresponding function values* $\mathbf{f} = [f(\mathbf{x_1}), f(\mathbf{x_2}), \ldots, f(\mathbf{x_n})]^\top$ *follow a joint multivariate Gaussian distribution*

$$\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, K),$$

where:

- $\boldsymbol{\mu} = [m(\mathbf{x_1}), m(\mathbf{x_2}), \ldots, m(\mathbf{x_n})]^\top$ *is the mean vector;*
- $\mathbf{K} \in R^{n \times n}$ *is the covariance matrix with entries* $K_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$*;*
- $\mathcal{N}$ *denotes the multivariate Gaussian distribution.*

In GPs, the kernel function (covariance matrix) plays a central role in defining their properties. The kernel is typically chosen to express the assumption that points $\mathbf{x_n}$ and $\mathbf{x_m}$ that are close or similar in input space should correspond to function values $f(\mathbf{x_n})$ and $f(\mathbf{x_m})$ that are strongly correlated. Conversely, as points become more distant, their corresponding outputs are expected to become less correlated.

Formally, the kernel is a function $k : R^d \times R^d \to R$ such that for any finite set of inputs $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, the resulting covariance matrix $K \in R^{n \times n}$ is symmetric and positive semi-definite.

There are various types of kernel functions used in Gaussian Processes, such as the *Mercer Kernel*, the *squared exponential Kernel* (also known as the Radial Basis Function (RBF) kernel) and the *Linear Kernel*. Each of these kernels exhibits distinct characteristics, making it possible to shape the covariance structure of the model to suit different types of data and problem domains.

Kernel	**Mathematical Form**	**Description**
Mercer Kernel	$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$	A positive semi-definite function that corresponds to an inner product in a (possibly infinite-dimensional) Hilbert space $\mathcal{H}$ via a feature map $\phi(\cdot)$. Such kernels guarantee that the associated Gram matrix is positive semi-definite for any finite set of inputs, ensuring the existence of a valid Gaussian Processes prior. All kernels listed here are special cases of Mercer kernels.
Squared Exponential Kernel	$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$	A stationary Mercer kernel that produces functions which are infinitely mean-square differentiable.
Linear Kernel	$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$	A specific Mercer kernel where the feature map is the identity: $\phi(\mathbf{x}) = \mathbf{x}$. This kernel encodes the assumption of linearity in the original input space and is appropriate when the relationship between $\mathbf{x}$ and $f(\mathbf{x})$ is well modelled by a linear function.

Table 1: Examples of Mercer kernels adapted from *Machine Learning: A Probabilistic Perspective* (Murphy, 2012) and *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006), with their mathematical forms and interpretations.

The flexibility in defining the covariance structure is a key strength of Gaussian Processes, as it allows to adapt to a wide range of real-world scenarios by choosing an appropriate kernel function. However, this flexibility also introduces a challenge, selecting the most suitable kernel is not always straightforward and often requires domain knowledge or empirical experimentation. The choice of kernel has a significant impact on the performance of the model.

Gaussian Processes can be also interpreted as non-parametric Bayesian models. The term non-parametric indicates that, unlike parametric models where the functional form is determined by a finite set of parameters (such as the weights in linear regression), Gaussian Processes do not restrict the function space to a fixed parametric family, instead, they define a prior distribution directly over the space of functions. This prior encodes assumptions about the properties of the functions, such as smoothness or correlation between input points, through the choice of the kernel.

Once observations are available, the prior is updated via Bayes' rule (Consult Appendix .1 for details) to obtain a posterior distribution over functions that reflects both the prior assumptions and the evidence provided by the data. This Bayesian updating mechanism enables Gaussian Processes not only to make predictions for unseen inputs but also to attach principled measures of uncertainty to those predictions.

Overall, Gaussian Processes are powerful method used to solve regression and classification problems with some advantages with respect to other methods.

One key strenght of GPs models is the capacity of interpolating among observed data points, meaning they can accurately estimate function values at points where data has been collected. This interpolation property is valuable in many applications, as it allows for the reconstruction of functions from sparse or noisy data.

Furthermore, GPs models offer a probabilistic framework that yields both a mean prediction and an associated uncertainty, making them well-suited for tasks requiring confidence estimation. At locations where data are abundant and consistent, the model's predictive variance will be low; conversely, at locations distant from the training data, the uncertainty will increase, reflecting the model's reduced confidence.

Figure 3 summarizes the core mechanism of Gaussian Processes. It illustrates how an input is transformed into an output via function, and emphasizes that while the functions are correlated, the observed outputs are independent each other.

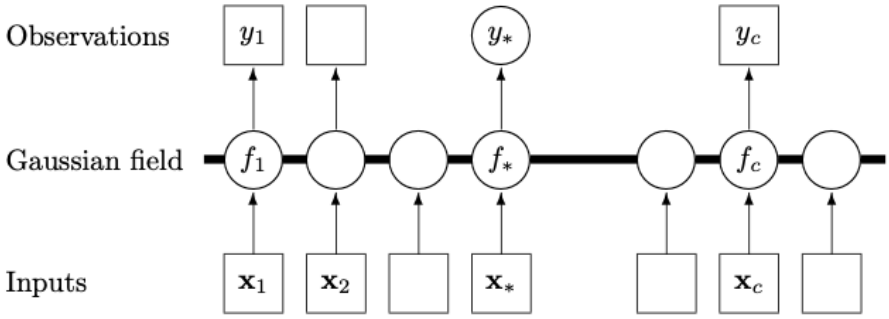


Figure 3: Schematic representation of Gaussian Processes: inputs $\mathbf{x}_i$ are mapped through Gaussian functions f_i, producing observations y_i. Adapted from *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006).

3 Gaussian Processes Regression

3.1 The regression problem

This section focuses on the use of Gaussian Processes in regression tasks.

Unlike classical regression approaches that yield point estimates, Guassian Processes Regression (GPR) returns a full predictive distribution over outputs.

The effectiveness of GPR is strongly influenced by the selection of an appropriate kernel function and by the tuning of its hyperparameters. These elements determine the model's capacity to generalize from data while maintaining sufficient flexibility to capture complex patterns.

Gaussian Processes Regression can be seen as a generalization of Bayesian Linear Regression, where model parameters are treated as random variables with associated prior distributions. This contrasts with frequentist regression approaches, which treat parameters as fixed but unknown quantities and do not provide probabilistic predictions over functions.

3.2 Frequentist Linear Regression

In frequentist linear regression, the parameters $\mathbf{w}$ are treated as fixed but unknown quantities. The model assumes a linear relationship between inputs and outputs, defined by

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}.$$

The observed output is modeled as

$$y = f(\mathbf{x}) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ is Gaussian noise with zero mean and variance σ_n^2, representing model error.

Evaluating the function at $\mathbf{x} = 0$ gives

$$f(0) = \mathbf{0}^\top \mathbf{w} = 0.$$

Given a training dataset $\mathcal{D} = \{X, \mathbf{y}\}$, where $X \in R^{n \times d}$ is the design matrix, a matrix that represents the input data: each of the n rows corresponds to a data point, and each of the d columns corresponds to a feature, and $\mathbf{y} \in R^n$ the observed outputs, the goal is to find a point estimate $\hat{\mathbf{w}}$ that minimizes the residual sum of squares

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|^2.$$

This approach yields a single best estimate $\hat{\mathbf{w}}$ without quantifying uncertainty over the parameters.

3.3 Bayesian Linear Regression

Bayesian Linear Regression (BLR) takes a probabilistic approach, treating the weights $\mathbf{w}$ as random variables and placing a prior distribution over them. This results in a posterior distribution over weights once the data is observed, enabling predictions that incorporate uncertainty.

The model assumes a linear form for the latent function $f(\mathbf{x})$, which maps inputs to outputs

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}.$$

The observed output is modeled as

$$y = f(\mathbf{x}) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$.

Evaluating at the origin yields

$$f(0) = \mathbf{0}^\top \mathbf{w} = 0.$$

Given a training dataset $\mathcal{D} = \{X, \mathbf{y}\}$, the goal of Bayesian Linear Regression is to infer the posterior distribution over the weights $\mathbf{w}$ and use it to compute the predictive distribution for new inputs.

3.3.1 Posterior and Predictive Distribution for Bayesian Linear Regression

Once the model assumptions are formalized, a linear relationship between inputs and outputs, Gaussian noise on observations, and a Gaussian prior on the weights—the next step is to perform Bayesian inference.

Model Setup

Let the dataset be $\mathcal{D} = \{X, \mathbf{y}\}$, the likelihood for the model defined as

$$\mathbf{y} \mid X, \mathbf{w} \sim \mathcal{N}(X\mathbf{w}, \sigma_n^2 I),$$

where the likelihood represents the probability distribution of the observed outputs $\mathbf{y}$, given the inputs X and model parameters $\mathbf{w}$. It measures how well the model, given parameters, explains the observed data. The term σ_n^2 is the noise while the term $I \in R^{n \times n}$ is the identity matrix (defined in Notation section).

The prior over the weights is Gaussian

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma_p),$$

where $\mathbf{w} \in R^d$ and $\Sigma_p \in R^{d \times d}$. The mean equal to zero is typically assumed when no prior knowledge suggests a preferred direction in the parameter space. To ensure that the prior distribution is well-defined and that the resulting posterior is proper, Σ_p must be symmetric and positive semi-definite $\Sigma_p >= 0$.

Theorem 1. *Let $\mathbf{x}_* \in R^d$ be a test input. Then the predictive distribution of the function output $f_* = \mathbf{x}_*^\top \mathbf{w}$ is given by*

$$f_* \mid \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1} X^\top \mathbf{y},\, \mathbf{x}_*^\top A^{-1} \mathbf{x}_*\right),$$

where

$$A = \frac{1}{\sigma_n^2} X^\top X + \Sigma_p^{-1}$$

1. Posterior of the weights The objective is to find the posterior distribution for $f_* = \mathbf{x}_*^\top \mathbf{w}$.

Proof. Adapted from *Pattern Recognition and Machine Learning* (Bishop, 2006).

Applying Bayes' rule (Consult Appendix .1 for details) to the Bayesian Linear Regression setting we have

$$p(\mathbf{w} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid X, \mathbf{w})\, p(\mathbf{w})}{p(\mathbf{y} \mid X)} \propto p(\mathbf{y} \mid X, \mathbf{w})\, p(\mathbf{w}),$$

where the proportionality holds because the denominator does not depend on $\mathbf{w}$. The prior distribution over weights

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \Sigma_p),$$

and the likelihood are Gaussian distributions,

$$p(\mathbf{y} \mid X, \mathbf{w}) = \mathcal{N}(\mathbf{y} \mid X\mathbf{w}, \sigma_n^2 I).$$

To derive the posterior analytically, the logarithm of the unnormalized (without denominator) posterior is taken from

$$\log p(\mathbf{w} \mid X, \mathbf{y}) = \log p(\mathbf{y} \mid X, \mathbf{w}) + \log p(\mathbf{w}) + \mathrm{c}.$$

where c>0 is the normalization constant. This follows from the definition of Bayes' rule and the logarithmic identity $\log(ab) = \log a + \log b$, which simplifies the product of the likelihood and the prior into a sum of log terms. Taking the logarithm facilitates the algebraic manipulation of exponential terms and is a standard approach when dealing with Gaussian distributions.

Each of the terms can now be expanded using the definition of the logarithm of a multivariate Gaussian distribution

$$\log p(\mathbf{y} \mid X, \mathbf{w}) = -\frac{1}{2\sigma_n^2}\|\mathbf{y} - X\mathbf{w}\|^2 + \mathrm{c},$$
$$\log p(\mathbf{w}) = -\frac{1}{2}\mathbf{w}^\top \Sigma_p^{-1}\mathbf{w} + \mathrm{c},$$

In particular, the squared norm $\|\mathbf{y} - X\mathbf{w}\|^2$ corresponds to the squared Euclidean distance between the observed outputs and the model predictions

$$\|\mathbf{y} - X\mathbf{w}\|^2 = (\mathbf{y} - X\mathbf{w})^\top(\mathbf{y} - X\mathbf{w}).$$

The equations follow directly from the log-density of a multivariate Gaussian distribution

$$\log \mathcal{N}(\mathbf{y} \mid X\mathbf{w}, \sigma_n^2 I) = -\frac{1}{2}(\mathbf{y} - X\mathbf{w})^\top(\sigma_n^2 I)^{-1}(\mathbf{y} - X\mathbf{w}) - \frac{1}{2}\log\det(2\pi\sigma_n^2 I)$$

where only the quadratic terms in $\mathbf{w}$ (first part of the equation) are retained here, since terms that do not depend on $\mathbf{w}$ are not relevant for posterior inference.

The squared norm in the likelihood can be expanded as

$$\|\mathbf{y} - X\mathbf{w}\|^2 = (\mathbf{y} - X\mathbf{w})^\top(\mathbf{y} - X\mathbf{w}) = \mathbf{y}^\top\mathbf{y} - 2\mathbf{y}^\top X\mathbf{w} + \mathbf{w}^\top X^\top X\mathbf{w}.$$

Substituting the expansions into the log posterior yields

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \propto -\frac{1}{2\sigma_n^2}\left(\mathbf{y}^\top\mathbf{y} - 2\mathbf{y}^\top X\mathbf{w} + \mathbf{w}^\top X^\top X\mathbf{w}\right) - \frac{1}{2}\mathbf{w}^\top\Sigma_p^{-1}\mathbf{w}.$$

Discarding the constant term $\mathbf{y}^\top\mathbf{y}$, which does not depend on $\mathbf{w}$, the expression simplifies to

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \propto -\frac{1}{2\sigma_n^2}\mathbf{w}^\top X^\top X\mathbf{w} + \frac{1}{\sigma_n^2}\mathbf{w}^\top X^\top\mathbf{y} - \frac{1}{2}\mathbf{w}^\top\Sigma_p^{-1}\mathbf{w}.$$

where $-\frac{1}{2\sigma_n^2}\mathbf{w}^\top X^\top X\mathbf{w}$ and $-\frac{1}{2}\mathbf{w}^\top\Sigma_p^{-1}\mathbf{w}$ are the quadratic terms, corresponding respectively to the likelihood and the prior and $\frac{1}{\sigma_n^2}\mathbf{w}^\top X^\top\mathbf{y}$ is the linear term, derived from the likelihood.

Grouping the quadratic terms and the linear,

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \propto -\frac{1}{2}\mathbf{w}^\top\left(\frac{1}{\sigma_n^2}X^\top X + \Sigma_p^{-1}\right)\mathbf{w} + \mathbf{w}^\top\left(\frac{1}{\sigma_n^2}X^\top\mathbf{y}\right).$$

The expression matches the canonical quadratic form of the logarithm of a multivariate Gaussian distribution,

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \propto -\frac{1}{2}\mathbf{w}^\top A\mathbf{w} + \mathbf{w}^\top\mathbf{b}$$

To identify the mean and covariance explicitly, we complete the square by rewriting this expression as

$$-\frac{1}{2}\mathbf{w}^\top A\mathbf{w} + \mathbf{w}^\top \mathbf{b} = -\frac{1}{2}\left(\mathbf{w}^\top A\mathbf{w} - 2\mathbf{w}^\top \mathbf{b}\right) = -\frac{1}{2}\left(\mathbf{w} - A^{-1}\mathbf{b}\right)^\top A\left(\mathbf{w} - A^{-1}\mathbf{b}\right) + \frac{1}{2}\mathbf{b}^\top A^{-1}\mathbf{b}.$$

This is the standard log-density of a multivariate Gaussian distribution with mean $\boldsymbol{\mu} = A^{-1}\mathbf{b}$ and covariance matrix $\Sigma = A^{-1}$. Therefore, the posterior distribution takes the form

$$p(\mathbf{w} \mid X, \mathbf{y}) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_{\text{post}}, \Sigma_{\text{post}}),$$

with parameters

$$\Sigma_{\text{post}} = A^{-1} = \left(\frac{1}{\sigma_n^2}X^\top X + \Sigma_p^{-1}\right)^{-1},$$
$$\mathbf{w}_{\text{post}} = \Sigma_{\text{post}}\left(\frac{1}{\sigma_n^2}X^\top \mathbf{y}\right).$$

2. Predictive Distribution Having obtained the posterior distribution over the weights, the next step is to derive the predictive distribution for the output corresponding to a new input $\mathbf{x}_* \in R^d$. The predictive quantity of interest is

$$f_* = \mathbf{x}_*^\top \mathbf{w},$$

which represents the model's output at the test location $\mathbf{x}_*$. Since f_* depends linearly on $\mathbf{w}$, and the posterior over $\mathbf{w}$ is Gaussian, the marginal distribution of f_* is also Gaussian. Formally,

$$p(f_* \mid \mathbf{x}_*, X, \mathbf{y}) = \int p(f_* \mid \mathbf{x}_*, \mathbf{w})\, p(\mathbf{w} \mid X, \mathbf{y})\, d\mathbf{w}.$$

Substituting $f_* = \mathbf{x}_*^\top \mathbf{w}$, and noting that $\mathbf{w} \sim \mathcal{N}(\mathbf{w}_{\text{post}}, \Sigma_{\text{post}})$, the distribution of f_* is given by

$$f_* \mid \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(E[f_*], \text{Var}(f_*)\right),$$

where

$$E[f_*] = \mathbf{x}_*^\top \mathbf{w}_{\text{post}} = \mathbf{x}_*^\top \Sigma_{\text{post}}\left(\frac{1}{\sigma_n^2}X^\top \mathbf{y}\right) = \frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1}X^\top \mathbf{y},$$
$$\text{Var}(f_*) = \mathbf{x}_*^\top \Sigma_{\text{post}}\mathbf{x}_* = \mathbf{x}_*^\top A^{-1}\mathbf{x}_*.$$

Therefore, the predictive distribution is

$$f_* \mid \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1}X^\top \mathbf{y},\, \mathbf{x}_*^\top A^{-1}\mathbf{x}_*\right).$$

□

Interpretation

The predictive distribution describes the value of the function at a new input $\mathbf{x}_*$, after observing the data $(X, \mathbf{y})$. The mean term,

$$\frac{1}{\sigma_n^2}\mathbf{x}_*^\top A^{-1}X^\top \mathbf{y},$$

is a linear combination of the observed outputs $\mathbf{y}$. The weights depend on the relation between the new input $\mathbf{x}_*$ and the training inputs, as well as on the noise level σ_n^2 and the prior covariance Σ_p. The matrix A^{-1} combines the contribution from the data through $X^\top X$ and from the prior through Σ_p^{-1}.

The variance term,

$$\mathbf{x}_*^\top A^{-1} \mathbf{x}_*,$$

represents the uncertainty associated with the prediction at $\mathbf{x}_*$. It is small when $\mathbf{x}_*$ lies close to the training inputs, and increases when $\mathbf{x}_*$ is far from them. This term reflects the uncertainty and does not include the noise variance, since the distribution is over the latent function value f_*, not the noise observation.

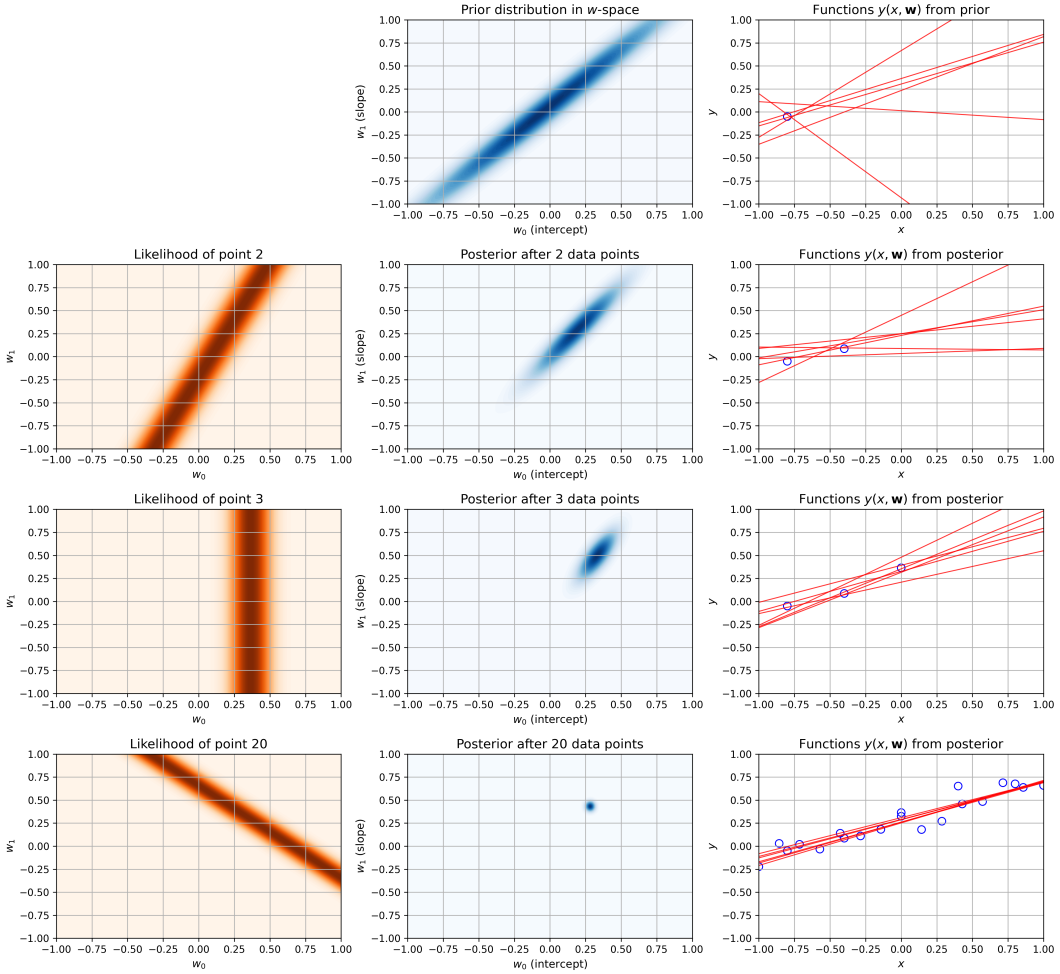


Figure 4: Sequential Bayesian learning in a linear regression model. The left column (except the first row) shows the likelihood at data point; the middle column shows the prior or posterior distribution over weights (w_0, w_1); the right column displays samples of the regression function $y(x, \mathbf{w})$ drawn from the corresponding distribution over $\mathbf{w}$. As more data is observed, the posterior and the predictive functions concentrate around the true model. Adapted from *Pattern Recognition and Machine Learning* (Bishop, 2006) and implemented from scratch in Python.

3.4 Gaussian Processes Regression

In GPR, the focus is on the distribution over functions since rather than defining a prior over the parameters as in BLR, it is defined a prior directly over functions.

In the Gaussian Processes Regression, $f(\mathbf{x})$ previously introduced as a latent function in BLR, is expressed as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),$$

where $m(\mathbf{x})$ is the mean function and $k(\mathbf{x}, \mathbf{x}')$ is the covariance (or kernel) function.

The goal of Gaussian Processes Regression is to update the prior over functions into a Gaussian Processes posterior once data have been observed. In practice, predictions at new inputs are obtained from the predictive distribution, which corresponds to the marginal of the GPs posterior at the test point. Hence, while the GPs posterior describes a distribution over all possible functions, the predictive distribution provides the practically relevant quantity for inference.

3.4.1 Predictive Distribution for Gaussian Processes Regression

Model Setup

In GPR, any finite set of function values follows a multivariate Gaussian distribution (with $m(x) = 0$)

$$\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top \sim \mathcal{N}(\mathbf{0}, K),$$

with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ for all $i, j \in \{1, \ldots, n\}$.

Assuming observations with noise

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_n^2 I), \text{where the mean is equal to 0,}$$

the distribution over the observations becomes

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, K + \sigma_n^2 I).$$

Theorem 2. *Given training data $\mathcal{D} = \{X, \mathbf{y}\}$, where $X \in R^{n \times d}$, $\mathbf{y} \in R^n$, and given a test input $\mathbf{x}_* \in R^d$, the predictive distribution over the function output $f_* = f(\mathbf{x}_*)$ is*

$$f_* \mid \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{y},\ k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{k}_*\right),$$

Focusing on the terms of the predictive distribution, we first consider the matrix $K \in R^{n \times n}$, which is the covariance matrix over the training inputs. The vector $\mathbf{k}_* = [k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_n, \mathbf{x}_*)]^\top \in R^n$ is the vector of covariances between training inputs and the test input, because each entry $k(\mathbf{x}_i, \mathbf{x}_*)$ quantifies the prior covariance between the function value at the training point $\mathbf{x}_i$ and the test point $\mathbf{x}_*$. The term σ_n^2 is the observation noise variance and the matrix I is the identity matrix of size $n \times n$. Finally, $k(\mathbf{x}_*, \mathbf{x}_*)$ represents the prior variance of the function value at the test input $\mathbf{x}_*$, because it represents the variance of the function f at the test input $\mathbf{x}_*$ before any training data has been observed.

Proof. Adapted from *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006).

The goal is to compute the predictive distribution $p(f_* \mid \mathbf{x}_*, X, \mathbf{y})$.

Under the GPs prior, the joint distribution of the observed outputs $\mathbf{y}$ and the value f_* of the latent function at the test input is given by

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma_n^2 I & \mathbf{k}_* \\ \mathbf{k}_*^\top & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right).$$

To proceed, the standard result for conditioning a joint Gaussian distribution is applied. Let

$$\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right),$$

where $\mathbf{z}_1$ denotes observed random variables. In the context of Gaussian Processes Regression, this corresponds to the vector of noise observations $\mathbf{y}$.

The variable $\mathbf{z}_2$ is the random variable we wish to predict conditionally, that is, the function value at the test input f_*.

The vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ represent the prior means of $\mathbf{z}_1$ and $\mathbf{z}_2$, respectively. In standard GPR, these are both set to zero: $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \mathbf{0}$.

The matrix Σ_{11} is the prior covariance matrix of $\mathbf{z}_1$, which in GPR corresponds to $K + \sigma_n^2 I$. The term Σ_{22} is the prior variance of the function at the test input, given by $k(\mathbf{x}_*, \mathbf{x}_*)$.

The matrix Σ_{12} is the cross-covariance vector between the training inputs and the test input, given by $\mathbf{k}_*$, where $\mathbf{k}_* = [k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_n, \mathbf{x}_*)]^\top$.

Finally, Σ_{21} is simply the transpose of Σ_{12}, i.e., $\mathbf{k}_*^\top$.

Then the conditional distribution is

$$\mathbf{z}_2 \mid \mathbf{z}_1 \sim \mathcal{N}\left(\boldsymbol{\mu}_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{z}_1 - \boldsymbol{\mu}_1), \quad \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\right).$$

Identifying terms, we have $\mathbf{z}_1 = \mathbf{y}$, $\boldsymbol{\mu}_1 = \mathbf{0}$, $\mathbf{z}_2 = f_*$, $\boldsymbol{\mu}_2 = 0$, $\Sigma_{11} = K + \sigma_n^2 I$, $\Sigma_{21} = \mathbf{k}_*^\top$, $\Sigma_{12} = \mathbf{k}_*$, and $\Sigma_{22} = k(\mathbf{x}_*, \mathbf{x}_*)$.

Substituting into the conditioning formula, we have the predictive distribution

$$f_* \mid \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{y},\ k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{k}_*\right)$$

□

Interpretation

The mean term, $\mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{y}$, is a weighted average of the training outputs $\mathbf{y}$, where the weights depend on the similarity between the test input $\mathbf{x}_*$ and the training inputs. This similarity is measured by the kernel function. The matrix $(K + \sigma_n^2 I)^{-1}$ adjusts these weights by accounting for both the structure of the data (through the kernel matrix K) and the presence of noise in the observations (through the noise term σ_n^2).

The variance term, $k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{k}_*$, quantifies the uncertainty in the prediction at $\mathbf{x}_*$. It starts from the prior variance $k(\mathbf{x}_*, \mathbf{x}_*)$, then subtracts the amount of information gained from the training data. The second term acts as a correction: the more similar $\mathbf{x}_*$ is to the training inputs, the larger this subtraction becomes, reducing uncertainty. In regions far from the training data, the correction becomes small, and the model reverts to the prior. This balance ensures that the predictive distribution interpolates the data while retaining uncertainty where appropriate.

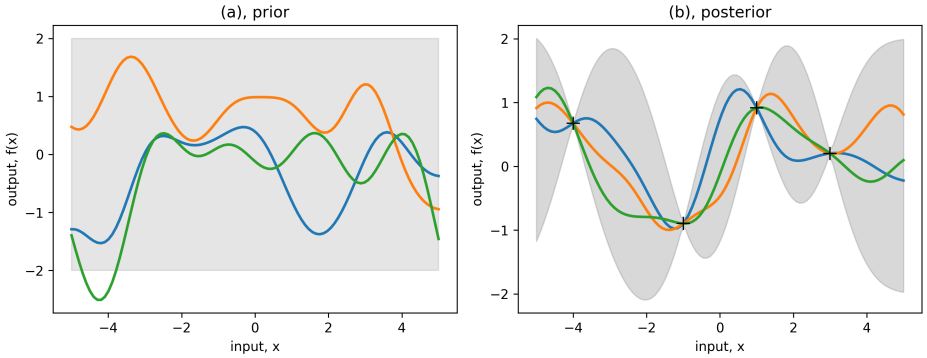


Figure 5: Gaussian Processes Regression with zero-mean prior and squared exponential kernel. (a) Three sample functions drawn from the GPs prior, which reflect the model's assumptions before any data is observed. (b) Three sample functions drawn from the posterior after conditioning on observed data (black crosses). The shaded areas indicates the uncertainty (± 2 standard deviation). Adapted *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006) and implemented from scratch in Python.

3.5 Bayesian Linear Regression vs Gaussian Processes Regression

Looking to the equations of BLR and GPR, the same $f(\mathbf{x})$ is defined in two different forms. This difference highlights a fundamental conceptual shift: Bayesian Linear Regression relies on a parameterized model, assuming a specific structure for $f(\mathbf{x})$ and fitting it to the data. In contrast, Gaussian Processes Regression does not assume a fixed functional form. Instead, it defines a prior directly over functions and uses observed data to refine this belief, with the kernel function controlling how flexible and generalised the resulting predictions can be.

Although both Bayesian Linear Regression and Gaussian Processes Regression yield Gaussian predictive distributions, they differ significantly in formulation, interpretability, and flexibility.

From a computational standpoint, BLR requires operations involving the design matrix X, such as the inversion of $X^\top X$, and relies on an explicit prior over $\mathbf{w}$. In contrast, GPR performs inference entirely through kernel evaluations, which makes it possible to work implicitly in infinite-dimensional feature spaces without computing explicit feature representations. However, this advantage comes at a computational cost: GPR requires forming and inverting the $n \times n$ kernel matrix, which entails a complexity of $\mathcal{O}(n^3)$. This scalability limitation contrasts with BLR, where the inversion involves a $d \times d$ matrix, with complexity $\mathcal{O}(d^3)$. As a result, BLR can be computationally more efficient when the number of features d is relatively small compared to the number of data points n, whereas GPR becomes expensive as the dataset size grows.

Another difference is in levels of flexibility. BLR is fundamentally limited by its linear structure. GPR, instead, is inherently more flexible due to the expressive power of kernels, enabling it to capture non-linear patterns naturally and without requiring explicit feature engineering.

In both methods, the predictive distribution is Gaussian, but the route to obtaining it differs. In BLR, it is derived by marginalizing over the posterior distribution of the weights. In GPR, the predictive distribution for the function value at a new input is obtained by conditioning the joint Gaussian prior on the observed outputs.

In summary, while BLR offers a parameter-centric, interpretable approach to regression, GPR provides a powerful non-parametric framework that models distributions over functions directly, enabling a more data-adaptive inference.

Visual Comparison of the Regression Models

To complement the theoretical derivations and provide a more intuitive understanding of the differences among the three regression models—Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression—we now present a visual analysis on three distinct datasets: one synthetic and linear, one synthetic and nonlinear, and a real-world dataset.

The first dataset consists of synthetic observations generated from a linear function with additive Gaussian noise. Specifically, inputs were sampled uniformly in a symmetric interval and outputs were computed from a linear function with normally distributed noise added. This dataset serves as a controlled environment in which the underlying function truly is linear, thus allowing us to evaluate how closely each model adheres to the true generative process when the linear assumption holds exactly.

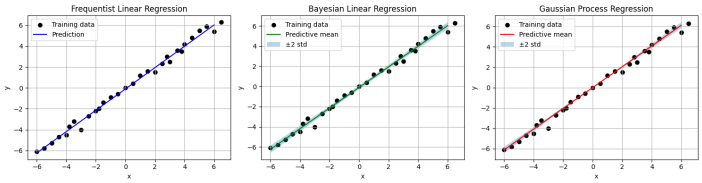


Figure 6: Regression results on a synthetic linear dataset. All three methods recover the linear structure accurately. The Bayesian and Gaussian Processes models additionally provide predictive uncertainty in the form of confidence bands.

As shown in Figure 6, all three methods perform nearly identically in terms of mean prediction. The frequentist method returns a single best-fit line, while the Bayesian and Gaussian Process models additionally offer an estimate of predictive uncertainty. However, in this setting, the confidence bands remain narrow and consistent across the input space, indicating that all models agree strongly and that the uncertainty is uniformly low. This is expected, as the data are generated according to the linear model assumption.

The second dataset is also synthetic but generated from a highly nonlinear function: a sine curve with additive Gaussian noise. The aim here is to test how the three models behave when the true underlying relationship departs significantly from linearity. This dataset illustrates the limitations of linear models when the linear assumptions no longer hold and highlights the advantages of nonparametric approaches like Gaussian Processes.

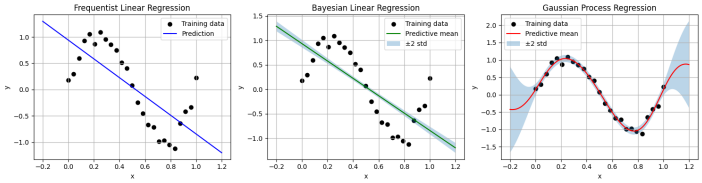


Figure 7: Regression results on a synthetic nonlinear dataset sampled from a sine function with noise. While both linear models fail to capture the underlying pattern, Gaussian Processes Regression adapts flexibly to the curvature and yields accurate uncertainty estimates.

Figure 7 clearly demonstrates that both frequentist and Bayesian Linear Regression are inadequate in capturing the oscillatory behavior of the data. Their predictions follow a straight line and thus incur systematic bias, regardless of the noise level. In contrast, Gaussian Processes Regression successfully models the underlying sine wave, including the peaks and troughs, and provides uncertainty estimates that widen in regions with fewer data points.

Finally, we consider a real-world dataset known as the *Advertising* dataset, which records the amount spent on advertising campaigns through various media channels and the corresponding product sales. For simplicity and interpretability, we focus on a univariate regression problem: predicting sales as a function of TV advertising expenditure. The relationship is generally expected to be positive and approximately linear, though local fluctuations may exist due to unobserved factors or nonlinear market effects.

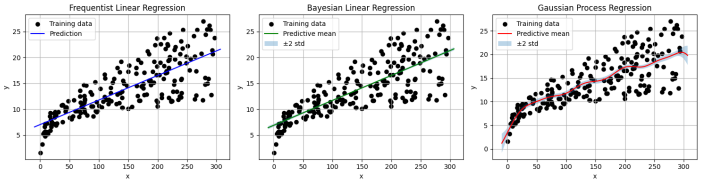


Figure 8: Regression results on the *Advertising* dataset, modeling product sales as a function of TV advertising budget. The models produce broadly similar predictions. Small local deviations are slightly better captured by Gaussian Processes Regression.

As shown in Figure 8, the three models provide very similar predictions for this dataset. This is consistent with the assumption of an approximately linear relationship between TV spending and sales. However, small nonlinear deviations are more visibly captured by Gaussian Processes Regression, which slightly adjusts its predictions in regions where the data suggest curvature. The uncertainty band is constant in the BLR while in GPR remain relatively narrow in the central regions and widen near the boundaries, reflecting reduced confidence in extrapolation.

These examples collectively demonstrate that while linear models are effective and sufficient when the true relationship is linear or nearly so, Gaussian Processes Regression offers a flexible alternative that adapts naturally to more complex patterns in the data. In both synthetic and real-world settings, GPR not only matches the accuracy of linear models but also improves performance and interpretability where linear assumptions no longer hold.

3.6 Hyperparameters

In Gaussian Processes Regression, a set of hyperparameters is used to define the structure of the kernel function.

In this framework, we adopt the Squared Exponential kernel (define in the Notation section).

How Hyperparameters influence GPR

Each hyperparameter in the kernel controls a distinct property of the GPR.

$\ell > 0$: the length-scale. It determines how quickly the correlation between function values decays with distance in input space. A small ℓ leads to functions that vary quickly over short distances (high-frequency functions), allowing the model to follow local fluctuations in the data. A large ℓ forces the function to vary more slowly, capturing only global trends. It effectively determines the smoothness of the function.

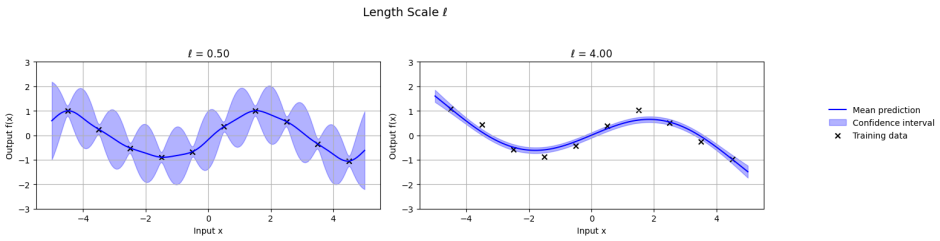


Figure 9: Effect of the length-scale ℓ: left shows a small ℓ (high flexibility), right shows a large ℓ (smoother functions).

$\sigma_f^2 > 0$: the signal variance. It defines the vertical scale of the variations in the function values, how far values of $f(\mathbf{x})$ deviate from the mean. A large σ_f^2 allows the model to fit functions with high-amplitude oscillations, while a small value restricts the function to stay closer to the mean.

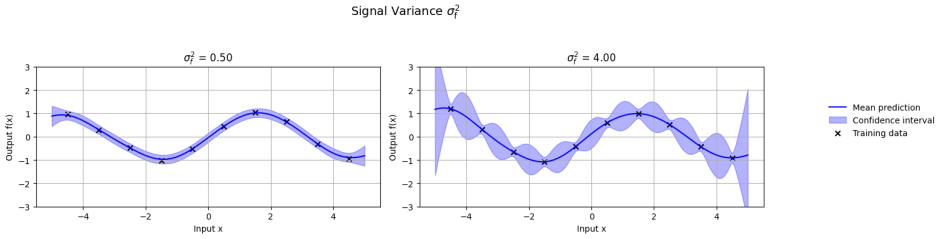


Figure 10: Effect of signal variance σ_f^2: left shows low variance (flattened function), right shows high variance (more vertical variation).

$\sigma_n^2 > 0$: the noise variance. This parameter models the level of additive Gaussian noise assumed in the training observations. It is added only on the diagonal of the covariance matrix. A small σ_n^2 leads the model to interpret even small variations as signal (risking overfitting), while a large σ_n^2 makes the model ignore minor deviations, potentially underfitting the data.

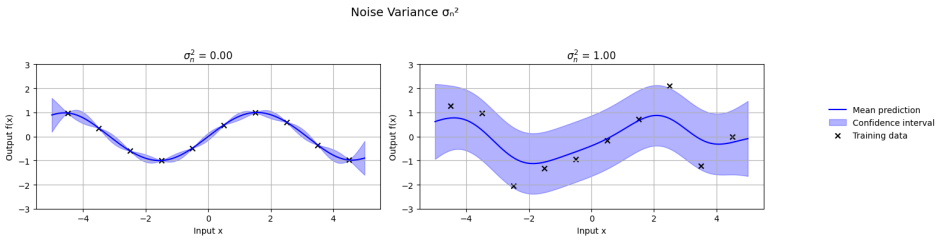


Figure 11: Effect of noise variance σ_n^2: left shows small noise, right shows high noise.

Hyperparameters Selection

Hyperparameters can be manually set based on prior domain knowledge, but more robustly they are learned from data by maximizing the log marginal likelihood, also known as the evidence, that is the probability of the observed data under the model after integrating out the function values

$$\log p(\mathbf{y} \mid X, \theta) = -\frac{1}{2}\mathbf{y}^\top (K + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log\left|K + \sigma_n^2 I\right| - \frac{n}{2}\log(2\pi),$$

where $\mathbf{y}$ is the vector of training targets, K is the covariance matrix computed by the kernel function over the training inputs X, σ_n^2 represents the noise variance, I is the identity matrix, and $\theta = \{\ell, \sigma_f^2, \sigma_n^2\}$ denotes the set of hyperparameters. The symbol $|\cdot|$ refers to the determinant of a matrix.

This expression comprises three terms. The first term, $-\frac{1}{2}\mathbf{y}^\top (K + \sigma_n^2 I)^{-1}\mathbf{y}$, quantifies how well the model explains the observed data. It is a weighted quadratic form that penalizes poor fit to the training targets. The second term, $-\frac{1}{2}\log\left|K + \sigma_n^2 I\right|$, penalizes overly flexible models by measuring the determinant of the covariance matrix. Larger determinants indicate more uncertainty or flexibility, leading to stronger penalization. The third term, $-\frac{n}{2}\log(2\pi)$, arises from the Gaussian likelihood and is independent of the hyperparameters.

Maximizing the log marginal likelihood enables the automatic selection of hyperparameters by balancing model fit and complexity.

3.7 Equivalent Kernel

In Gaussian Processes Regression, the model is traditionally formulated in terms of a kernel $k(\mathbf{x}, \mathbf{x}')$, defining a prior over functions. However, GPR can equivalently be viewed as a linear smoother. This means that the predictive mean at a new input point $\mathbf{x}_*$ can be expressed as a weighted sum of the training targets

$$\mu(\mathbf{x}_*) = \sum_{i=1}^{n} \alpha_i(\mathbf{x}_*)\, \mathbf{y}_i,$$

where $\mu(\mathbf{x}_*) \in R$ is the predictive mean at test input $\mathbf{x}_*$, $\alpha_i(\mathbf{x}_*) \in R$ is the weight (from the equivalent kernel) assigned to training target $\mathbf{y}_i$, and $\mathbf{y}_i \in R$ is the i-th training target.

The weights $\alpha_i(\mathbf{x}_*)$ depend only on the training inputs and the kernel function, not on the target values themselves. These weights form the so-called *equivalent kernel*, denoted $\alpha(\mathbf{x}_*) \in R^n$.

This interpretation is made precise in the following equation, which expresses the predictive mean as

$$\mu(\mathbf{x}_*) = \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{y},$$

where $\mathbf{k}_* \in R^n$ is the vector $[k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_n)]^\top$, $K \in R^{n \times n}$ is the kernel matrix between training inputs and $\mathbf{y} \in R^n$ is the vector of training target values.

By comparing this expression to the linear smoother form, it follows that

$$\alpha(\mathbf{x}_*) = (K + \sigma_n^2 I)^{-1} \mathbf{k}_*,$$

where $\alpha(\mathbf{x}_*) \in R^n$ is the vector of weights (equivalent kernel) used in the linear combination of targets, and the other symbols are as defined above.

Hence, the equivalent kernel is the vector of weights that defines the influence of each training target $\mathbf{y}_i$ on the prediction at $\mathbf{x}_*$. These weights depend on the spatial arrangement of the inputs and on the kernel function, encoding how uncertainty and geometry interact in the model.

A key feature of this formulation is that the weighting structure is determined entirely by the kernel and the input configuration—before observing the target values. The target vector $\mathbf{y}$ only enters the predictive mean via this fixed linear combination.

The notion of the equivalent kernel shifts the focus from the values of the targets to the structure imposed by the kernel. It reveals that the Gaussian Processes defines in advance how information will flow from the training inputs to any test point, thereby offering a transparent and interpretable mechanism for understanding both prediction and uncertainty in terms of data geometry and kernel behavior.

3.8 Basis Functions

Although Gaussian Processes Regression is inherently a non-parametric method, it can be extended to include a parametric component in the form of basis functions. These are explicit functions of the input, such as polynomials (e.g., $\phi(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^2]^\top$), that capture global trends or structured behavior in the data.

In this extended formulation, the Gaussian Processes prior is defined not just on the function $f(\mathbf{x})$, but on deviations from a parametric mean function. Specifically, we define the prior as

$$f(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad \text{with} \quad m(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\beta},$$

where $\boldsymbol{\phi}(\mathbf{x}) \in R^p$ is a vector of basis functions (p basis functions) evaluated at input $\mathbf{x}$, and $\boldsymbol{\beta} \in R^p$ is a vector of coefficients (often assumed to follow a Gaussian prior).

Typical choices for $\boldsymbol{\phi}(\mathbf{x})$ include constant function $\phi(\mathbf{x}) = 1$, linear terms $\phi(\mathbf{x}) = \mathbf{x}$, polynomial terms $\phi(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^2, \ldots, \mathbf{x}^d]^\top$, and Fourier components $\phi(\mathbf{x}) = [\sin(\mathbf{x}), \cos(\mathbf{x})]^\top$.

The posterior predictive mean of GPR with basis functions becomes

$$\mu(\mathbf{x}_*) = \boldsymbol{\phi}(\mathbf{x}_*)^\top \boldsymbol{\beta}_* + \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} (\mathbf{y} - \Phi \boldsymbol{\beta}_*),$$

where $\boldsymbol{\phi}(\mathbf{x}_*) \in R^p$ is the vector of basis functions evaluated at the test input $\mathbf{x}_*$, $\boldsymbol{\beta}_* \in R^p$ is the posterior mean of the basis coefficients, $\mathbf{k}_* \in R^n$ is the vector of kernel evaluations $[k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_n)]^\top$, $K \in R^{n \times n}$ is the covariance matrix computed from the kernel function over the training inputs, $\Phi \in R^{n \times p}$ is the design matrix whose i-th row is $\boldsymbol{\phi}(\mathbf{x}_i)^\top$, and $\mathbf{y} \in R^n$ is the vector of training targets.

This approach allows the model to capture both global trends, through the parametric mean function $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\beta}$, and local, non-linear variations, through the stochastic process governed by the kernel $k(\mathbf{x}, \mathbf{x}')$.

Including basis functions is especially useful when prior knowledge suggests a functional form, such as linear or quadratic behavior. For instance, if the data trend is approximately linear, choosing $\boldsymbol{\phi}(\mathbf{x}) = [1, \mathbf{x}]^\top$ enables the GPR to explicitly model this component and reserve the kernel for modeling residual structure.

Thus, GPR with basis functions results in a hybrid model that combines the interpretability and extrapolation capabilities of parametric models with the flexibility of non-parametric kernel-based inference.

3.9 Real World Case

In this section, a real world case for Gaussian Processes Regression is presented.

3.9.1 Real-world Application: Predicting Concrete Strength with GPR

We apply the Gaussian Processes Regression to a real-world dataset, the Concrete Compressive Strength dataset from the UCI Machine Learning Repository. This dataset contains 1030 observations, each representing a different concrete mixture with its measured compressive strength (in MPa) as the target variable. Each instance includes eight features: the amounts of cement, blast furnace slag, fly ash, water, superplasticizer, coarse and fine aggregates, and the age of the concrete in days.

Model Setup. A GPR model was trained on a subset of 800 samples. Prior to training, both input features and target values were standardized to zero mean and unit variance. A zero-mean Gaussian Processes prior was assumed

$$f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')),$$

where $k(\mathbf{x}, \mathbf{x}')$ is the Radial Basis Function kernel. To account for observation noise, the covariance matrix was augmented with a noise term $\sigma_n^2 I$. All three hyperparameters—length-scale ℓ, signal variance σ_f^2, and noise variance σ_n^2—were optimized by maximizing the log-marginal likelihood.

Prediction and Evaluation. The model was evaluated on 230 unseen test samples. For each test input, GPR provides both the predictive mean and predictive variance, capturing the model's confidence in its predictions. The Root Mean Squared Error (RMSE) (see Appendix .8 for details) between predicted and true compressive strengths was computed on the test set. The final RMSE for GPR was 5.68 MPa.

As a baseline, a Bayesian Linear Regression model was trained on the same training samples and evaluated on the same test set. The resulting RMSE was 10.02 MPa.

Results and Interpretation. Figure 12 reports the predictive performance of the GPR model on the test set. The model is able to closely follow the underlying trends in the observed compressive strengths, and the predictive uncertainty shows some variations across the test points, although these differences are not particularly pronounced.

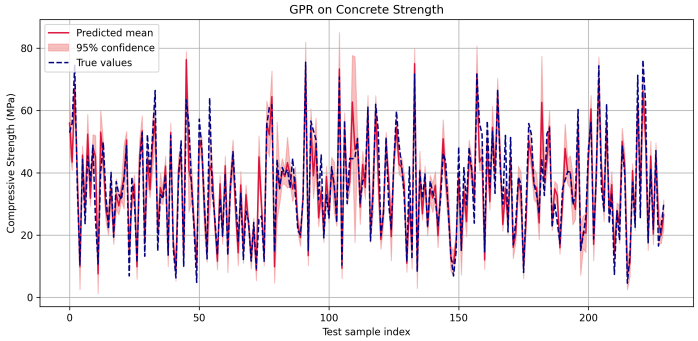


Figure 12: Gaussian Processes Regression predictions on the Concrete Strength dataset. The red curve indicates the predicted mean, the shaded area represents the 95% confidence interval and the dashed blue line shows the observed values.

Figure 13 presents the Bayesian Linear Regression baseline. The resulting bands tend to remain relatively uniform across the input space, conveying a global notion of uncertainty rather than adapting strongly to local variations in the data.

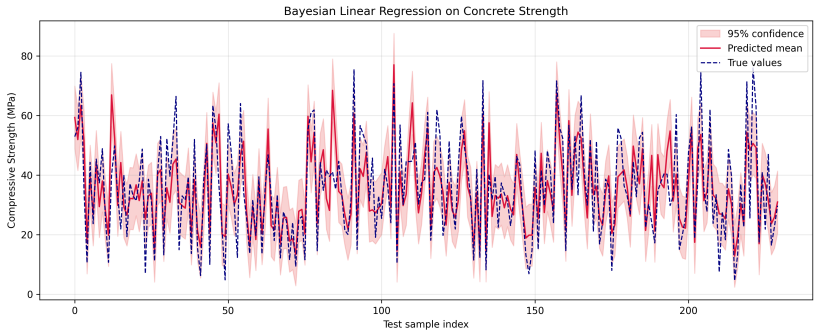


Figure 13: Bayesian Linear Regression predictions on the Concrete Strength dataset. The red curve indicates the predicted mean, the shaded band corresponds to the 95% confidence interval and the dashed blue line shows the observed values.

Overall, the comparison shows that GPR achieves a substantially lower RMSE (5.68 MPa) than Bayesian Linear Regression (10.02 MPa). Moreover, GPR produces predictive intervals that adapt to the distribution of training data, capturing input-dependent uncertainty. In contrast, Bayesian Linear Regression yields intervals that remain more uniform across the input space, as uncertainty is primarily driven by global parameter variability and observation noise. This makes GPR generally more flexible and informative for predicting compressive strength in concrete mixtures.

4 Gaussian Processes Classification

After presenting Gaussian Processes in the context of regression, we now extend the discussion to the classification setting.

4.1 General Framework for Classification

Classification problems can be approached by considering the joint distribution $p(y, \mathbf{x})$, where y represents the class label and $\mathbf{x}$ is the input. Bayes' theorem allows this joint distribution to be factorized in two different ways: either as $p(y)p(\mathbf{x}|y)$ or as $p(\mathbf{x})p(y|\mathbf{x})$. These two decompositions lead to distinct modelling paradigms known respectively as the generative and the discriminative approaches.

In the generative approach to classification, the class-conditional distributions $p(\mathbf{x} \mid y)$ are modeled for each class y, along with the prior class probabilities $p(y)$. Given these components, the posterior distribution over the class labels can be computed using Bayes' rule. In a multiclass setting with C distinct classes $\{C_1, \ldots, C_C\}$, the posterior probability of class y given an input $\mathbf{x}$ is given by

$$p(y \mid \mathbf{x}) = \frac{p(y)\, p(\mathbf{x} \mid y)}{\sum_{c=1}^{C} p(C_c)\, p(\mathbf{x} \mid C_c)},$$

where $p(y)$ denotes the prior probability of class y, $p(\mathbf{x} \mid y)$ is the likelihood of the input under that class, and the denominator ensures proper normalization over all possible classes.

Conversely, the discriminative approach focuses on modelling the posterior $p(y|\mathbf{x})$ directly, without requiring an explicit model for the input distribution $p(\mathbf{x})$ or the class-conditional densities. This approach, referred to as the "diagnostic paradigm", is particularly appealing when the primary objective is classification itself rather than understanding the data generation process.

An illustrative example of a discriminative model is logistic regression (see Appendix .2 for details). This approach transforms the output of a linear model $\mathbf{x}^\top \mathbf{w}$ into a valid class probability through a response function. A common choice for this transformation is the logistic (sigmoid) function (see Appendix .3 for details)

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

The resulting probability of class membership becomes $p(C_i|\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w})$, which maps the real-valued output of the linear model into the interval $[0, 1]$, allowing for probabilistic interpretation. An alternative to the logistic function is the cumulative density function (CDF) (see Appendix .3 for details) of the standard normal distribution, leading to what is known as probit regression.

Both generative and discriminative approaches have strengths and limitations. The discriminative framework is particularly efficient when the ultimate goal is prediction, since it directly targets the conditional distribution $p(y|\mathbf{x})$. It also avoids the complexity of modelling $p(\mathbf{x}|y)$, which may be unnecessary if classification is the only concern. However, the generative approach can offer advantages in situations involving missing data, outliers, as it models the joint distribution and can therefore marginalize over unobserved variables.

The Gaussian Processes Classification approach discussed in this chapter adopts a discriminative perspective. Conceptually, this framework extends logistic regression by replacing the fixed parametric form with a more flexible kernel-based model, enabling non-linear decision boundaries and more expressive uncertainty estimates.

4.2 Linear Models for Classification with Bayesian Inference (Bayesian Logistic Regression)

Linear models represent the foundation for many supervised classification algorithms. In this setting, a real-valued latent function is introduced, defined as

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w},$$

where $\mathbf{x} \in R^d$ is the input vector and $\mathbf{w} \in R^d$ is a vector of weights. The function $f(\mathbf{x})$ is intrepreted as a score reflecting the confidence of the model in assigning the input to one class or the other.

To convert this score into a valid probability, a response function $\sigma(\cdot)$ is applied.

Bayesian Inference: Posterior Distribution To perform Bayesian inference, a prior is placed on the weights $\mathbf{w}$, commonly a zero-mean Gaussian

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \Sigma_p).$$

where $\Sigma_p \in R^{d \times d}$. The data then enter the model through the likelihood. For binary classification with labels $y_i \in \{-1, +1\}$, the likelihood is defined as

$$p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \sigma\big(y_i \cdot \mathbf{x}_i^\top \mathbf{w}\big),$$

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, assuming conditional independence, the posterior distribution over $\mathbf{w}$ is obtained via Bayes' rule

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathbf{w}) \prod_{i=1}^n p(y_i \mid \mathbf{x}_i, \mathbf{w}).$$

Substituting the expressions, we get the unnormalized posterior

$$p(\mathbf{w} \mid \mathcal{D}) \propto \exp\left(-\frac{1}{2}\mathbf{w}^\top \Sigma_p^{-1} \mathbf{w}\right) \prod_{i=1}^n \sigma(y_i \cdot \mathbf{x}_i^\top \mathbf{w}).$$

Taking the logarithm, the log-posterior becomes

$$\log p(\mathbf{w} \mid \mathcal{D}) = -\frac{1}{2}\mathbf{w}^\top \Sigma_p^{-1} \mathbf{w} + \sum_{i=1}^n \log \sigma(y_i \cdot \mathbf{x}_i^\top \mathbf{w}) + \text{c}.$$

This posterior is non-Gaussian and does not admit a closed-form solution. However, it can be approximated using methods such as Laplace approximation (see Appendix .4 for details), which fits a Gaussian centered at the MAP (maximum a posteriori) estimate.

Bayesian Prediction: Predictive Distribution To predict the label y_* for a new input $\mathbf{x}_*$, we compute the predictive distribution

$$p(y_* \mid \mathbf{x}_*, \mathcal{D}) = \int p(y_* \mid \mathbf{x}_*, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w}.$$

For binary classification

$$p(y_* = +1 \mid \mathbf{x}_*, \mathcal{D}) = \int \sigma(\mathbf{x}_*^\top \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w}.$$

This integral is intractable because the posterior $p(\mathbf{w} \mid \mathcal{D})$ is non-Gaussian, due to the non-linearity of the response function in the likelihood. To obtain a tractable approximation, we employ the Laplace approximation, which approximates the posterior distribution $p(\mathbf{w} \mid \mathcal{D})$ by a Gaussian centered at the mode. Specifically, the posterior is approximated as

$$p(\mathbf{w} \mid \mathcal{D}) \approx \mathcal{N}(\mathbf{w}_{\text{MAP}}, A^{-1}),$$

where $\mathbf{w}_{\text{MAP}}$ is the mode of the posterior distribution, the value of $\mathbf{w}$ that maximizes $\log p(\mathbf{w} \mid \mathcal{D})$, obtained by minimizing the negative log-posterior

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} \left[-\log p(\mathbf{w} \mid \mathcal{D}) \right],$$

and A is the matrix of second derivatives of the negative log-posterior, evaluated at $\mathbf{w}_{\text{MAP}}$,

$$A = \nabla^2 \left[-\log p(\mathbf{w} \mid \mathcal{D}) \right] \big|_{\mathbf{w}=\mathbf{w}_{\text{MAP}}}.$$

This matrix captures the local curvature of the posterior density near the mode and is positive definite, ensuring a valid Gaussian approximation with covariance A^{-1}.

Under this approximation, the predictive distribution becomes

$$p(y_* = +1 \mid \mathbf{x}_*, \mathcal{D}) \approx \sigma \left(\frac{\mathbf{x}_*^\top \mathbf{w}_{\text{MAP}}}{\sqrt{1 + \frac{\pi}{8} \mathbf{x}_*^\top A^{-1} \mathbf{x}_*}} \right),$$

where $\mathbf{x}_* \in R^d$ is the test input vector, $\sigma(\cdot)$ is the logistic sigmoid function defined as $\sigma(z) = \frac{1}{1+e^{-z}}$, and $\mathbf{x}_*^\top A^{-1} \mathbf{x}_*$ corresponds to the predictive variance of the latent function $f(\mathbf{x}_*) = \mathbf{x}_*^\top \mathbf{w}$ under the posterior distribution over the weights. The constant $\frac{\pi}{8}$ appears from an analytic approximation that allows integration of the sigmoid function over the Gaussian posterior.

This expression reflects both the mean prediction and the uncertainty in the weight distribution, offering a calibrated probabilistic prediction. Classification decisions can be made by applying a threshold to the predicted probability; for instance, one typically predicts $+1$ if $p(y = +1 \mid \mathbf{x}) \geq 0.5$, and -1 otherwise.

Extension to Multiclass Classification In the multiclass setting with C classes, a common approach is to introduce a separate weight vector $\mathbf{w}_c \in R^d$ for each class $c \in \{1, \ldots, C\}$, and define the latent score for class c as

$$f_c(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_c.$$

Likelihood. Given an input $\mathbf{x}$, the vector of scores $\{f_1(\mathbf{x}), \ldots, f_C(\mathbf{x})\}$ is passed through the softmax function (see Appendix .5 for details) to produce a valid probability distribution over classes

$$p(y = c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp(f_c(\mathbf{x}))}{\sum_{c'=1}^{C} \exp(f_{c'}(\mathbf{x}))},$$

where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_C] \in R^{d \times C}$ collects all weight vectors. This defines the multinomial (or multiclass) logistic regression likelihood.

Posterior. To perform Bayesian inference, a prior is placed independently on each weight vector, typically a zero-mean Gaussian

$$p(\mathbf{W}) = \prod_{c=1}^{C} \mathcal{N}(\mathbf{w}_c \mid \mathbf{0}, \Sigma_p).$$

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the posterior distribution over the weights becomes

$$p(\mathbf{W} \mid \mathcal{D}) \propto p(\mathbf{W}) \prod_{i=1}^{n} p(y_i \mid \mathbf{x}_i, \mathbf{W}).$$

Predictive distribution. To make a prediction for a new input $\mathbf{x}_*$, the predictive distribution is obtained by marginalizing over the posterior

$$p(y_* = c \mid \mathbf{x}_*, \mathcal{D}) = \int p(y_* = c \mid \mathbf{x}_*, \mathbf{W})\, p(\mathbf{W} \mid \mathcal{D})\, d\mathbf{W}.$$

As in the binary case, this integral is intractable and can be approximated using methods such as the Laplace approximation.

The final prediction is obtained by selecting the class with the highest predictive probability

$$\hat{y_*} = \arg\max_c \; p(y_* = c \mid \mathbf{x}_*, \mathcal{D}).$$

Geometric Interpretation Linear classifiers define a hyperplane as the decision boundary in the input space. This boundary corresponds to the set of points for which $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} = 0$. On either side of this hyperplane, the classifier assigns different class labels. The orientation of the hyperplane is determined by the direction of the weight vector $\mathbf{w}$, while its distance from the origin is affected by the inclusion of a bias term.

Limitations and Extensions Linear models are computationally efficient and often perform well in low-dimensional settings. However, when the true decision boundary is nonlinear, linear models may fail to capture the underlying class structure, leading to poor generalization. Nonetheless, Bayesian linear models provide a well-understood and interpretable framework, and serve as a building block for more complex methods such as Gaussian Processes Classification.

4.3 Gaussian Processes Classification

Gaussian Processes Classification extends the Gaussian Processes Regression framework to address problems involving categorical outputs.

Model Setup In the binary classification setting, the dataset consists of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where each input $\mathbf{x}_i \in R^d$ and the corresponding label $y_i \in \{-1, +1\}$.

A Gaussian Processes prior is placed over the latent function

$$f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')),$$

Given a set of training inputs $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$, the prior distribution over the latent vector $\mathbf{f} = [f_1, \ldots, f_n]^\top$ is multivariate Gaussian

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K),$$

where $K \in R^{n \times n}$ with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

To relate the latent values to the observed binary labels, a sigmoidal likelihood function is used

$$p(y_i \mid f_i) = \sigma(y_i f_i),$$

where σ is typically either the logistic sigmoid $\sigma(f) = \frac{1}{1+e^{-f}}$ or the probit function $\sigma(f) = \Phi(f)$, with Φ denoting the cumulative distribution function of the standard normal distribution.

Assuming conditional independence of the labels given the latent function, the likelihood of the entire dataset factorizes as

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^{n} \sigma(y_i f_i).$$

Posterior Distribution Bayesian inference proceeds by combining this likelihood with the GPs prior via Bayes' rule to obtain the posterior over the latent values

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f})\, \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)}{p(\mathbf{y} \mid X)},$$

where $p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^n \sigma(y_i f_i)$ is the likelihood, $\mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)$ is the GPs prior,

and $p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}) \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)\, d\mathbf{f}$ is the marginal likelihood ensuring normalization.

Predictive Distribution Given a test input $\mathbf{x}_* \in R^d$, we aim to compute the predictive distribution over the latent value f_*, and consequently the predictive class probability $\bar{\pi}_* = p(y_* = +1 \mid \mathbf{x}_*, \mathcal{D})$. Since the posterior is intractable, we approximate it using either Laplace approximation or Expectation Propagation, and propagate the result to obtain an approximation of the predictive distribution.

Laplace Approximation. The unnormalized log-posterior distribution is defined as

$$\Psi(\mathbf{f}) = \log p(\mathbf{y} \mid \mathbf{f}) - \frac{1}{2}\mathbf{f}^\top K^{-1}\mathbf{f}.$$

The term is called unnormalized because the log-marginal likelihood $\log p(\mathbf{y} \mid X)$ is omitted, being constant with respect to $\mathbf{f}$ and thus irrelevant for optimization.

Let $\hat{\mathbf{f}}$ denote the mode of Ψ. Define $W \in R^{n \times n}$ as the diagonal matrix with entries $W_{ii} = \sigma(f_i)(1-\sigma(f_i))$. Each diagonal element quantifies the local curvature of the log-likelihood at f_i, and thus encodes the amount of uncertainty contributed by the corresponding training point. The posterior over $\mathbf{f}$ is then approximated by the Gaussian

$$q(\mathbf{f}) = \mathcal{N}(\hat{\mathbf{f}}, \Sigma)$$

with $\Sigma = (K^{-1} + W)^{-1}$. Note that the notation $q(\mathbf{f})$ is used to emphasize that this is an approximation to the true posterior $p(\mathbf{f} \mid X, \mathbf{y})$, which is analytically intractable due to the non-Gaussian likelihood.

The predictive distribution over f_* is then obtained by conditioning the joint Gaussian prior between $\mathbf{f}$ and f_*, resulting in

$$f_* \mid \mathbf{x}_*, \mathcal{D} \sim \mathcal{N}(\bar{f}_*, \mathrm{Var}(f_*)),$$

with

$$\bar{f}_* = \mathbf{k}_*^\top K^{-1}\hat{\mathbf{f}}, \quad \mathrm{Var}(f_*) = k_{**} - \mathbf{k}_*^\top (K + W^{-1})^{-1}\mathbf{k}_*.$$

Finally, the predictive class probability is computed by marginalizing over f_*,

$$\bar{\pi}_* = \int \sigma(f_*)\, \mathcal{N}(f_* \mid \bar{f}_*, \mathrm{Var}(f_*))\, df_*,$$

The form of the integral depends on the specific choice of sigmoid function. When the probit function is used, the integral admits a closed-form expression $\bar{\pi}_* = \Phi\left(\frac{\bar{f}_*}{\sqrt{1+\mathrm{Var}(f_*)}}\right)$. In contrast, when the logistic sigmoid is adopted, the integral does not have an analytical solution and must be approximated numerically.

Expectation Propagation. Alternatively, the posterior distribution can be approximated using Expectation Propagation (EP) (see Appendix .10 for details) , which replaces each non-Gaussian likelihood factor $p(y_i \mid f_i)$ with a local Gaussian site function of the form $\tilde{t}_i(f_i) = \tilde{Z}_i \mathcal{N}(f_i \mid \tilde{\mu}_i, \tilde{\sigma}_i^2)$, where $\tilde{Z}_i$ is a normalizing constant, and $\tilde{\mu}_i$, $\tilde{\sigma}_i^2$ are site-specific parameters updated iteratively. The resulting approximate posterior takes the form

$$q(\mathbf{f}) = \frac{1}{Z_{\text{EP}}} \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K) \prod_{i=1}^{n} \tilde{t}_i(f_i) = \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$

where the posterior mean and covariance are given by

$$\Sigma = (K^{-1} + \tilde{\Sigma}^{-1})^{-1}, \quad \boldsymbol{\mu} = \Sigma \tilde{\Sigma}^{-1} \tilde{\boldsymbol{\mu}},$$

with $\tilde{\Sigma} = \text{diag}(\tilde{\sigma}_1^2, \ldots, \tilde{\sigma}_n^2)$ and $\tilde{\boldsymbol{\mu}} = [\tilde{\mu}_1, \ldots, \tilde{\mu}_n]^\top$.

In order to update each site approximation $\tilde{t}_i(f_i)$, EP proceeds by isolating the contribution of the i-th likelihood. This is done by forming the cavity distribution, defined as the marginal distribution over f_i obtained by removing $\tilde{t}_i(f_i)$ from the current approximate posterior. Formally, it is given by

$$q_{-i}(f_i) \propto \frac{q(f_i)}{\tilde{t}_i(f_i)} = \mathcal{N}(f_i \mid \mu_{-i}, \sigma_{-i}^2),$$

where μ_{-i} and σ_{-i}^2 are the cavity mean and variance. This distribution captures the influence of the prior and all likelihood terms except the i-th one. It serves as a temporary baseline for evaluating how well the current site $\tilde{t}_i$ matches the exact likelihood $p(y_i \mid f_i)$.

The cavity distribution is then multiplied with the true likelihood to form the unnormalized tilted distribution

$$q_{\text{new}}(f_i) \propto q_{-i}(f_i)\, p(y_i \mid f_i).$$

Since this tilted distribution is not Gaussian, it is approximated by a new Gaussian distribution that matches its first two moments. This moment-matching step determines the updated site parameters $\tilde{\mu}_i$ and $\tilde{\sigma}_i^2$, and the new site function $\tilde{t}_i(f_i)$ is recomputed accordingly. The updated site is then reintegrated into the posterior, and the process is repeated iteratively across all sites until convergence.

To conclude we obtain the predictive distribution over f_*, by conditioning on the joint Gaussian prior between $\mathbf{f}$ and f_*, and using the EP approximation

$$f_* \mid \mathbf{x}_*, \mathcal{D} \sim \mathcal{N}(\bar{f}_*, \text{Var}(f_*)),$$

where

$$\bar{f}_* = \mathbf{k}_*^\top (K + \tilde{\Sigma})^{-1} \tilde{\boldsymbol{\mu}}, \quad \text{Var}(f_*) = k_{**} - \mathbf{k}_*^\top (K + \tilde{\Sigma})^{-1} \mathbf{k}_*.$$

The predictive class probability is then computed by marginalizing over f_*,

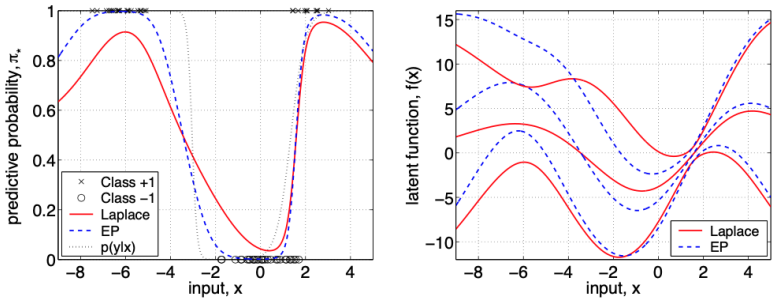
$$\bar{\pi}_* = \int \sigma(f_*)\, \mathcal{N}(f_* \mid \bar{f}_*, \text{Var}(f_*))\, df_*,$$

where the function σ determines whether the integral is analytically tractable.

Comparison of Laplace and Expectation Propagation. Both the Laplace approximation and Expectation Propagation approximate the intractable posterior distribution $p(\mathbf{f} \mid \mathcal{D})$ with a multivariate Gaussian, but they adopt fundamentally different strategies. Laplace centers the approximation at the mode of the unnormalized log-posterior and uses the local curvature at that point to define the covariance structure. In contrast, EP iteratively refines a global Gaussian approximation by matching the moments of each marginal likelihood factor, using a cavity distribution that excludes the current data point.

This difference has significant implications. While Laplace relies on second-order information at a single point and is relatively fast to compute, it may poorly approximate posteriors that are far from Gaussian, particularly in the tails. EP, by contrast, tends to yield better predictive probabilities, especially in scenarios involving class imbalance or multimodal posteriors, since it distributes the approximation effort across the entire dataset.

From a computational perspective, Laplace is typically faster and easier to implement, making it more suitable for large datasets. EP is more computationally expensive due to the iterative moment-matching process, but often provides more accurate predictive performance on small to medium sized problems.



(a) (b)

Figure 14: One-dimensional toy classification dataset. Panel (a) shows the dataset, where points from class $+1$ have been plotted at $\pi = 1$ and class -1 at $\pi = 0$, together with the predictive probability for Laplace's method and the EP approximation. Also shown is the probability $p(y = +1 \mid \mathbf{x})$ of the data generating process. Panel (b) shows the corresponding distribution of the latent function $f(\mathbf{x})$, the central curves (red for Laplace, blue for EP) represent the predictive means, while the outer curves show the corresponding uncertainty regions given by the mean ± 2 standard deviations (95% confidence intervals). Adapted from *Gaussian Processes for Machine Learning* (Rasmussen and Williams, 2006).

4.4 Gaussian Processes Classification: Multiclass Laplace Approximation

Gaussian Processes Classification can be extended to problems involving more than two classes by introducing a latent function for each class. In the multiclass case with C classes, the model maintains the nonparametric nature of Gaussian Processes, but introduces a more complex structure in the latent space and likelihood. In particular, the likelihood is defined using a softmax function. Since the posterior is analytically intractable, we employ the Laplace approximation, generalizing the approach used in the binary setting. The result is a Gaussian approximation to the posterior, from which predictive distributions and class probabilities can be derived.

Model Setup In the multiclass classification setting, the output space consists of C classes, and the label associated with input $\mathbf{x}_i \in R^d$ is denoted by $y_i \in \{1, \ldots, C\}$. For each class $c \in \{1, \ldots, C\}$, a latent function $f^c(\cdot)$ is defined, and the vector of latent function values at $\mathbf{x}_i$ is $\mathbf{f}_i = [f^1(\mathbf{x}_i), \ldots, f^C(\mathbf{x}_i)]^\top \in R^C$, which is assigned a Gaussian Processes prior

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K),$$

where $K \in R^{Cn \times Cn}$ is block-diagonal, with each block $K^c \in R^{n \times n}$ given by the kernel function $k^c(\mathbf{x}_i, \mathbf{x}_j)$ for class c.

The likelihood is defined through the softmax function. Let $\mathbf{f}_i = [f_i^1 = f^1(\mathbf{x}_i), \ldots, f_i^C = f^C(\mathbf{x}_i)]^\top$, then the probability of assigning label $y_i = c$ is modeled as

$$p(y_i = c \mid \mathbf{f}_i) = \pi_i^c = \frac{\exp(f_i^c)}{\sum_{c'=1}^C \exp(f_i^{c'})},$$

where the softmax maps latent function values into a categorical probability distribution. Since the latent vectors $\mathbf{f}_1, \ldots, \mathbf{f}_n$ are assumed independent a priori, and the likelihood factors across data points, the overall likelihood over the dataset factorizes as

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^n \pi_i^{y_i},$$

where each term $\pi_i^{y_i}$ denotes probability corresponding to the observed class label y_i for input $\mathbf{x}_i$.

By Bayes' theorem, the posterior over the latent variables is given by

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f})\mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)}{p(\mathbf{y} \mid X)},$$

where $p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^n \pi_i^{y_i}$, $\mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)$ is the Gaussian Processes prior, and $p(\mathbf{y} \mid X)$ is the marginal likelihood.

Theorem 3. *Let $\hat{\mathbf{f}} \in R^{Cn}$ be the mode of the log-posterior distribution, and define the probability vector $\boldsymbol{\pi} \in R^{Cn}$ evaluated at $\hat{\mathbf{f}}$, with block structure $\boldsymbol{\pi} = [\pi^1; \ldots; \pi^C]$ where $\pi^c \in R^n$.*

Let $\mathbf{x}_* \in R^d$ be a test input and $\mathbf{f}_* = [f_*^1, \ldots, f_*^C]^\top \in R^C$ the vector of latent values at $\mathbf{x}_*$. Then, under the Laplace approximation, the predictive distribution over $\mathbf{f}_*$ is Gaussian,

$$\mathbf{f}_* \mid \mathbf{x}_*, \mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*),$$

with mean and covariance given by

$$\boldsymbol{\mu}_* = Q_*^\top(\mathbf{y} - \boldsymbol{\pi}), \quad \Sigma_* = \operatorname{diag}(k(\mathbf{x}_*, \mathbf{x}_*)) - Q_*^\top(K + W^{-1})^{-1}Q_*,$$

Analysing the term of the equation, $Q_* \in R^{Cn \times C}$ is the block matrix of covariances between training points and $\mathbf{x}_*$, $W \in R^{Cn \times Cn}$ is the negative Hessian of the log-posterior at $\hat{\mathbf{f}}$, $\mathbf{y} \in \{0,1\}^{Cn}$ is the one-hot encoding of the class labels, $\boldsymbol{\pi} \in R^{Cn}$ stacks the class probability vectors for the training inputs, and $\mathrm{diag}(k(\mathbf{x}_*, \mathbf{x}_*)) \in R^{C \times C}$ denotes the diagonal matrix built from the prior variances of the test point.

Proof. The unnormalized log-posterior is given by

$$\Psi(\mathbf{f}) = -\frac{1}{2}\mathbf{f}^\top K^{-1}\mathbf{f} + \mathbf{y}^\top \mathbf{f} - \sum_{i=1}^{n} \log \sum_{c=1}^{C} \exp(f_i^c),$$

where $\mathbf{f} \in R^{Cn}$, $K \in R^{Cn \times Cn}$, $\mathbf{y} \in R^{Cn}$ and f_i^c denotes the latent value for input $\mathbf{x}_i$ and class c. The first term corresponds to the Gaussian prior on the latent variables, the second term represents the contribution of the likelihood through the observed class labels and the third term arises from the softmax normalization and ensures proper normalization across classes.

Differentiating Ψ gives

$$\nabla\Psi = -K^{-1}\mathbf{f} + \mathbf{y} - \boldsymbol{\pi},$$

Setting the gradient to zero yields the MAP estimate

$$\hat{\mathbf{f}} = K(\mathbf{y} - \hat{\boldsymbol{\pi}}),$$

where $\hat{\boldsymbol{\pi}}$ denotes the probabilities evaluated at $\hat{\mathbf{f}}$.

The Hessian (second derivative) of Ψ at $\hat{\mathbf{f}}$ is

$$\nabla\nabla\Psi = -K^{-1} - W,$$

where $W = \mathrm{diag}(\pi) - \pi\pi^\top \in R^{Cn \times Cn}$ is the generalized Fisher information matrix (see Appendix .6 for details). Here, $\pi \in R^{Cn \times C}$ is the matrix obtained by stacking the class probability vectors π^c, and $\mathrm{diag}(\pi) \in R^{Cn \times Cn}$ is the block-diagonal matrix whose diagonal blocks are built from the class probability vectors. The matrix W is positive definite, ensuring concavity of Ψ.

Thus, the Laplace approximation gives

$$q(\mathbf{f}) = \mathcal{N}(\hat{\mathbf{f}}, \Sigma).$$

where $\Sigma = (K^{-1} + W)^{-1} \in R^{Cn \times Cn}$ is the covariance of the approximate posterior.

Let $Q_* \in R^{Cn \times C}$ contain the cross-covariances between the test point $\mathbf{x}_*$ and the training points for each class,

$$Q_* = \begin{bmatrix} k^1(\mathbf{x}_*) \\ \vdots \\ k^C(\mathbf{x}_*) \end{bmatrix},$$

where each $k^c(\mathbf{x}_*) \in R^n$ has entries $[k^c(\mathbf{x}_*)]_i = k^c(\mathbf{x}_i, \mathbf{x}_*)$.

Then, using standard conditioning formulas for Gaussians (joint Gaussian prior), the predictive distribution becomes

$$\mathbf{f}_* \mid \mathcal{D}, \mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*),$$

where

$$\boldsymbol{\mu}_* = Q_*^\top K^{-1}\hat{\mathbf{f}} = Q_*^\top(\mathbf{y} - \hat{\boldsymbol{\pi}}), \quad \Sigma_* = \mathrm{diag}(k(\mathbf{x}_*, \mathbf{x}_*)) - Q_*^\top (K + W^{-1})^{-1} Q_*.$$

Finally, the predictive class probabilities are computed by integrating the softmax function over the Gaussian Processes distribution

$$\bar{\boldsymbol{\pi}}_* = E_{\mathbf{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*)}[\text{softmax}(\mathbf{f}_*)],$$

where $\text{softmax}(\mathbf{f}_*)_c = \frac{\exp(f_*^c)}{\sum_{c'=1}^{C} \exp(f_*^{c'})}$. Since this expectation is analytically intractable due to the nonlinearity of the softmax, it is typically approximated via Monte Carlo sampling (see Appendix .7 for details)

$$\bar{\boldsymbol{\pi}}_* \approx \frac{1}{S} \sum_{s=1}^{S} \text{softmax}(\mathbf{f}_*^{(s)}),$$

where $\mathbf{f}_*^{(s)} \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*)$ are independent samples drawn from the predictive Gaussian Processes distribution.

4.5 Real World Case

In this section, a real world case for Gaussian Processes Classification is presented.

4.5.1 Gaussian Processes Classification on Breast Cancer Dataset

We apply Gaussian Processes Classification to the Breast Cancer Wisconsin dataset. The goal is to classify tumors as either malignant or benign based on various numerical features extracted from digitized images of fine needle aspirates. The dataset is publicly available from the UCI Machine Learning Repository and contains 569 instances and 31 real-valued input features.

The binary labels are encoded as $y_i \in \{0, 1\}$, where 1 denotes malignant and 0 benign. The inputs are standardized before training.

Laplace Approximation We first train a GPC model using the Laplace approximation on 400 training samples.

When evaluated on the test set (169 samples), the model achieved an accuracy of 95.9%. The ROC curve in Figure 15 shows that the classifier maintains high true positive rates across almost all decision thresholds, with minimal false positives. The curve remains close to the top-left corner of the plot, resulting in an AUC (see Appendix .9) of 0.98. This indicates excellent discriminative performance, as the model is able to confidently distinguish between benign and malignant tumors in most cases.

Expectation Propagation (EP) We also implement Expectation Propagation on the same test set, the EP-based classifier achieved a slightly lower accuracy of 93.5% and an AUC of 0.96. As shown in Figure 16, the EP-based classifier also demonstrates strong classification ability, however, compared to Laplace, the curve is slightly less steep near the origin. This suggests that while EP still performs very well, its predictions are slightly more conservative in pushing scores away from the decision boundary.

Comparison with Bayesian Logistic Regression To provide a baseline, we also trained a standard Bayesian Logistic Regression classifier on the same dataset. This linear model reached an accuracy of 97.0% and an AUC of 1.00. Its ROC curve (Figure 17) lies almost perfectly against the top-left corner, showing near-optimal discrimination.

Interpretation Both Laplace and EP approximations yielded high predictive performance on this binary classification task. The Laplace approximation slightly outperformed EP in terms of AUC, however, EP remains a competitive method. Bayesian Logistic Regression achieved the highest AUC in this case, reflecting the linear separability of the dataset.

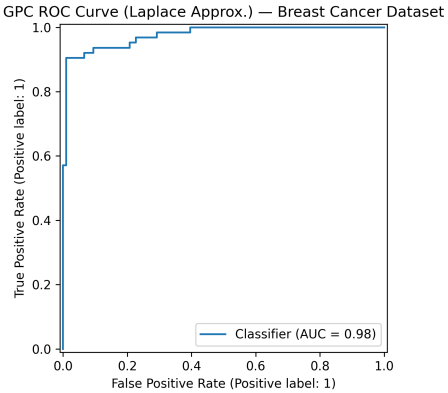


Figure 15: ROC curve for GPC with Laplace approximation.

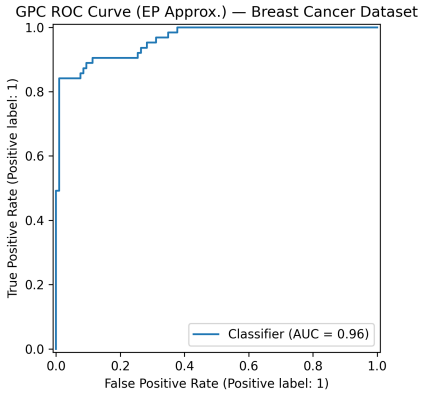


Figure 16: ROC curve for GPC with EP approximation.

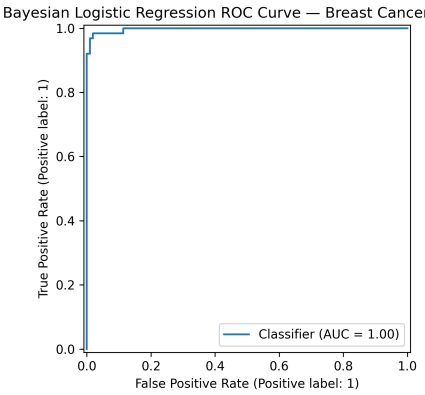


Figure 17: ROC curve for Bayesian Logistic Regression.

Predictive Variance Analysis Figure 18 illustrates the predictive variance surfaces obtained with Expectation Propagation (left) and Laplace approximation (right), projected onto the first two principal components of the feature space.

The summary statistics highlight a substantial difference between the two approximations:

- **EP:** Mean variance 0.784 ± 0.274, with a wide range from 0.063 to 1.000. This indicates that EP produces a more diverse distribution of uncertainty values, strongly distinguishing between regions of high confidence (low variance near clusters of training points) and high uncertainty (far from the data).
- **Laplace:** Mean variance 0.958 ± 0.062, ranging from 0.647 to 1.000. The Laplace approximation yields consistently higher variance values, with less spread. This suggests that Laplace tends to be more conservative, assigning substantial uncertainty almost everywhere, even near regions densely populated by training points.

From a qualitative perspective, the EP variance map exhibits a structured pattern where uncertainty is lowest inside the benign and malignant clusters (light regions, corresponding to low variance), reflecting strong data support, and it increases smoothly when moving away from the data (darker areas).

In contrast, the Laplace variance map appears more uniform, with a generally higher baseline variance (darker overall background) and less contrast between confident and uncertain regions. This flatter representation suggests that Laplace provides a more cautious but less detailed account of predictive uncertainty.

Consequently, EP yields a more nuanced characterization of uncertainty across the input space, whereas Laplace tends to produce a conservative but less informative variance.

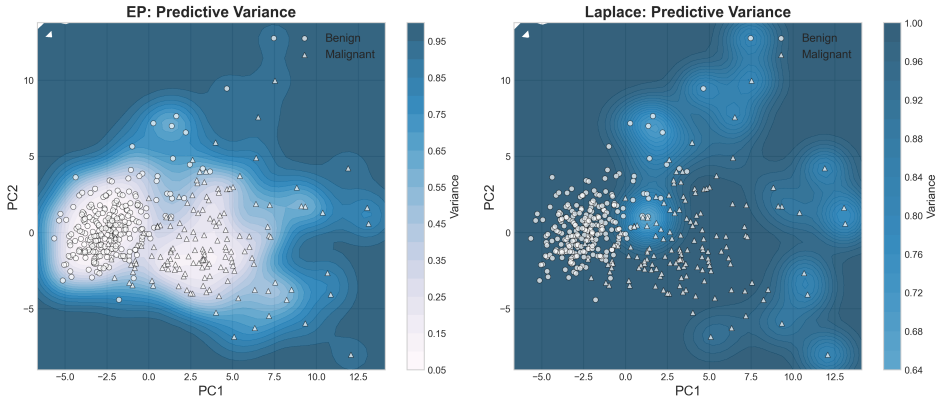


Figure 18: Predictive variance maps for GPC using EP (left) and Laplace (right).

5 Simulation-Based Calibration for validating Gaussian Processes inference

We now introduce a procedure to validate Gaussian Processes inference.

Simulation-Based Calibration (SBC) is a diagnostic procedure for validating Bayesian inference algorithms and is founded on the self-consistency property of the Bayesian joint distribution. This property states that, under exact Bayesian inference, the posterior distribution averaged over all possible datasets generated from the model coincides exactly with the prior distribution. In other words, if the model is correctly specified and the computation is accurate, the variability of the posterior across many simulated datasets will reflect precisely the variability encoded in the prior. Any systematic discrepancy between these two indicates either a modelling error or an inaccuracy in the inference algorithm. We now formalise this idea and present the formulas on which SBC is based.

The SBC procedure begins by drawing synthetic pairs $(\tilde{\theta}, \tilde{y})$ from the joint distribution $p(\theta, y)$, obtained by combining the prior $p(\theta)$ with the likelihood $p(y \mid \theta)$.

$$(\tilde{\theta}, \tilde{y}) \sim p(\theta, y) = p(y \mid \theta)\, p(\theta). \tag{1}$$

Given the simulated data $\tilde{y}$, the posterior distribution is computed via Bayes' theorem as

$$p(\theta \mid \tilde{y}) \;\propto\; p(\tilde{y} \mid \theta)\, p(\theta), \tag{2}$$

where the proportionality symbol indicates equality up to the marginal likelihood $p(\tilde{y})$.

If the inference is exact, the following self–consistency identity holds

$$p(\theta) \;=\; \iint p(\theta \mid \tilde{y})\, p(\tilde{y} \mid \tilde{\theta})\, p(\tilde{\theta})\, d\tilde{y}\, d\tilde{\theta}, \tag{3}$$

where $p(\tilde{\theta})$ is the prior over the simulated parameter, $p(\tilde{y} \mid \tilde{\theta})$ is the likelihood generating the synthetic dataset, and $p(\theta \mid \tilde{y})$ is the posterior distribution given the synthetic dataset.

This identity follows directly from the laws of probability. In fact, the term $p(\tilde{y} \mid \tilde{\theta})p(\tilde{\theta})$ represents the joint distribution of the simulated parameter and data, and integrating it over $\tilde{\theta}$ yields the marginal distribution of the synthetic dataset,

$$\int p(\tilde{y} \mid \tilde{\theta})p(\tilde{\theta})\, d\tilde{\theta} = p(\tilde{y}).$$

Substituting this result back into equation (3) gives

$$\int p(\theta \mid \tilde{y})p(\tilde{y})\, d\tilde{y} = p(\theta),$$

which shows that the prior distribution is recovered.

The self–consistency property implies a rank–uniformity result. Let $h : \Theta \to R$ be any scalar functional of θ, for each simulated dataset $\tilde{y}$, one draws L posterior samples

$$\{\theta^{(1)}, \ldots, \theta^{(L)}\} \sim p(\theta \mid \tilde{y}), \tag{4}$$

and computes the rank statistic

$$r = \#\left\{\ell \in \{1, \ldots, L\} \;:\; h(\theta^{(\ell)}) < h(\tilde{\theta})\right\} \quad \in \quad \{0, \ldots, L\}, \tag{5}$$

where # denotes the number of posterior draws smaller than the true simulated value $h(\tilde{\theta})$. Repeating this process independently N times produces a sequence of ranks $r_1, \ldots, r_N$ which, under exact inference

and a correctly specified model, are distributed as a discrete uniform distribution over $\{0, \ldots, L\}$. This uniformity arises because, if the posterior distribution is consistent with the prior distribution, then the true simulated value $h(\tilde{\theta})$ is exchangeable with the posterior draws $\{h(\theta^{(1)}), \ldots, h(\theta^{(L)})\}$. As a consequence, the rank of $h(\tilde{\theta})$ among these $L+1$ values is equally likely to fall in any of the $L+1$ possible positions.

The SBC diagnostic is based on the histogram of the ranks. A flat histogram indicates that the posterior is correctly calibrated. A U-shaped histogram indicates that the posterior is underdispersed, meaning it is too concentrated and thus overconfident. An inverted U-shaped histogram indicates that the posterior is overdispersed, meaning it is too wide and thus underconfident. A skewed histogram indicates that the posterior is biased, revealing a systematic shift from the truth.

5.1 Simulation-Based Calibration for Gaussian Processes

In Gaussian Processes models, the latent variable θ corresponds to the latent function $f(\mathbf{x})$. SBC can be applied by defining f in the equation of rank as the pointwise evaluation $f_\star = f(\mathbf{x}_\star)$ at one or more fixed test inputs $\mathbf{x}_\star$. The rank uniformity test is then applied directly to these function values, comparing the posterior draws with the corresponding ground-truth values used to generate the synthetic dataset.

According to this, the Simulation Based Calibration methodology extends naturally to Gaussian Processes by treating parameter draws as function values evaluated at fixed inputs.

In this work, we assume fixed hyperparameters θ for simplicity, although SBC can also be extended to account for hyperparameter uncertainty by integrating over their posterior distribution. In addition, we consider a Gaussian Process prior $f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$, and training inputs $\mathbf{x} \in R^d$ and test inputs $\mathbf{x}_* \in R^d$. The corresponding function values are collected in $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]^\top$ at the training points and $f_* = f(\mathbf{x}_*)$ at the test points. Under the prior, the joint law of $(\mathbf{f}, f_*)$ is multivariate normal with

$$\begin{pmatrix} \mathbf{f} \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} m_{\mathbf{x}} \\ m_* \end{pmatrix}, \begin{pmatrix} K_{\mathbf{xx}} & K_{\mathbf{xx}_*} \\ K_{\mathbf{x}_*\mathbf{x}} & K_{\mathbf{x}_*\mathbf{x}_*} \end{pmatrix} \right),$$

where $m_{\mathbf{x}} = m(\mathbf{x})$ denotes the prior mean vector evaluated at the training inputs $\mathbf{x}$, $m_* = m(\mathbf{x}_*)$ denotes the prior mean vector evaluated at the test inputs $\mathbf{x}_*$, $K_{\mathbf{xx}} = k(\mathbf{x}, \mathbf{x})$ is the covariance matrix between training inputs, $K_{\mathbf{xx}_*} = k(\mathbf{x}, \mathbf{x}_*)$ is the cross-covariance matrix between training and test inputs, $K_{\mathbf{x}_*\mathbf{x}} = k(\mathbf{x}_*, \mathbf{x})$ is the cross-covariance matrix between test and training inputs (the transpose of $K_{\mathbf{xx}_*}$), and $K_{\mathbf{x}_*\mathbf{x}_*} = k(\mathbf{x}_*, \mathbf{x}_*)$ is the covariance matrix between test inputs.

The following representation provides the foundation for the Gaussian Processes Simulation–Based Calibration (GP–SBC) procedure. At each iteration, ground–truth function values $(\tilde{\mathbf{f}}, \tilde{f}_*)$ are sampled from this joint prior. Synthetic observations $\tilde{y}$ are then generated from the likelihood model $p(\tilde{y} \mid \tilde{\mathbf{f}}, \theta)$. GPs inference method is then applied to the simulated dataset $(\mathbf{x}, \tilde{y})$ to obtain the posterior distribution over the latent function values at the test inputs,

$$p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*, \theta).$$

The GPs data-averaged posterior, so called because it averages the posterior distribution across datasets sampled from the prior, can be expressed as

$$p(f_* \mid \mathbf{x}, \mathbf{x}_*) \;=\; \iint p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*, \theta)\, p(\tilde{y} \mid \tilde{\mathbf{f}}, \mathbf{x}, \theta)\, p(\tilde{\mathbf{f}} \mid \mathbf{x}, \theta)\, d\tilde{y}\, d\tilde{\mathbf{f}} \tag{6}$$

where $p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*, \theta)$ is the posterior distribution of the latent function values at the test inputs $\mathbf{x}_*$ given the simulated dataset $(\mathbf{x}, \tilde{y})$ and hyperparameters θ, $p(\tilde{y} \mid \tilde{\mathbf{f}}, \mathbf{x}, \theta)$ is the likelihood of the synthetic observations and $p(\tilde{\mathbf{f}} \mid \mathbf{x}, \theta)$ is the GPs prior distribution.

From the data-averaged posterior (6), L independent samples $f_*^{(1)}, \ldots, f_*^{(L)}$ are drawn. For each test location $\mathbf{x}_j^*$, the rank statistic

$$r_j = \#\left\{\ell \in \{1, \ldots, L\} \,:\, f_*^{(\ell)}(\mathbf{x}_j^*) < \tilde{f}_*(\mathbf{x}_j^*)\right\} \quad \in \quad \{0, \ldots, L\}$$

is computed by comparing the ground truth value $\tilde{f}_*(\mathbf{x}_j^*)$ with the posterior samples at the same input.

An important aspect of the GP–SBC procedure concerns the treatment of the hyperparameters θ. The choice of whether to keep θ fixed or to treat it as a random variable depends on the specific validation objective.

When the purpose is to verify the correctness of the inference for the latent function values conditional on known hyperparameters, a fixed vector θ is used in all replications. This configuration isolates the calibration of

$$p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*, \theta)$$

and ensures that any systematic deviation from uniformity in the SBC ranks can be attributed solely to errors in the posterior computation for the latent process, without the confounding effect of hyperparameter uncertainty.

Alternatively, when the aim is to validate the full Bayesian model, including uncertainty over θ, the generative step of SBC is extended. At each replication, a hyperparameter vector $\tilde{\theta}$ is first drawn from its prior distribution $p(\theta)$. Conditional on this draw, synthetic latent values $(\mathbf{f}, \tilde{f}_*)$ and observations $\tilde{y}$ are generated. The inference step then targets the posterior over latent function values and integrating over the posterior distribution of the hyperparameters we have

$$p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*) \;=\; \int p(f_* \mid \tilde{y}, \mathbf{x}, \mathbf{x}_*, \theta)\, p(\theta \mid \tilde{y})\, d\theta,$$

where $p(\theta \mid \tilde{y})$ is the posterior distribution of the hyperparameters.

In this second setting, deviations from uniform rank distributions may arise from inaccuracies in modelling either the latent function or the hyperparameters, as both contribute to the posterior uncertainty.

5.2 SBC evaluation in Gaussian Processes Regression and Classification

This section presents experiment of Simulation-Based Calibration in Gaussian Processes models. In the regression setting, exact samples from the posterior can be obtained, so SBC is expected to perform optimally. By contrast, in the classification setting an approximation to the posterior is required, and the validity of SBC is not guaranteed a priori.

5.3 SBC in Gaussian Processes Regression

We begin with Gaussian Processes Regression and then extend the discussion to classification.

5.3.1 Synthetic experiment

We evaluate GP–SBC procedure in a one dimensional synthetic setting. The training input set

$$X = \{x_i\}_{i=1}^n \subset [-2, 2]$$

is fixed on a uniform grid of $n = 25$ points, while the test set consists of a single location $x_* = 0$ placed exactly at the center of the interval.

The latent function $f(\mathbf{x})$ is assumed to follow a Gaussian Processes prior with squared exponential kernel while observations are subject to additive Gaussian noise σ^2.

In the reported run, the hyperparameters are fixed to $\ell = 0.5$, $\sigma_f^2 = 1.0$, $\sigma^2 = 0.02$, with $L = 50$ posterior draws per replication and $N = 800$ replications in total. The resulting histogram (Figure 19) is visually flat within the expected variability, and the empirical mean rank is 24.5, extremely close to the theoretical value $L/2 = 25$. This agreement confirms that, in this setting, the GPs regression pipeline is correctly implemented and yields calibrated posterior predictions.

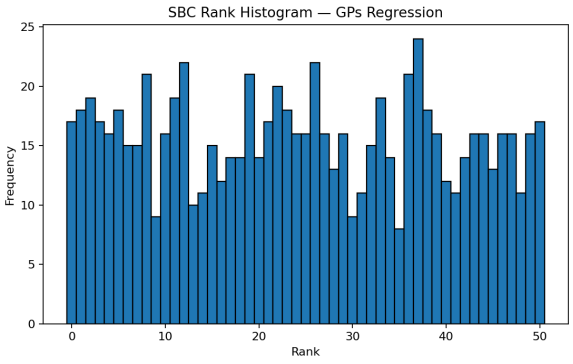


Figure 19: Rank histogram for the synthetic GP–SBC experiment with GPR

5.4 SBC in Gaussian Processes Classification

This section investigates Simulation-Based Calibration in Gaussian Processes Classification, comparing two approximate inference schemes: the Laplace approximation and Expectation Propagation. Two synthetic experiments with different input ranges are considered to enable a deeper and more robust comparison. A novel contribution is the use of SBC as a diagnostic tool to evaluate and compare the calibration of these two inference methods.

In general, we adopt a Gaussian Processes prior over the latent function $f(\mathbf{x})$ with squared exponential kernel and apply the probit likelihood $\Phi(f)$. For a single test location $\mathbf{x}_*$, the posterior distribution over the latent value f_* is approximated using either the Laplace method or Expectation Propagation. Then the posterior distribution of the latent function value at a new test input $\mathbf{x}_*$ is given by

$$p(f_* \mid \mathbf{x}_*, \mathbf{x}, \hat{\mathbf{f}}) \;=\; \mathcal{N}(\bar{f}_*, \mathrm{Var}(f_*)).$$

where the posterior mean and variance are given by

$$\bar{f}_* = \mathbf{k}_*^\top K^{-1}\hat{\mathbf{f}}, \qquad \mathrm{Var}(f_*) = k_{**} - \mathbf{k}_*^\top (K + W^{-1})^{-1}\mathbf{k}_*.$$

in the Laplace case (with W the negative Hessian of the log likelihood), and

$$\bar{f}_* = \mathbf{k}_*^\top (K + \tilde{\Sigma})^{-1}\tilde{\mu}, \qquad \mathrm{Var}(f_*) = k_{**} - \mathbf{k}_*^\top (K + \tilde{\Sigma})^{-1}\mathbf{k}_*.$$

in EP case.

To conclude L posterior samples per replication are generated and the SBC rank statistic is defined as

$$r = \#\left\{\ell \in \{1, \ldots, L\} : f_*^{(\ell)} < \tilde{f}_*\right\} \in \{0, \ldots, L\}.$$

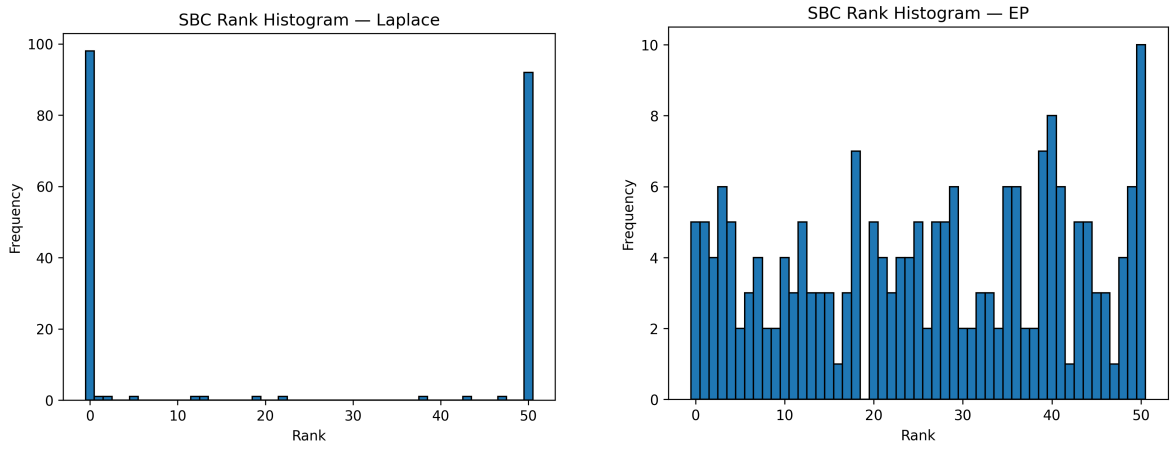
5.4.1 First Synthetic experiment

The first experiment is conducted with the training inputs $X = \{x_i\}_{i=1}^n$ equally spaced on a uniform grid over the interval $[-2, 2]$ with $n = 25$, while a single test location is fixed at the centre of the domain, $x_* = 0$. The latent function $f(\mathbf{x})$ follows Gaussian Processes prior with squared exponential kernel and hyperparameters $\ell = 0.5$ and $\sigma_f^2 = 1.0$. The binary responses are generated according to a probit likelihood,

$$y_i \sim \text{Bernoulli}\big(\Phi(f(x_i))\big),$$

In each replication, a joint prior draw $(\tilde{\mathbf{f}}, \tilde{f}_*)$ is obtained, from which synthetic responses $\tilde{y}$ are simulated. The model is then fitted using either the Laplace approximation or Expectation Propagation, producing an approximate Gaussian Processes posterior distribution for the latent value at $\mathbf{x}_*$.

From this approximate distribution, $L = 50$ independent posterior draws are generated. The SBC rank statistic is then computed. The experiment is repeated for $N = 200$ independent replications, and the collection of rank values is used to build the histograms shown in Figure 20. The EP approximation produces a histogram that is essentially flat, with empirical mean rank of 26.20 (theoretical 25.00), consistent with correct calibration. Conversely, the Laplace approximation yields a pronounced U–shaped distribution with a concentration of ranks near the boundaries 0 and L, and an empirical mean rank of 24.01. Although both empirical means are relatively close to $L/2$, the EP results exhibit much less deviation from uniformity in the histogram, indicating that EP is better calibrated than Laplace in this setting.



(a) Laplace approximation

(b) EP approximation

Figure 20: SBC rank histograms with GPC for first synthetic data experiment under Laplace and EP approximations

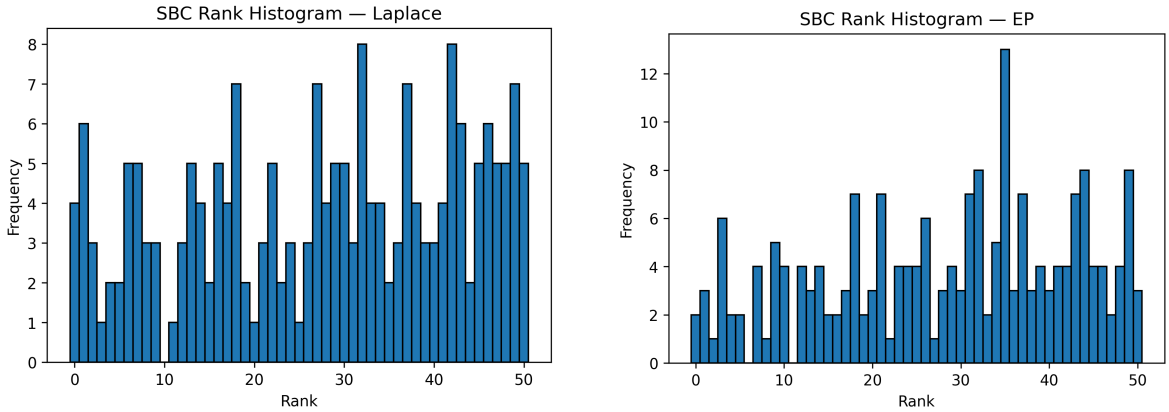
5.4.2 Second Synthetic experiment

The second experiment is conducted with the training inputs $X = \{x_i\}_{i=1}^n$ equally spaced on a uniform grid over the wider interval $[-30, 30]$ with $n = 25$, while again using a single test location fixed at the centre of the domain, $x_* = 0$. The latent function $f(\mathbf{x})$ follows a Gaussian Process prior with squared exponential kernel and hyperparameters $\ell = 0.5$ and $\sigma_f^2 = 1.0$. The binary responses are generated according to a probit likelihood,

$$y_i \sim \text{Bernoulli}(\Phi(f(x_i))).$$

The inference and SBC procedure are the same as in the first experiment: for each replication, a joint prior draw $(\tilde{f}, \tilde{f}_*)$ is obtained, synthetic responses $\tilde{y}$ are generated, the model is fitted with either the Laplace approximation or Expectation Propagation, and $L = 50$ posterior draws are taken to compute the SBC rank statistic. The experiment is repeated for $N = 200$ independent replications, and the collection of rank values is used to build the histograms shown in Figure 21.

The results show that, unlike in the narrow range design, the Laplace approximation no longer exhibits a pronounced U–shape. With more widely spread input points, the differences between the two approximations become less evident in the rank histograms. Nevertheless, the empirical mean ranks differ substantially from previous experiment, Laplace yields 27.64 (theoretical 25.00), while EP gives 28.35.



(a) Laplace approximation

(b) EP approximation

Figure 21: SBC rank histograms with GPC for second synthetic data experiment under Laplace and EP approximations

In summary, the experiments reveal a nuanced behaviour. In the restricted input domain, Expectation Propagation delivers posterior distributions that are better calibrated than those of the Laplace approximation, which shows clear signs of overconfidence and miscalibration. This outcome can be explained by the fact that, with densely spaced training points, the model receives highly concentrated information and the probit likelihood becomes strongly non–Gaussian, exposing the limitations of the Laplace approximation. However, when the input range is expanded while keeping the number of training points fixed, the observations become sparser and the influence of each observation is weaker. As a consequence, the posterior distributions are closer to Gaussian in shape, which makes the Laplace approximation less prone to miscalibration. In this setting, the strong U–shaped distortion disappears and the differences between the two methods diminish, with both achieving comparable performance though with slightly different mean ranks.

Overall, the SBC results suggest that Expectation Propagation provides a more robust approximation than Laplace across different input designs.

These findings illustrate that Simulation–Based Calibration can highlight not only systematic miscalibration but also how the relative behaviour of inference schemes depends critically on the experimental design, and in particular on the density and distribution of the input locations.

6 Conclusion

This thesis has provided a comprehensive examination of Gaussian Processes from both theoretical and practical perspectives, with particular emphasis on their role in regression and classification tasks.

The empirical analyses confirmed that Gaussian Processes Regression performs comparably to linear methods in settings where the underlying data structure is simple and approximately linear, while offering clear advantages in more complex and nonlinear contexts. A key strength of Gaussian Processes lies in the flexibility provided by the choice of kernel function, which acts as a prior over the function space and determines the smoothness, periodicity, and other structural properties of the learned mapping. This kernel-based adaptability enables the Gaussian Processes models to adapt to different input structures and data patterns.
Equally important is the ability of Gaussian Processes to provide heteroscedastic predictive intervals, which adapt the width of uncertainty bands according to the local density and variability of the observations. This feature proved particularly valuable in the Concrete Strength prediction task, where GPR not only achieved higher predictive accuracy but also delivered more informative uncertainty quantification. The resulting predictive intervals conveyed not just the expected outcome but also the reliability of each prediction, thereby offering an additional layer of interpretability and robustness in decision-making contexts.

In Classification, the empirical comparison highlights a key trade-off between predictive accuracy and uncertainty quantification. Bayesian Logistic Regression achieved the highest discriminative performance on the Breast Cancer dataset, with an AUC equal to 1.0, reflecting the almost linear separability of the data. However, despite its excellent accuracy, Bayesian Logistic Regression does not provide a data-dependet uncertainty. In contrast, Gaussian Processes Classification with Laplace and Expectation Propagation approximations yielded slightly lower AUC scores (0.98 and 0.96, respectively), yet offered richer insights into predictive uncertainty. The Laplace approximation tended to produce more conservative variance estimates, while Expectation Propagation provided sharper and more informative uncertainty maps, distinguishing clearly between regions of high and low confidence. These findings underline that, while standard linear models can excel in terms of accuracy on linearly separable datasets, Gaussian Processes remain valuable when the reliability of uncertainty estimates is critical for decision making.

The introduction of Simulation-Based Calibration as a diagnostic tool for Gaussian Processes inference further strengthened the methodological contribution of this work. By testing both regression and classification pipelines under synthetic scenarios, SBC was shown to provide a rigorous framework for verifying the correctness and calibration of Gaussian Processes inference.

Taken together, the results of this thesis suggest that Gaussian Processes occupy a distinctive position within the modern machine learning and statistics landscape. At the same time, their limitations should not be overlooked. Computational scalability, kernel selection, and the reliance on approximate inference for non-Gaussian likelihoods remain active challenges.

Future directions include the development of sparse and scalable Gaussian Processes methods to handle large datasets or the integration of Gaussian Processes into hierarchical or deep architectures. Furthermore, Gaussian Processes are expected to find increasing application across interdisciplinary domains such as medicine, engineering, and environmental sciences, underscoring their flexibility and adaptability to a wide range of contexts.

In conclusion, Gaussian Processes provide a unique balance of flexibility and principled uncertainty quantification, that makes them a powerful and enduring tool for advancing both the theory and practice of statistical learning.

7 Appendix

.1 Bayes' Theorem

Bayes' theorem provides a general rule for updating a prior distribution in light of observed data:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{p(\mathcal{D})},$$

where:

- $p(\theta)$ is the **prior**, encoding prior beliefs about the parameters θ;
- $p(\mathcal{D} \mid \theta)$ is the **likelihood**, the probability of the data given the parameters;
- $p(\mathcal{D})$ is the **evidence** or marginal likelihood, acting as a normalization constant;
- $p(\theta \mid \mathcal{D})$ is the **posterior**, which combines prior knowledge with observed evidence.

.2 Logistic Regression

Logistic regression is a widely used discriminative model for binary classification tasks, where the response variable $y \in \{0, 1\}$. Unlike generative approaches, logistic regression directly models the posterior probability $p(y \mid \mathbf{x})$ without assuming a particular form for the input distribution $p(\mathbf{x})$.

The model assumes that the log-odds of the probability of class $y = 1$ are a linear function of the input features:

$$\log\left(\frac{p(y=1 \mid \mathbf{x})}{p(y=0 \mid \mathbf{x})}\right) = \mathbf{w}^\top \mathbf{x} + b,$$

where $\mathbf{w} \in R^d$ is the weight vector, $b \in R$ is the bias term, and $\mathbf{x} \in R^d$ is the input feature vector.

Solving for $p(y = 1 \mid \mathbf{x})$, we obtain:

$$p(y=1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)}.$$

.3 Logistic Sigmoid and Probit Functions

.3.1 Logistic Sigmoid Function

The logistic sigmoid function is a smooth, monotonically increasing function that maps real numbers to the interval $[0, 1]$. It is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

This function is commonly used in binary classification models, such as logistic regression, to map the output of a linear function $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$ to a probability:

$$p(y=1 \mid \mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}).$$

Properties:

- $\sigma(z)$ is differentiable and strictly increasing.
- $\sigma(-z) = 1 - \sigma(z)$, which is useful for symmetric likelihoods.
- The derivative is given by:
$$\sigma'(z) = \sigma(z)\,(1 - \sigma(z))\,,$$
making it well-suited for gradient-based optimization.

.3.2 Probit Function

The probit function is defined as the cumulative distribution function (CDF) of the standard normal distribution:
$$\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(t \mid 0, 1)\, dt = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{t^2}{2}\right) dt.$$

In probit regression, the class probability is modeled as:
$$p(y = 1 \mid \mathbf{x}) = \Phi(\mathbf{x}^\top \mathbf{w}).$$

Properties:

- $\Phi(z)$ is smooth and monotonically increasing.
- $\Phi(-z) = 1 - \Phi(z)$, satisfying the symmetry condition.
- It arises naturally when assuming a latent variable model with Gaussian noise:
$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}) + \varepsilon > 0, \\ 0 & \text{otherwise,} \end{cases} \quad \text{with } \varepsilon \sim \mathcal{N}(0, 1).$$

.4 Laplace Approximation

The Laplace approximation is a technique for approximating a probability distribution with a Gaussian distribution centered at the mode. It is particularly useful in Bayesian inference when the posterior distribution does not admit a closed-form expression.

Let $p(\mathbf{w} \mid \mathcal{D})$ denote the posterior distribution over the parameters $\mathbf{w}$ given data $\mathcal{D}$. If the posterior is proportional to an unnormalized density of the form
$$p(\mathbf{w} \mid \mathcal{D}) \propto \exp(\ell(\mathbf{w})),$$
where $\ell(\mathbf{w}) = \log p(\mathcal{D} \mid \mathbf{w}) + \log p(\mathbf{w})$ is the log-posterior, the Laplace approximation proceeds by expanding $\ell(\mathbf{w})$ as a second-order Taylor series around its mode $\mathbf{w}_{\text{MAP}}$. This gives:
$$\ell(\mathbf{w}) \approx \ell(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}(\mathbf{w} - \mathbf{w}_{\text{MAP}})^\top A (\mathbf{w} - \mathbf{w}_{\text{MAP}}),$$
where the first term $\ell(\mathbf{w}_{\text{MAP}})$ is the log-posterior evaluated at the mode, and the second quadratic term $-\frac{1}{2}(\mathbf{w} - \mathbf{w}_{\text{MAP}})^\top A (\mathbf{w} - \mathbf{w}_{\text{MAP}})$ captures the local curvature of the log-posterior around $\mathbf{w}_{\text{MAP}}$, with $A = -\nabla^2 \ell(\mathbf{w})\big|_{\mathbf{w}=\mathbf{w}_{\text{MAP}}}$ the Hessian of the negative log-posterior.

Exponentiating both sides leads to a Gaussian approximation to the posterior:
$$p(\mathbf{w} \mid \mathcal{D}) \approx \mathcal{N}(\mathbf{w}_{\text{MAP}}, A^{-1}).$$

This approximation simplifies posterior inference and enables analytical integration in cases where exact inference is intractable. In the context of Bayesian classification models, such as logistic or probit regression, the Laplace approximation allows for efficient computation of approximate predictive distributions by integrating over the approximate posterior.

.5 Softmax function

Given a vector of class-specific latent values $\mathbf{f}_* = [f_*^1, \ldots, f_*^C]^\top \in R^C$, the softmax function maps these values to a valid probability distribution over the C classes:

$$\text{softmax}(\mathbf{f}_*)_c = \frac{\exp(f_*^c)}{\sum_{c'=1}^{C} \exp(f_*^{c'})}, \quad \text{for } c = 1, \ldots, C.$$

This function ensures that $\sum_{c=1}^{C} \text{softmax}(\mathbf{f}_*)_c = 1$ and each component lies in the interval $(0, 1)$.

.6 Generalized Fisher information matrix

The Fisher information matrix measures the curvature of the log-likelihood and quantifies how much information the data provide about the model parameters. In the multiclass setting with softmax likelihood, the matrix has the following form

$$W = \text{diag}(\boldsymbol{\pi}) - \pi\pi^\top \in R^{Cn \times Cn},$$

.7 Monte Carlo estimation of expectations

When computing expectations of nonlinear functions over Gaussian distributions, a closed-form expression is often unavailable. In such cases, one may resort to Monte Carlo (MC) estimation. Given a predictive distribution $\mathbf{f}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*)$, the expected class probability vector is

$$\bar{\boldsymbol{\pi}}_* = E[\text{softmax}(\mathbf{f}_*)] \approx \frac{1}{S} \sum_{s=1}^{S} \text{softmax}(\mathbf{f}_*^{(s)}),$$

where $\mathbf{f}_*^{(s)} \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*)$ are S independent samples. This estimate converges to the true expectation as $S \to \infty$.

.8 Root Mean Squared Error

For regression models, the Root Mean Squared Error is a widely used metric to quantify predictive accuracy. RMSE captures the average magnitude of the residuals between predicted values $\hat{y}_i$ and actual observations y_i. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}.$$

Here, n is the number of test points. In the context of this work, RMSE was used to compare the predictive performance of Bayesian Linear Regression and Gaussian Processes Regression on a real dataset.

.9 Area Under the ROC Curve (AUC)

In binary classification tasks, the Area Under the Receiver Operating Characteristic (ROC) Curve, serves as a robust performance metric. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) across various classification thresholds. The AUC measures the model's ability to distinguish between the two classes.

Formally, let $\pi_* \in [0, 1]$ denote the predicted probability of belonging to the positive class. For a classifier that outputs such probabilities, the ROC curve is generated by evaluating the binary decisions

over varying thresholds $t \in [0, 1]$, and the AUC is computed as the integral,

$$\text{AUC} = \int_0^1 \text{TPR}(t)\, d\text{FPR}(t),$$

where $\text{TPR}(t)$ and $\text{FPR}(t)$ denote the true and false positive rates at threshold t.

An AUC of 1 indicates perfect discrimination, whereas a value of 0.5 suggests no discriminative power (equivalent to random guessing).

.10 Expectation Propagation

Expectation Propagation is an approximate inference method used to estimate the posterior distribution in models such as Gaussian Processes Classification with non-Gaussian likelihoods. In such settings, the exact posterior

$$p(\mathbf{f} \mid \mathcal{D}) \propto \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K) \prod_{i=1}^{n} p(y_i \mid f_i)$$

is intractable due to the non-Gaussian form of the likelihoods $p(y_i \mid f_i)$. EP approximates each likelihood factor with a site function,

$$p(y_i \mid f_i) \approx \tilde{Z}_i\, \tilde{t}_i(f_i) = \tilde{Z}_i \exp\left(-\frac{1}{2}\tilde{\nu}_i f_i^2 + \tilde{\tau}_i f_i\right),$$

resulting in a Gaussian approximation to the full posterior,

$$q(\mathbf{f}) = \frac{1}{Z_{\text{EP}}} \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K) \prod_{i=1}^{n} \tilde{t}_i(f_i) = \mathcal{N}(\boldsymbol{\mu}, \Sigma).$$

EP operates iteratively, updating each site $\tilde{t}_i(f_i)$ using moment matching,

1. Compute the cavity distribution (removing site i):

$$q^{\setminus i}(f_i) \propto \frac{q(f_i)}{\tilde{t}_i(f_i)}.$$

2. Form the tilted distribution:

$$\hat{p}_i(f_i) \propto q^{\setminus i}(f_i) p(y_i \mid f_i).$$

3. Match the moments of $\hat{p}_i(f_i)$ with a Gaussian:

$$\hat{p}_i(f_i) \approx \mathcal{N}(f_i \mid m_i, v_i).$$

4. Update the site parameters $\tilde{\nu}_i, \tilde{\tau}_i$ so that when recombined, $q(f_i)$ matches m_i, v_i.

.11 Cholesky Factorization for Sampling from a Multivariate Gaussian distribution

Let $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ denote a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu} \in R^n$ and covariance matrix $\Sigma \in R^{n \times n}$, where Σ is symmetric and positive definite. The Cholesky factorization decomposes Σ as

$$\Sigma = LL^\top,$$

where L is a lower triangular matrix with positive diagonal entries.

Given this factorization, samples from $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ can be generated as follows:

1. Draw $\mathbf{z} \sim \mathcal{N}(0, I_n)$, a vector of n independent standard normal variables.
2. Transform $\mathbf{z}$ via
$$\mathbf{x} = \boldsymbol{\mu} + L\mathbf{z}.$$

The resulting vector $\mathbf{x}$ has the desired distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ because

$$E[\mathbf{x}] = \boldsymbol{\mu}, \qquad \mathrm{Cov}(\mathbf{x}) = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top] = L\, E[zz^\top]\, L^\top = LIL^\top = \Sigma.$$

In summary, the Cholesky factorization provides an efficient and numerically stable procedure for generating correlated samples from a multivariate Gaussian distribution. By transforming independent standard normal draws through the triangular factor L, the method ensures that the resulting samples exhibit the correct mean and covariance structure specified by $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

7 Code

7.1 Gaussian Mean & Variance

Listing 1: Effect of μ and σ^2 on the Gaussian Distribution

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(-6, 6, 500)

fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharey=True)

# === Plot 1: Varying sigma, fixed mu
mus_var = [0, 0, 0]
sigmas_var = [np.sqrt(0.5), 1.0, np.sqrt(2.0)]
labels_var = [r'$\sigma^2 = 0.5$', r'$\sigma^2 = 1$', r'$\sigma^2 = 2$']
colors_var = ['blue', 'red', 'green']

axes[0].set_title("Effect of Variance (Fixed $\mu = 0$)")
for sigma, label, color in zip(sigmas_var, labels_var, colors_var):
    y = norm.pdf(x, loc=0, scale=sigma)
    axes[0].plot(x, y, label=label, color=color, linewidth=2)

axes[0].set_xlabel(r"$z$")
axes[0].set_ylabel(r"$p(z)$")
axes[0].legend()
axes[0].grid(True)

# === Plot 2: Varying mu, fixed sigma
mus_mu = [-2, 0, 2]
sigmas_mu = [1.0, 1.0, 1.0]
labels_mu = [r'$\mu = -2$', r'$\mu = 0$', r'$\mu = 2$']
colors_mu = ['purple', 'black', 'orange']

axes[1].set_title("Effect of Mean (Fixed $\sigma^2 = 1$)")
for mu, label, color in zip(mus_mu, labels_mu, colors_mu):
    y = norm.pdf(x, loc=mu, scale=1.0)
    axes[1].plot(x, y, label=label, color=color, linestyle='--', linewidth=2)

axes[1].set_xlabel(r"$z$")
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.savefig("gaussian_mu_vs_sigma_subplots.png", dpi=300)
plt.show()
```

7.2 Bivariate Gaussian Contour Plots

Listing 2: Contour plots of bivariate Gaussian distributions with varying correlation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Grid for plotting
x, y = np.mgrid[-3:3:.01, -3:3:.01]
pos = np.dstack((x, y))

# Parameters: mean vector is always 0
mean = [0, 0]
correlations = [0.8, 0.0, -0.8]
titles = [r'$\rho = 0.8$', r'$\rho = 0$', r'$\rho = -0.8$']

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for ax, rho, title in zip(axes, correlations, titles):
    cov = [[1, rho], [rho, 1]]
    rv = multivariate_normal(mean, cov)
    ax.contour(x, y, rv.pdf(pos), levels=8, cmap='viridis')
    ax.set_title(title, fontsize=14)
    ax.set_xlabel(r"$y_1$")
    ax.set_ylabel(r"$y_2$")
    ax.set_aspect('equal')
    ax.grid(True)

fig.suptitle("Contour plots of bivariate Gaussian distributions with varying
    correlation", fontsize=16)
plt.tight_layout()
plt.subplots_adjust(top=0.85)
plt.savefig("bivariate_gaussian_contours.png", dpi=300)
plt.show()
```

7.3 Bayesian Linear Regression: Prior, Likelihood, Posterior, and Sampled Functions

Listing 3: The evolution of Bayesian Linear Regression from prior to posterior

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Seed for reproducibility
np.random.seed(42)

# True weights and noise variance
w_true = np.array([0.3, 0.5])
sigma2 = 0.1 ** 2

# Prior mean and covariance
prior_mean = np.array([0.0, 0.0])
prior_cov = np.eye(2)

# Generate synthetic data
X_seq = np.array([-0.8, -0.4, 0.0, 0.4, 0.8] + list(np.linspace(-1, 1, 20 - 5)
    ))
Y_seq = w_true[0] + w_true[1] * X_seq + np.random.normal(0, np.sqrt(sigma2),
    size=X_seq.shape)

# Create grid for contour plots
w0 = np.linspace(-1, 1, 200)
w1 = np.linspace(-1, 1, 200)
W0, W1 = np.meshgrid(w0, w1)
pos = np.dstack((W0, W1))

# Function to compute the posterior distribution
def compute_posterior(X, Y, m0, S0, sigma2):
    X_design = np.vstack([np.ones_like(X), X]).T
    S0_inv = np.linalg.inv(S0)
    SN_inv = S0_inv + (1 / sigma2) * X_design.T @ X_design
    SN = np.linalg.inv(SN_inv)
    mN = SN @ (S0_inv @ m0 + (1 / sigma2) * X_design.T @ Y)
    return mN, SN

# Function to compute the likelihood of a single point
def compute_likelihood_contour(x, y):
    likelihood = np.zeros(W0.shape)
    for i in range(W0.shape[0]):
        for j in range(W0.shape[1]):
            w = np.array([W0[i, j], W1[i, j]])
            y_pred = w[0] + w[1] * x
            likelihood[i, j] = np.exp(-0.5 * (y - y_pred)**2 / sigma2)
    return likelihood / np.max(likelihood)

# Final version: last row uses 20 data points, others use 0  3
fig, axes = plt.subplots(4, 3, figsize=(15, 14))
plt.subplots_adjust(hspace=0.5, wspace=0.3)

steps = [0, 1, 2, 19]  # Indices of points to show (last step uses 20th point)

```

```
for idx, step in enumerate(steps):
    # Data up to current step
    X_step = X_seq[:step + 1]
    Y_step = Y_seq[:step + 1]

    # Compute posterior
    mN, SN = compute_posterior(X_step, Y_step, prior_mean, prior_cov, sigma2)
    rv_post = multivariate_normal(mN, SN)
    Z_post = rv_post.pdf(pos)

    # === Left Column: Likelihood of last point ===
    ax_like = axes[idx, 0]
    if step > 0:
        likelihood = compute_likelihood_contour(X_step[-1], Y_step[-1])
        ax_like.contourf(W0, W1, likelihood, levels=50, cmap='Oranges')
        ax_like.set_title(f"Likelihood of point {step + 1}", fontsize=13)
    else:
        ax_like.axis('off')  # no likelihood for step 0
    ax_like.set_xlim([-1, 1])
    ax_like.set_ylim([-1, 1])
    ax_like.set_xlabel(r'$w_0$', fontsize=11)
    ax_like.set_ylabel(r'$w_1$', fontsize=11)
    ax_like.grid(True)

    # === Middle Column: Prior or Posterior ===
    ax_post = axes[idx, 1]
    ax_post.contourf(W0, W1, Z_post, levels=50, cmap='Blues')
    ax_post.set_xlim([-1, 1])
    ax_post.set_ylim([-1, 1])
    ax_post.set_xlabel(r'$w_0$ (intercept)', fontsize=11)
    ax_post.set_ylabel(r'$w_1$ (slope)', fontsize=11)
    ax_post.grid(True)
    if step == 0:
        ax_post.set_title("Prior distribution in $w$-space", fontsize=13)
    elif step == 19:
        ax_post.set_title("Posterior after 20 data points", fontsize=13)
    else:
        ax_post.set_title(f"Posterior after {step + 1} data points", fontsize
            =13)

    # === Right Column: Sampled functions ===
    ax_data = axes[idx, 2]
    x_vals = np.linspace(-1, 1, 100)
    for _ in range(6):
        w_sample = np.random.multivariate_normal(mN, SN)
        y_vals = w_sample[0] + w_sample[1] * x_vals
        ax_data.plot(x_vals, y_vals, 'r', linewidth=1, alpha=0.8)
    ax_data.scatter(X_step, Y_step, facecolors='none', edgecolors='b', s=60,
        label='Data')
    ax_data.set_xlim([-1, 1])
    ax_data.set_ylim([-1, 1])
    ax_data.set_xlabel('$x$', fontsize=11)
    ax_data.set_ylabel('$y$', fontsize=11)
    ax_data.grid(True)
    if step == 0:
```

```
        ax_data.set_title("Functions $y(x, \\mathbf{w})$ from prior", fontsize=13)
    else:
        ax_data.set_title("Functions $y(x, \\mathbf{w})$ from posterior", fontsize=13)

# Save figure
plt.tight_layout()
plt.savefig('bayesian_regression.png', dpi=300)
plt.show()
```

7.4 Gaussian Processes Prior and Posterior

Listing 4: Sampling from a Gaussian Processes prior and posterior with a squared exponential kernel

```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define input space
X = np.linspace(-5, 5, 100).reshape(-1, 1)

# Squared exponential kernel
def squared_exponential_kernel(x1, x2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(x1**2, 1).reshape(-1, 1) + np.sum(x2**2, 1) - 2 * np.dot(x1, x2.T)
    return sigma_f**2 * np.exp(-0.5 / l**2 * sqdist)

# Sample from the GP prior
K_prior = squared_exponential_kernel(X, X)
samples_prior = np.random.multivariate_normal(mean=np.zeros(len(X)), cov=K_prior, size=3)

# Plot GP prior
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
for i in range(3):
    plt.plot(X, samples_prior[i], lw=2)
plt.title("(a), prior")
plt.fill_between(X.flatten(), -2, 2, color="gray", alpha=0.2)
plt.xlabel("input, x")
plt.ylabel("output, f(x)")

# Generate some observed data
X_obs = np.array([[-4], [-1], [1], [3]])
y_obs = np.sin(X_obs).flatten() + 0.1 * np.random.randn(len(X_obs))

# Compute posterior
K = squared_exponential_kernel(X_obs, X_obs)
K_s = squared_exponential_kernel(X_obs, X)
K_ss = squared_exponential_kernel(X, X)
K_inv = np.linalg.inv(K + 1e-8 * np.eye(len(X_obs)))

# Predictive mean and covariance
```

```
mu_post = K_s.T.dot(K_inv).dot(y_obs)
cov_post = K_ss - K_s.T.dot(K_inv).dot(K_s)
samples_post = np.random.multivariate_normal(mean=mu_post, cov=cov_post, size=3)

# Plot GP posterior
plt.subplot(1, 2, 2)
for i in range(3):
    plt.plot(X, samples_post[i], lw=2)
plt.plot(X_obs, y_obs, 'k+', ms=10)
plt.fill_between(X.flatten(), mu_post - 2*np.sqrt(np.diag(cov_post)),
                 mu_post + 2*np.sqrt(np.diag(cov_post)), color="gray", alpha=0.3)
plt.title("(b), posterior")
plt.xlabel("input, x")
plt.ylabel("output, f(x)")
plt.tight_layout()
plt.savefig("gp_prior_posterior.png", dpi=300)
plt.show()
```

7.5 Comparison of Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression

7.5.1 Comparison of Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression on a synthetic linear dataset

Listing 5: Regression with frequentist, Bayesian, and Gaussian Processes approaches on the same synthetic linear dataset

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
from scipy.stats import multivariate_normal

# Simulated dataset
np.random.seed(42)
n = 30
X_sim = np.array([...])
y_sim = np.array([...])
X_design = np.vstack([np.ones(n), X_sim]).T  # Design matrix with intercept
y = y_sim
x_test = np.linspace(-6, 6, 100)
X_test_design = np.vstack([np.ones_like(x_test), x_test]).T

# Frequentist Linear Regression
w_frequentist = inv(X_design.T @ X_design) @ X_design.T @ y
y_pred_freq = X_test_design @ w_frequentist

# Bayesian Linear Regression
sigma_n = 0.5
Sigma_p = np.eye(2)
A = (1 / sigma_n**2) * X_design.T @ X_design + inv(Sigma_p)
Sigma_post = inv(A)
mu_post = Sigma_post @ ((1 / sigma_n**2) * X_design.T @ y)

# Predictive mean and variance for Bayesian Linear Regression
```

```
y_pred_bayes_mean = X_test_design @ mu_post
y_pred_bayes_var = np.sum(X_test_design @ Sigma_post * X_test_design, axis=1)

# Gaussian Process Regression with linear kernel
def linear_kernel(X1, X2):
    return X1 @ X2.T

X_gp = X_sim[:, None]
X_test_gp = x_test[:, None]

K = linear_kernel(X_gp, X_gp) + sigma_n**2 * np.eye(len(X_gp))
K_s = linear_kernel(X_gp, X_test_gp)
K_ss = linear_kernel(X_test_gp, X_test_gp)

K_inv = inv(K)
gp_mean = K_s.T @ K_inv @ y
gp_cov = K_ss - K_s.T @ K_inv @ K_s
gp_std = np.sqrt(np.diag(gp_cov))

# Plotting
plt.figure(figsize=(15, 4))

# Frequentist
plt.subplot(1, 3, 1)
plt.title("Frequentist Linear Regression")
plt.scatter(X_sim, y_sim, color='black', label='Training data')
plt.plot(x_test, y_pred_freq, label='Prediction', color='blue')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()

# Bayesian
plt.subplot(1, 3, 2)
plt.title("Bayesian Linear Regression")
plt.scatter(X_sim, y_sim, color='black', label='Training data')
plt.plot(x_test, y_pred_bayes_mean, label='Predictive mean', color='green')
plt.fill_between(x_test,
                 y_pred_bayes_mean - 2 * np.sqrt(y_pred_bayes_var),
                 y_pred_bayes_mean + 2 * np.sqrt(y_pred_bayes_var),
                 alpha=0.3, label=' 2  std')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()

# Gaussian Process
plt.subplot(1, 3, 3)
plt.title("Gaussian Process Regression")
plt.scatter(X_sim, y_sim, color='black', label='Training data')
plt.plot(x_test, gp_mean, label='Predictive mean', color='red')
plt.fill_between(x_test, gp_mean - 2 * gp_std, gp_mean + 2 * gp_std,
                 alpha=0.3, label=' 2  std')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
```

```
plt.legend()

plt.tight_layout()
plt.savefig("regression_comparison.png")
plt.show()
```

7.5.2 Comparison of Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression on a synthetic nonlinear dataset

Listing 6: Comparison between Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression on a synthetic nonlinear dataset. The data are generated from $y = \sin(2\pi x)$ with Gaussian noise.

```
# Simulated nonlinear dataset
np.random.seed(0)
n_points = 25
X_sim = np.linspace(0, 1, n_points)
y_sim = np.sin(2 * np.pi * X_sim) + np.random.normal(0, 0.1, n_points)

# Design matrix for linear models
X_design = np.vstack([np.ones(n_points), X_sim]).T
y = y_sim
x_test = np.linspace(-0.2, 1.2, 200)
X_test_design = np.vstack([np.ones_like(x_test), x_test]).T

# Frequentist Linear Regression
w_frequentist = inv(X_design.T @ X_design) @ X_design.T @ y
y_pred_freq = X_test_design @ w_frequentist

# Bayesian Linear Regression
sigma_n = 0.1
Sigma_p = np.eye(2)
A = (1 / sigma_n**2) * X_design.T @ X_design + inv(Sigma_p)
Sigma_post = inv(A)
mu_post = Sigma_post @ ((1 / sigma_n**2) * X_design.T @ y)
y_pred_bayes_mean = X_test_design @ mu_post
y_pred_bayes_var = np.sum(X_test_design @ Sigma_post * X_test_design, axis=1)

# Gaussian Process Regression with RBF kernel
def rbf_kernel(X1, X2, lengthscale=0.2, variance=1.0):
    sqdist = np.subtract.outer(X1, X2)**2
    return variance * np.exp(-0.5 * sqdist / lengthscale**2)

X_gp = X_sim[:, None]
X_test_gp = x_test[:, None]
K = rbf_kernel(X_gp.ravel(), X_gp.ravel()) + sigma_n**2 * np.eye(n_points)
K_s = rbf_kernel(X_gp.ravel(), x_test)
K_ss = rbf_kernel(x_test, x_test)

K_inv = inv(K)
gp_mean = K_s.T @ K_inv @ y
gp_cov = K_ss - K_s.T @ K_inv @ K_s
gp_std = np.sqrt(np.maximum(np.diag(gp_cov), 0))  # numerical safety

# Plotting
plt.figure(figsize=(15, 4))
```

```
44
45 # Frequentist
46 plt.subplot(1, 3, 1)
47 plt.title("Frequentist Linear Regression")
48 plt.scatter(X_sim, y_sim, color='black', label='Training data')
49 plt.plot(x_test, y_pred_freq, label='Prediction', color='blue')
50 plt.xlabel("x")
51 plt.ylabel("y")
52 plt.grid(True)
53 plt.legend()
54
55 # Bayesian
56 plt.subplot(1, 3, 2)
57 plt.title("Bayesian Linear Regression")
58 plt.scatter(X_sim, y_sim, color='black', label='Training data')
59 plt.plot(x_test, y_pred_bayes_mean, label='Predictive mean', color='green')
60 plt.fill_between(x_test,
61                  y_pred_bayes_mean - 2 * np.sqrt(y_pred_bayes_var),
62                  y_pred_bayes_mean + 2 * np.sqrt(y_pred_bayes_var),
63                  alpha=0.3, label=' 2  std')
64 plt.xlabel("x")
65 plt.ylabel("y")
66 plt.grid(True)
67 plt.legend()
68
69 # Gaussian Process
70 plt.subplot(1, 3, 3)
71 plt.title("Gaussian Process Regression")
72 plt.scatter(X_sim, y_sim, color='black', label='Training data')
73 plt.plot(x_test, gp_mean, label='Predictive mean', color='red')
74 plt.fill_between(x_test, gp_mean - 2 * gp_std, gp_mean + 2 * gp_std,
75                  alpha=0.3, label=' 2  std')
76 plt.xlabel("x")
77 plt.ylabel("y")
78 plt.grid(True)
79 plt.legend()
80
81 plt.tight_layout()
82 plt.savefig("nonlinear_regression_comparison.png")
83 plt.show()
```

7.5.3 Comparison of Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression on a real dataset

Listing 7: Comparison between Frequentist Linear Regression, Bayesian Linear Regression, and Gaussian Processes Regression on the Advertising dataset using TV as predictor.

```
1 # Import necessary modules
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from numpy.linalg import inv
6
7 # Load the Advertising dataset
8 url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.
    csv"
```

```
df = pd.read_csv(url)

# Use only the 'TV' feature and 'Sales' target
X_real = df["TV"].values
y_real = df["sales"].values
n_points = len(X_real)

# Sort for plotting consistency
sorted_idx = np.argsort(X_real)
X_real = X_real[sorted_idx]
y_real = y_real[sorted_idx]

# Design matrix for linear models
X_design_real = np.vstack([np.ones(n_points), X_real]).T

# Test points for prediction
x_test = np.linspace(X_real.min() - 10, X_real.max() + 10, 200)
X_test_design = np.vstack([np.ones_like(x_test), x_test]).T

# Frequentist Linear Regression
w_frequentist = inv(X_design_real.T @ X_design_real) @ X_design_real.T @
    y_real
y_pred_freq = X_test_design @ w_frequentist

# Bayesian Linear Regression
sigma_n = 1.0
Sigma_p = np.eye(2)
A = (1 / sigma_n**2) * X_design_real.T @ X_design_real + inv(Sigma_p)
Sigma_post = inv(A)
mu_post = Sigma_post @ ((1 / sigma_n**2) * X_design_real.T @ y_real)
y_pred_bayes_mean = X_test_design @ mu_post
y_pred_bayes_var = np.sum(X_test_design @ Sigma_post * X_test_design, axis=1)

# Gaussian Process Regression with RBF kernel
def rbf_kernel(X1, X2, lengthscale=30.0, variance=10.0):
    sqdist = np.subtract.outer(X1, X2)**2
    return variance * np.exp(-0.5 * sqdist / lengthscale**2)

X_gp = X_real[:, None]
X_test_gp = x_test[:, None]
K = rbf_kernel(X_gp.ravel(), X_gp.ravel()) + sigma_n**2 * np.eye(n_points)
K_s = rbf_kernel(X_gp.ravel(), x_test)
K_ss = rbf_kernel(x_test, x_test)

K_inv = inv(K)
gp_mean = K_s.T @ K_inv @ y_real
gp_cov = K_ss - K_s.T @ K_inv @ K_s
gp_std = np.sqrt(np.maximum(np.diag(gp_cov), 0))

# Plotting
plt.figure(figsize=(15, 4))

# Frequentist
plt.subplot(1, 3, 1)
plt.title("Frequentist Linear Regression")
plt.scatter(X_real, y_real, color='black', label='Training data')
```

```
plt.plot(x_test, y_pred_freq, label='Prediction', color='blue')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()

# Bayesian
plt.subplot(1, 3, 2)
plt.title("Bayesian Linear Regression")
plt.scatter(X_real, y_real, color='black', label='Training data')
plt.plot(x_test, y_pred_bayes_mean, label='Predictive mean', color='green')
plt.fill_between(x_test,
                 y_pred_bayes_mean - 2 * np.sqrt(y_pred_bayes_var),
                 y_pred_bayes_mean + 2 * np.sqrt(y_pred_bayes_var),
                 alpha=0.3, label=' 2  std')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()

# Gaussian Process
plt.subplot(1, 3, 3)
plt.title("Gaussian Process Regression")
plt.scatter(X_real, y_real, color='black', label='Training data')
plt.plot(x_test, gp_mean, label='Predictive mean', color='red')
plt.fill_between(x_test, gp_mean - 2 * gp_std, gp_mean + 2 * gp_std,
                 alpha=0.3, label=' 2  std')
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.savefig("advertising_regression_comparison.png")
plt.show()
```

7.6 Hyperparameters in Gaussian Processes Regression

Listing 8: Effect of Gaussian Processes Regression kernel hyperparameters (length-scale, signal variance, noise variance) on predictive mean and uncertainty.

```
import numpy as np
import matplotlib.pyplot as plt

def rbf_kernel(X1, X2, length_scale, sigma_f):
    sqdist = np.subtract.outer(X1, X2) ** 2
    return sigma_f ** 2 * np.exp(-0.5 / length_scale**2 * sqdist)

def gp_posterior(X_train, y_train, X_test, length_scale, sigma_f, sigma_n):
    K = rbf_kernel(X_train, X_train, length_scale, sigma_f) + sigma_n**2 * np.
        eye(len(X_train))
    K_s = rbf_kernel(X_train, X_test, length_scale, sigma_f)
    K_ss = rbf_kernel(X_test, X_test, length_scale, sigma_f) + 1e-8 * np.eye(
        len(X_test))
    K_inv = np.linalg.inv(K)

```

```
    mu_s = K_s.T @ K_inv @ y_train
    cov_s = K_ss - K_s.T @ K_inv @ K_s
    return mu_s, np.sqrt(np.diag(cov_s))

# Base data
X_test = np.linspace(-5, 5, 200)
X_train = np.linspace(-4.5, 4.5, 10)
true_function = lambda x: np.sin(x)
rng = np.random.default_rng(42)

# Base parameters
base_l = 1.0
base_sf = 1.0
base_sn = 0.1

def plot_param_comparison(title, param_pair, filename):
    fig, axs = plt.subplots(1, 2, figsize=(13.5, 4))
    for col_idx, params in enumerate(param_pair):
        sigma_n = params["sigma_n"]
        length_scale = params["length_scale"]
        sigma_f = params["sigma_f"]

        y_train = true_function(X_train) + rng.normal(0, sigma_n, size=X_train.shape)
        mu_s, std_s = gp_posterior(X_train, y_train, X_test, length_scale, sigma_f, sigma_n)

        ax = axs[col_idx]
        ax.plot(X_test, mu_s, 'b', label="Mean prediction")
        ax.fill_between(X_test, mu_s - 2 * std_s, mu_s + 2 * std_s,
                        color='blue', alpha=0.3, label="Confidence interval")
        ax.scatter(X_train, y_train, c='k', marker='x', label="Training data")
        ax.set_ylim(-3, 3)
        ax.set_xlabel("Input x")
        ax.set_ylabel("Output f(x)")
        ax.grid(True)

        # Titles
        if title == "Noise Variance        ":
            desc = f"$\\sigma_n^2$ = {sigma_n:.2f}"
        elif title == "Length Scale    ":
            desc = f"$\\ell$ = {length_scale:.2f}"
        else:
            desc = f"$\\sigma_f^2$ = {sigma_f:.2f}"
        ax.set_title(desc)

    handles, labels = axs[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc="center left", bbox_to_anchor=(1.01, 0.5), frameon=False)
    fig.suptitle(title, fontsize=14)
    fig.tight_layout(rect=[0, 0, 0.95, 0.93])
    fig.savefig(filename, bbox_inches="tight")
    return fig

# Configurations
configs = {
```

```
    "Noise Variance        ": (
        [
            {"sigma_n": 0, "length_scale": base_l, "sigma_f": base_sf},
            {"sigma_n": 1.0, "length_scale": base_l, "sigma_f": base_sf}
        ],
        "noisevariance_effect.png"
    ),
    "Length Scale    ": (
        [
            {"sigma_n": base_sn, "length_scale": 0.5, "sigma_f": base_sf},
            {"sigma_n": base_sn, "length_scale": 4.0, "sigma_f": base_sf}
        ],
        "lengthscale_effect.png"
    ),
    r"Signal Variance $\sigma_{\mathrm{f}}^2$": (
        [
            {"sigma_n": base_sn, "length_scale": base_l, "sigma_f": 0.5},
            {"sigma_n": base_sn, "length_scale": base_l, "sigma_f": 4.0}
        ],
        "signalvariance_effect.png"
    )
}

# Generate figures
fig1 = plot_param_comparison("Length Scale", *configs["Length Scale"])
fig2 = plot_param_comparison("Signal Variance", *configs["Signal Variance"])
fig3 = plot_param_comparison("Noise Variance", *configs["Noise Variance"])
```

7.7 Gaussian Processes Regression on the Concrete Strength dataset

Listing 9: Gaussian Processes Regression on the Concrete Strength dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from scipy.optimize import minimize
from ucimlrepo import fetch_ucirepo

# Load dataset via ucimlrepo
concrete = fetch_ucirepo(id=165)  # Concrete Compressive Strength
X_df = concrete.data.features
y_df = concrete.data.targets.iloc[:, 0]  # MPa values

# Convert to numpy
X = X_df.values
y = y_df.values

# Standardize inputs and outputs
scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.reshape(-1,1)).flatten()

```

```
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_scaled, train_size=800, random_state=42
)

# RBF kernel
def rbf_kernel(A, B, l, sigma_f):
    sqdist = np.sum(A**2,1).reshape(-1,1) + np.sum(B**2,1) - 2*A.dot(B.T)
    return sigma_f**2 * np.exp(-0.5/sigma_f**2 * sqdist)

# Negative log-marginal likelihood function
def nll_fn(params):
    l, sigma_f, sigma_n = np.exp(params)
    K = rbf_kernel(X_train, X_train, l, sigma_f) + sigma_n**2 * np.eye(len(
        X_train))
    try:
        L = np.linalg.cholesky(K)
    except np.linalg.LinAlgError:
        return np.inf
    alpha = np.linalg.solve(L.T, np.linalg.solve(L, y_train))
    nll = 0.5 * y_train.T.dot(alpha) + np.sum(np.log(np.diag(L))) \
          + 0.5 * len(y_train) * np.log(2*np.pi)
    return nll

# Optimize hyperparameters
initial = np.log([1.0, 1.0, 0.1])
res = minimize(nll_fn, initial, method='L-BFGS-B')
l_opt, sigma_f_opt, sigma_n_opt = np.exp(res.x)
print(f"Optimized l={l_opt:.3f},  _f ={sigma_f_opt:.3f},  _n ={sigma_n_opt:.3f
    }")

# Build predictive posterior
K = rbf_kernel(X_train, X_train, l_opt, sigma_f_opt) + sigma_n_opt**2 * np.eye
    (len(X_train))
K_s = rbf_kernel(X_train, X_test, l_opt, sigma_f_opt)
K_ss = np.diag(rbf_kernel(X_test, X_test, l_opt, sigma_f_opt))

L = np.linalg.cholesky(K)
alpha = np.linalg.solve(L.T, np.linalg.solve(L, y_train))
mu_s = K_s.T.dot(alpha)
v = np.linalg.solve(L, K_s)
var_s = K_ss - np.sum(v**2, axis=0)
std_s = np.sqrt(np.maximum(var_s, 0.0))

# Transform predictions back to original scale
y_pred = scaler_y.inverse_transform(mu_s.reshape(-1,1)).flatten()
y_test_true = scaler_y.inverse_transform(y_test.reshape(-1,1)).flatten()
std_true = std_s * scaler_y.scale_[0]

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test_true, y_pred))
print(f"Final RMSE: {rmse:.2f} MPa")

# Plot results
plt.figure(figsize=(10,5))
plt.plot(y_pred, label='Predicted mean', color='crimson')
```

```
plt.fill_between(np.arange(len(y_pred)),
                 y_pred - 2*std_true,
                 y_pred + 2*std_true,
                 color='lightcoral', alpha=0.5,
                 label='95% confidence')
plt.plot(y_test_true, '--k', label='True values', color='navy')
plt.xlabel('Test sample index')
plt.ylabel('Compressive Strength (MPa)')
plt.title(f'GPR on Concrete Strength')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('gpr_concrete_strength_optimized.png', dpi=300)
plt.show()
```

7.8 Bayesian Linear Regression on the Concrete Strength dataset

Listing 10: Bayesian Linear Regression on the Concrete Strength dataset

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from scipy.stats import norm
from ucimlrepo import fetch_ucirepo

# Load dataset (Concrete Compressive Strength)
concrete = fetch_ucirepo(id=165)
X_df = concrete.data.features
y_df = concrete.data.targets.iloc[:, 0]  # MPa

# Convert to numpy arrays
X = X_df.values
y = y_df.values

# Standardize inputs and outputs
scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

# Train/test split (fixed seed and size)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_scaled, train_size=800, random_state=42
)

# Add intercept to design matrices
def add_intercept(A):
    return np.hstack([A, np.ones((A.shape[0], 1))])

X_train_aug = add_intercept(X_train)
X_test_aug  = add_intercept(X_test)

# Prior covariance and noise variance
sigma2_n = 0.1
Sigma_p = np.eye(X_train_aug.shape[1])

# Posterior over weights
Sigma_w = np.linalg.inv(
    (1/sigma2_n) * X_train_aug.T @ X_train_aug + np.linalg.inv(Sigma_p)
)
mu_w = (1/sigma2_n) * Sigma_w @ X_train_aug.T @ y_train

# Posterior predictive mean and variance
y_pred_test_scaled = X_test_aug @ mu_w
var_pred_scaled = np.sum(X_test_aug @ Sigma_w * X_test_aug, axis=1) + sigma2_n
std_pred_scaled = np.sqrt(var_pred_scaled)

# Transform back to original scale (MPa)
scale_y = scaler_y.scale_[0]
y_pred = scaler_y.inverse_transform(y_pred_test_scaled.reshape(-1, 1)).flatten
    ()
```

```
y_test_true = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
std_pred = std_pred_scaled * scale_y

# 95% prediction intervals
alpha = 0.05
zcrit = norm.ppf(1 - alpha/2)
pi_lower = y_pred - zcrit * std_pred
pi_upper = y_pred + zcrit * std_pred

# RMSE on test set
rmse = np.sqrt(mean_squared_error(y_test_true, y_pred))
print(f"Bayesian Linear Regression     RMSE: {rmse:.2f} MPa")

# Plot: predicted mean + 95% intervals and true values
plt.figure(figsize=(12, 5))
plt.fill_between(
    np.arange(len(y_pred)), pi_lower, pi_upper,
    color='lightcoral', alpha=0.35, label='95% prediction interval'
)
plt.plot(y_pred, color='crimson', lw=1.5, label='Predicted mean')
plt.plot(y_test_true, '--', color='navy', lw=1.2, label='True values')

plt.xlabel('Test sample index')
plt.ylabel('Compressive Strength (MPa)')
plt.title('Bayesian Linear Regression on Concrete Strength')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('bayesian_linear_regression_concrete_strength.png', dpi=300)
plt.show()
```

7.9 Gaussian Processes Classification on the Breast Cancer dataset

7.9.1 Gaussian Processes Classification with Laplace Approximation on the Breast Cancer dataset

Listing 11: Gaussian Processes Classification with Laplace Approximation on the Breast Cancer dataset

```
import numpy as np
import pandas as pd
from scipy.stats import norm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, roc_auc_score, RocCurveDisplay
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo

# 1. Load Breast Cancer Dataset (UCI)
bc = fetch_ucirepo(id=17)
X = bc.data.features
y = (bc.data.targets.values[:, 0] == 'M').astype(int)  # 1 = malignant, 0 =
    benign

# 2. Drop any rows with NaNs
df = X.copy()
df['target'] = y
```

```
df = df.dropna()
X = df.drop('target', axis=1).values
y = df['target'].values

# 3. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, train_size=400, stratify=y, random_state=42
)

# 5. RBF kernel function
def rbf_kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1**2, axis=1).reshape(-1,1) \
           + np.sum(X2**2, axis=1) - 2 * X1 @ X2.T
    return sigma_f**2 * np.exp(-0.5 * sqdist / l**2)

# 6. Laplace Approximation function
def laplace_gpc(K, y, max_iter=10):
    f = np.zeros(len(y))  # initialize latent function
    for _ in range(max_iter):
        pi = norm.cdf(f)                        # probit likelihood
        W = pi * (1 - pi) + 1e-6                # Hessian diagonal
        z = f + (y - pi) / W                    # Newton step
        W_sqrt = np.sqrt(W)
        B = W_sqrt[:, None] * K * W_sqrt[None, :]
        L = np.linalg.cholesky(np.eye(len(K)) + B)
        a = W_sqrt * np.linalg.solve(
            L.T, np.linalg.solve(L, W_sqrt * K.dot(W * z))
        )
        f = K.dot(W * z) - a
    return f, W, L

# 7. Fit the model using Laplace approximation
K = rbf_kernel(X_train, X_train, l=1.0, sigma_f=1.0)
f_hat, W, L = laplace_gpc(K, y_train)

# 8. Predictive distribution
K_s = rbf_kernel(X_train, X_test, l=1.0, sigma_f=1.0)
k_star_star = np.diag(rbf_kernel(X_test, X_test, l=1.0, sigma_f=1.0))
W_sqrt = np.sqrt(W)
K_tilde = W_sqrt[:, None] * K_s
v = np.linalg.solve(L, K_tilde)

# Predictive mean and variance
f_star_mean = K_s.T.dot(y_train - norm.cdf(f_hat))
f_star_var = k_star_star - np.sum(v**2, axis=0)
f_star_var = np.clip(f_star_var, 1e-6, None)

# Predictive class probabilities using probit link
probs = norm.cdf(f_star_mean / np.sqrt(1 + f_star_var))

# 9. Evaluation
acc = accuracy_score(y_test, probs >= 0.5)
```

```
auc = roc_auc_score(y_test, probs)
print(f"Laplace GPC      Accuracy: {acc:.3f}, ROC AUC: {auc:.3f}")

# 10. ROC Curve
RocCurveDisplay.from_predictions(y_test, probs)
plt.title("GPC ROC Curve (Laplace Approx.)      Breast Cancer Dataset")
plt.savefig("gpc_laplace_roc.png", dpi=300)
plt.show()
```

7.9.2 Gaussian Processes Classification with Expectation Propagation on the Breast Cancer dataset

Listing 12: Expectation Propagation for Gaussian Process Classification on the Breast Cancer dataset

```
import numpy as np
from scipy.stats import norm
from sklearn.metrics import accuracy_score, roc_auc_score, RocCurveDisplay
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo

# 1. Load and clean Breast Cancer dataset
bc = fetch_ucirepo(id=17)
X = bc.data.features
y = (bc.data.targets.values[:, 0] == 'M').astype(int)

df = X.copy()
df['target'] = y
df = df.dropna()
X = df.drop('target', axis=1).values
y = df['target'].values

# 2. Preprocessing: standardize and split
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, train_size=400, stratify=y, random_state=42
)

# 3. RBF kernel function
def rbf_kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1**2, axis=1).reshape(-1,1) + \
             np.sum(X2**2, axis=1) - 2 * X1 @ X2.T
    return sigma_f**2 * np.exp(-0.5 * sqdist / l**2)

# 4. Expectation Propagation (EP) algorithm for GPC
def ep_gpc(K, y, max_iter=25, tol=1e-4):
    n = len(y)
    y_bin = 2*y - 1  # convert labels to {-1,+1}
    tau_tilde = np.zeros(n)
    nu_tilde = np.zeros(n)
    Sigma = K.copy()
    mu = np.zeros(n)

    for _ in range(max_iter):
```

```
        Sigma_inv = np.linalg.inv(K) + np.diag(tau_tilde)
        Sigma = np.linalg.inv(Sigma_inv)
        mu = Sigma @ nu_tilde

        tau_old = tau_tilde.copy()
        for i in range(n):
            # cavity distribution
            sigma_cav = 1 / (1 / Sigma[i,i] - tau_tilde[i] + 1e-6)
            mu_cav = sigma_cav * (mu[i] / Sigma[i,i] - nu_tilde[i])

            # tilted distribution
            z = y_bin[i] * mu_cav / np.sqrt(1 + sigma_cav)
            Z = norm.cdf(z)
            if Z < 1e-10:
                Z = 1e-10

            ratio = norm.pdf(z) / Z
            mu_hat = mu_cav + y_bin[i] * sigma_cav * ratio / np.sqrt(1 +
                sigma_cav)
            sigma_hat = sigma_cav - (sigma_cav**2 * ratio * (z + ratio)) / (1
                + sigma_cav)

            # site update
            delta_tau = 1 / sigma_hat - 1 / sigma_cav
            delta_nu = mu_hat / sigma_hat - mu_cav / sigma_cav

            tau_tilde[i] += delta_tau
            nu_tilde[i] += delta_nu

        # check convergence
        if np.max(np.abs(tau_tilde - tau_old)) < tol:
            break

    return mu, Sigma, tau_tilde, nu_tilde

# 5. Train EP approximation
K = rbf_kernel(X_train, X_train)
mu_ep, Sigma_ep, _, _ = ep_gpc(K, y_train)

# 6. Predictive distribution
K_s = rbf_kernel(X_train, X_test)
k_star_star = np.diag(rbf_kernel(X_test, X_test))
v = np.linalg.solve(np.linalg.cholesky(K + np.linalg.inv(Sigma_ep)), K_s)

f_star_mean = K_s.T @ np.linalg.solve(K + np.linalg.inv(Sigma_ep), mu_ep)
f_star_var = k_star_star - np.sum(v**2, axis=0)
f_star_var = np.clip(f_star_var, 1e-6, None)

# 7. Predictive probabilities (probit link)
probs = norm.cdf(f_star_mean / np.sqrt(1 + f_star_var))

# 8. Evaluation
acc = accuracy_score(y_test, probs >= 0.5)
auc = roc_auc_score(y_test, probs)
print(f"EP GPC     Accuracy: {acc:.3f}, ROC AUC: {auc:.3f}")

```

```
# 9. ROC curve
RocCurveDisplay.from_predictions(y_test, probs)
plt.title("GPC ROC Curve (EP Approx.)     Breast Cancer Dataset")
plt.savefig("gpc_ep_roc.png", dpi=300)
plt.show()
```

7.9.3 Bayesian Logistic Regression on the Breast Cancer Dataset

Listing 13: Bayesian Logistic Regression on Breast Cancer Dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score, RocCurveDisplay
from ucimlrepo import fetch_ucirepo
import pandas as pd

# 1. Load Breast Cancer dataset (UCI)
bc = fetch_ucirepo(id=17)
X = bc.data.features
y = (bc.data.targets.values[:, 0] == 'M').astype(int)

# 2. Clean and prepare
df = X.copy()
df["target"] = y
df = df.dropna()
X = df.drop("target", axis=1).values
y = df["target"].values

# 3. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, train_size=400, stratify=y, random_state=42
)

# 5. Logistic Regression model
clf = LogisticRegression(
    penalty="l2",        # L2 regularization
    solver="lbfgs",      # optimization algorithm
    max_iter=1000,
    random_state=42
)
clf.fit(X_train, y_train)

# 6. Predictions (probabilities for class 1)
probs = clf.predict_proba(X_test)[:, 1]

# 7. Evaluation
acc = accuracy_score(y_test, probs >= 0.5)
auc = roc_auc_score(y_test, probs)
print(f"Logistic Regression     Accuracy: {acc:.3f}, ROC AUC: {auc:.3f}")
```

```
47
48 # 8. ROC curve
49 RocCurveDisplay.from_predictions(y_test, probs)
50 plt.title("Logistic Regression ROC Curve     Breast Cancer Dataset")
51 plt.savefig("logreg_roc.png", dpi=300)
52 plt.show()
```

7.10 Comparison of predictive variance between EP and Laplace approximations for Gaussian Processes Classification (Breast Cancer dataset, PCA projection)

Listing 14: Comparison of predictive variance between EP and Laplace approximations for Gaussian Processes Classification (Breast Cancer dataset, PCA projection).

```
1  import numpy as np
2  import pandas as pd
3  from scipy.stats import norm
4  from sklearn.model_selection import train_test_split
5  from sklearn.preprocessing import StandardScaler
6  from sklearn.decomposition import PCA
7  import matplotlib.pyplot as plt
8  from ucimlrepo import fetch_ucirepo
9  import seaborn as sns
10
11 # Set plotting style
12 plt.style.use('seaborn-v0_8-whitegrid')
13 sns.set_palette("husl")
14
15 # 1. Load and preprocess dataset
16 bc = fetch_ucirepo(id=17)
17 X = bc.data.features
18 y = (bc.data.targets.values[:, 0] == 'M').astype(int)
19 df = X.copy()
20 df['target'] = y
21 df = df.dropna()
22 X = df.drop('target', axis=1).values
23 y = df['target'].values
24
25 scaler = StandardScaler()
26 X_scaled = scaler.fit_transform(X)
27 X_train, X_test, y_train, y_test = train_test_split(
28     X_scaled, y, train_size=400, stratify=y, random_state=42
29 )
30
31 # 2. PCA for 2D visualization
32 pca = PCA(n_components=2)
33 X_train_pca = pca.fit_transform(X_train)
34 X_test_pca = pca.transform(X_test)
35
36 # 3. RBF kernel
37 def rbf_kernel(X1, X2, l=1.0, sigma_f=1.0):
38     sqdist = np.sum(X1**2, axis=1).reshape(-1,1) + np.sum(X2**2, axis=1) - 2 *
          X1 @ X2.T
39     return sigma_f**2 * np.exp(-0.5 * sqdist / l**2)
40
```

```python
# 4. Expectation Propagation (EP) for GPC
def ep_gpc(K, y, max_iter=25, tol=1e-4):
    n = len(y)
    y_bin = 2*y - 1
    K_stable = K + 1e-6 * np.eye(n)
    tau_tilde = np.zeros(n)
    nu_tilde = np.zeros(n)

    for iteration in range(max_iter):
        Sigma_inv = np.linalg.inv(K_stable) + np.diag(np.clip(tau_tilde, -1e6, 1e6))
        Sigma = np.linalg.inv(Sigma_inv)
        mu = Sigma @ nu_tilde
        tau_old = tau_tilde.copy()

        for i in range(n):
            sigma_ii = np.clip(Sigma[i,i], 1e-10, 1e6)
            tau_i = np.clip(tau_tilde[i], -1e6, 1e6)
            sigma_cav = 1 / (1 / sigma_ii - tau_i + 1e-8)
            mu_cav = sigma_cav * (mu[i] / sigma_ii - nu_tilde[i])
            z = y_bin[i] * mu_cav / np.sqrt(1 + sigma_cav)
            Z = np.clip(norm.cdf(z), 1e-10, 1-1e-10)
            ratio = norm.pdf(z) / Z
            mu_hat = mu_cav + y_bin[i] * sigma_cav * ratio / np.sqrt(1 + sigma_cav)
            sigma_hat = sigma_cav - (sigma_cav**2 * ratio * (z + ratio)) / (1 + sigma_cav)
            delta_tau = 1 / sigma_hat - 1 / sigma_cav
            delta_nu = mu_hat / sigma_hat - mu_cav / sigma_cav
            tau_tilde[i] += np.clip(delta_tau, -100, 100)
            nu_tilde[i] += np.clip(delta_nu, -100, 100)
        if np.max(np.abs(tau_tilde - tau_old)) < tol:
            print(f"EP converged after {iteration+1} iterations")
            break
    return mu, Sigma, tau_tilde, nu_tilde

# 5. Laplace Approximation for GPC
def laplace_gpc(K, y, max_iter=10):
    n = len(y)
    K_stable = K + 1e-6 * np.eye(n)
    f = np.zeros(n)
    for iteration in range(max_iter):
        pi = np.clip(norm.cdf(f), 1e-10, 1-1e-10)
        W = np.clip(pi * (1 - pi) + 1e-8, 1e-8, None)
        z = f + (y - pi) / W
        W_sqrt = np.sqrt(W)
        B = W_sqrt[:, None] * K_stable * W_sqrt[None, :]
        L = np.linalg.cholesky(np.eye(n) + B)
        a = W_sqrt * np.linalg.solve(L.T, np.linalg.solve(L, W_sqrt * K_stable.dot(W * z)))
        f_new = K_stable.dot(W * z) - a
        if np.max(np.abs(f_new - f)) < 1e-6:
            print(f"Laplace converged after {iteration+1} iterations")
            break
        f = f_new
    return f, W, L
```

```python

# 6. Train EP and Laplace models
K_train = rbf_kernel(X_train_pca, X_train_pca, l=1.5, sigma_f=1.0)
print("Fitting EP approximation...")
mu_ep, Sigma_ep, tau_ep, nu_ep = ep_gpc(K_train, y_train)
print("Fitting Laplace approximation...")
f_laplace, W_laplace, L_laplace = laplace_gpc(K_train, y_train)

# 7. Predictive variances on a grid
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50),
                     np.linspace(y_min, y_max, 50))
grid_points = np.c_[xx.ravel(), yy.ravel()]

K_grid_train = rbf_kernel(X_train_pca, grid_points, l=1.5, sigma_f=1.0)
k_grid_diag = np.diag(rbf_kernel(grid_points, grid_points, l=1.5, sigma_f=1.0)
    )

# EP predictive variance
Sigma_ep_inv = np.linalg.inv(Sigma_ep + 1e-8 * np.eye(len(Sigma_ep)))
K_ep_total = K_train + Sigma_ep_inv + 1e-6 * np.eye(len(K_train))
L_ep = np.linalg.cholesky(K_ep_total)
v_ep = np.linalg.solve(L_ep, K_grid_train)
f_grid_var_ep = np.clip(k_grid_diag - np.sum(v_ep**2, axis=0), 1e-8, 10)

# Laplace predictive variance
W_sqrt = np.sqrt(np.clip(W_laplace, 1e-8, None))
K_tilde = W_sqrt[:, None] * K_grid_train
v_laplace = np.linalg.solve(L_laplace, K_tilde)
f_grid_var_laplace = np.clip(k_grid_diag - np.sum(v_laplace**2, axis=0), 1e-8,
     10)

# 8. Visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
im1 = axes[0].contourf(xx, yy, f_grid_var_ep.reshape(xx.shape), levels=20,
    alpha=0.8, cmap='PuBu')
axes[0].scatter(X_train_pca[y_train==0,0], X_train_pca[y_train==0,1], c='white
    ', marker='o', edgecolors='black', s=30, label='Benign')
axes[0].scatter(X_train_pca[y_train==1,0], X_train_pca[y_train==1,1], c='white
    ', marker='^', edgecolors='black', s=30, label='Malignant')
axes[0].set_title('EP: Predictive Variance')
cbar1 = plt.colorbar(im1, ax=axes[0]); cbar1.set_label('Variance')

im2 = axes[1].contourf(xx, yy, f_grid_var_laplace.reshape(xx.shape), levels
    =20, alpha=0.8, cmap='PuBu')
axes[1].scatter(X_train_pca[y_train==0,0], X_train_pca[y_train==0,1], c='white
    ', marker='o', edgecolors='black', s=30, label='Benign')
axes[1].scatter(X_train_pca[y_train==1,0], X_train_pca[y_train==1,1], c='white
    ', marker='^', edgecolors='black', s=30, label='Malignant')
axes[1].set_title('Laplace: Predictive Variance')
cbar2 = plt.colorbar(im2, ax=axes[1]); cbar2.set_label('Variance')

plt.tight_layout()
plt.savefig("variance_ep_vs_laplace.png", dpi=300)
plt.show()
```

```
141
142 # 9. Variance statistics
143 print("EP Mean Variance:", np.mean(f_grid_var_ep))
144 print("Laplace Mean Variance:", np.mean(f_grid_var_laplace))
```

7.11 Simulation-Based Calibration for Gaussian Processes Regression on a synthetic dataset

Listing 15: Simulation-Based Calibration for Gaussian Process Regression on a synthetic dataset.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # --- RBF kernel and GP utilities ---
5  def rbf_kernel(X, Z, lengthscale=0.5, variance=1.0):
6      X = np.atleast_2d(X).astype(float)
7      Z = np.atleast_2d(Z).astype(float)
8      X2 = np.sum(X**2, axis=1)[:, None]
9      Z2 = np.sum(Z**2, axis=1)[None, :]
10     d2 = X2 + Z2 - 2 * X @ Z.T
11     return variance * np.exp(-0.5 * d2 / (lengthscale**2))
12
13 def gp_posterior(X, y, Xstar, sigma, lengthscale=0.5, variance=1.0, jitter=1e
       -8):
14     K_xx = rbf_kernel(X, X, lengthscale, variance)
15     K_xs = rbf_kernel(X, Xstar, lengthscale, variance)
16     K_ss = rbf_kernel(Xstar, Xstar, lengthscale, variance)
17     n = X.shape[0]
18     A = K_xx + (sigma**2)*np.eye(n) + jitter*np.eye(n)
19     L = np.linalg.cholesky(A)
20     alpha = np.linalg.solve(L.T, np.linalg.solve(L, y))
21     m_post = K_xs.T @ alpha
22     V = np.linalg.solve(L.T, np.linalg.solve(L, K_xs))
23     K_post = K_ss - V.T @ K_xs
24     K_post = 0.5*(K_post + K_post.T)    # ensure symmetry
25     return m_post, K_post
26
27 def sample_joint_prior(X, Xstar, lengthscale=0.5, variance=1.0, jitter=1e-10,
       rng=None):
28     rng = np.random.default_rng(rng)
29     Xc = np.vstack([X, Xstar])
30     K = rbf_kernel(Xc, Xc, lengthscale, variance)
31     K += jitter*np.eye(K.shape[0])
32     L = np.linalg.cholesky(K)
33     z = rng.standard_normal(K.shape[0])
34     f = L @ z
35     n = X.shape[0]
36     return f[:n], f[n:]
37
38 # --- SBC utilities ---
39 def sbc_once(X, Xstar, sigma, L_draws, lengthscale=0.5, variance=1.0, rng=None
       ):
40     rng = np.random.default_rng(rng)
41     fX_true, fstar_true = sample_joint_prior(X, Xstar, lengthscale, variance,
           rng=rng)
42     y_tilde = fX_true + sigma * rng.standard_normal(X.shape[0])
```

```
    m_post, K_post = gp_posterior(X, y_tilde, Xstar, sigma, lengthscale,
        variance)
    L_chol = np.linalg.cholesky(K_post + 1e-12*np.eye(K_post.shape[0]))
    Z = rng.standard_normal((K_post.shape[0], L_draws))
    F_samps = m_post[:, None] + L_chol @ Z  # (m, L)
    ranks = np.sum(F_samps < fstar_true[:, None], axis=1)  # rank statistic
    return ranks

def run_sbc(X, Xstar, sigma, N_iter, L_draws, lengthscale=0.5, variance=1.0,
    rng=None):
    rng = np.random.default_rng(rng)
    all_ranks = []
    for _ in range(N_iter):
        ranks = sbc_once(X, Xstar, sigma, L_draws, lengthscale, variance, rng=
            rng)
        all_ranks.append(ranks)
    return np.concatenate(all_ranks)

# --- Experimental setup ---
rng = np.random.default_rng(123)
X = np.linspace(-2.0, 2.0, 25)[:, None]
Xstar = np.array([[0.0]])   # single fixed test location
sigma = 0.15
N_iter = 800
L_draws = 50

ranks_syn = run_sbc(X, Xstar, sigma, N_iter, L_draws, lengthscale=0.5,
    variance=1.0, rng=rng)

# --- SBC histogram ---
bins = np.arange(-0.5, L_draws+1.5, 1.0)
plt.figure(figsize=(7,4.5))
plt.hist(ranks_syn, bins=bins, edgecolor='black')
plt.xlabel('Rank r in {0, ..., L}')
plt.ylabel('Count')
plt.title('SBC Rank Histogram     GP Regression')
plt.tight_layout()
plt.savefig('sbc_hist_synthetic.png', dpi=160)
plt.show()

print('Total ranks:', ranks_syn.size, ' Mean rank:', ranks_syn.mean())
```

7.12 Simulation-Based Calibration for Gaussian Processes Classification using Laplace and EP approximations

7.12.1 Simulation-Based Calibration for Gaussian Processes Classification using Laplace and EP approximations (synthetic data first experiment)

Listing 16: Simulation-Based Calibration for Gaussian Processes Classification using Laplace and EP approximations (synthetic data first experiment)

```
import numpy as np
from scipy.stats import norm
from scipy.linalg import cho_factor, cho_solve
import matplotlib.pyplot as plt
rng = np.random.default_rng(0)
```

```

# Probit link
def phi(z):
    z = np.clip(z, -10, 10)
    return norm.cdf(z)

# Squared Exponential (RBF) kernel
def se_kernel(X1, X2, lengthscale=1.0, variance=1.0):
    dists = np.sum((X1[:,None,:] - X2[None,:,:])**2, axis=2)
    return variance * np.exp(-0.5 * dists / lengthscale**2)

# Generate synthetic training and test inputs
def get_data():
    X = np.linspace(-2, 2, 25)[:,None]
    x_star = np.array([[0.0]])  # test location at 0
    return X, x_star

# Laplace approximation predictive mean and variance
def laplace_predictive(X, x_star, y, ell=1.0, sf2=1.0):
    K = se_kernel(X, X, ell, sf2) + 1e-6*np.eye(len(X))
    f = np.zeros(len(y))
    for _ in range(20):
        t = (2*y-1)*f
        r = norm.pdf(np.clip(t,-10,10))/np.maximum(phi(t), 1e-12)
        g = r*(2*y-1)
        h = -r*(t + r)
        W = -h
        sqrtW = np.sqrt(np.maximum(W, 1e-12))
        B = np.eye(len(X)) + (sqrtW[:,None]*K)*sqrtW[None,:]
        L = cho_factor(B)
        b = g - W*f
        Kb = K @ b
        v = cho_solve(L, sqrtW*Kb)
        f += Kb - (K*(sqrtW*v)[:,None]).sum(axis=1)
        if np.linalg.norm(Kb) < 1e-6:
            break
    B = np.eye(len(X)) + (sqrtW[:,None]*K)*sqrtW[None,:]
    L = cho_factor(B)
    tmp = cho_solve(L, (sqrtW[:,None]*K).T)
    Sigma = K - (K*sqrtW) @ tmp
    Kxs = se_kernel(X, x_star, ell, sf2)
    Kss = se_kernel(x_star, x_star, ell, sf2)
    Kinv = cho_solve(cho_factor(K), np.eye(len(X)))
    m_star = (Kxs.T @ (Kinv @ f)).ravel()
    v_star = (Kss - Kxs.T @ (Kinv - Kinv @ Sigma @ Kinv) @ Kxs).ravel()
    return m_star, np.maximum(v_star, 1e-12)

# Expectation Propagation predictive mean and variance
def ep_predictive(X, x_star, y, ell=1.0, sf2=1.0):
    K = se_kernel(X, X, ell, sf2) + 1e-6*np.eye(len(X))
    n = len(y)
    tau = np.zeros(n)
    nu = np.zeros(n)
    Kinv = np.linalg.inv(K)
    for _ in range(50):
        Sigma = np.linalg.inv(Kinv + np.diag(tau))
```

```
        mu = Sigma @ nu
        for i in range(n):
            tau_cav = 1.0/np.maximum(Sigma[i,i], 1e-12) - tau[i]
            nu_cav = mu[i]/np.maximum(Sigma[i,i], 1e-12) - nu[i]
            tau_cav = np.maximum(tau_cav, 1e-12)
            v_cav = 1.0/tau_cav
            m_cav = v_cav*nu_cav
            t = (2*y[i]-1)*m_cav/np.sqrt(1+v_cav)
            Z = np.maximum(phi(t), 1e-12)
            ratio = norm.pdf(np.clip(t,-10,10))/Z
            dm = (2*y[i]-1)*v_cav/np.sqrt(1+v_cav)*ratio
            dv = v_cav - v_cav**2 * ratio * ((t+ratio)/(1+v_cav))
            v_hat = np.maximum(dv, 1e-12)
            m_hat = m_cav + dm
            tau[i] = 1.0/v_hat - tau_cav
            nu[i] = m_hat/v_hat - nu_cav
    Sigma = np.linalg.inv(Kinv + np.diag(tau))
    mu = Sigma @ nu
    Kxs = se_kernel(X, x_star, ell, sf2)
    Kss = se_kernel(x_star, x_star, ell, sf2)
    m_star = (Kxs.T @ (Kinv @ mu)).ravel()
    v_star = (Kss - Kxs.T @ (Kinv - Kinv @ Sigma @ Kinv) @ Kxs).ravel()
    return m_star, np.maximum(v_star, 1e-12)

# SBC procedure with synthetic GP data
def sbc(method, N=200, L=50):
    X, x_star = get_data()
    ranks = []
    for _ in range(N):
        # Sample joint prior (f_X, f_star)
        K = se_kernel(X, X)
        K_joint = np.block([[K, se_kernel(X,x_star)],
                            [se_kernel(x_star,X), se_kernel(x_star,x_star)]])
        Lchol = np.linalg.cholesky(K_joint + 1e-9*np.eye(len(X)+1))
        f_joint = Lchol @ rng.standard_normal(len(X)+1)
        fX, f_star_true = f_joint[:-1], f_joint[-1]
        # Generate synthetic labels
        y = rng.binomial(1, np.clip(phi(fX), 1e-9, 1-1e-9))
        # Posterior predictive
        m_star, v_star = method(X, x_star, y)
        samples = rng.normal(m_star, np.sqrt(v_star), size=L)
        r = np.sum(samples < f_star_true)
        ranks.append(r)
    return np.array(ranks)

# Run SBC with Laplace and EP
for name, method in [('Laplace', laplace_predictive), ('EP', ep_predictive)]:
    ranks = sbc(method, N=200, L=50)
    np.save(f"ranks_{name.lower()}_synthetic.npy", ranks)
    plt.hist(ranks, bins=np.arange(-0.5, 51.5, 1), edgecolor='k')
    plt.title(f"SBC Rank Histogram     {name}")
    plt.xlabel("Rank")
    plt.ylabel("Frequency")
    plt.savefig(f"sbc_hist_synthetic_{name.lower()}.png", dpi=300, bbox_inches
        ='tight')
    plt.close()
```

```
    print(f"{name}: mean rank = {np.mean(ranks):.2f} (theoretical 25.00)")
```

7.12.2 Simulation-Based Calibration for Gaussian Processes Classification using Laplace and EP approximations (synthetic data second experiment)

Listing 17: Simulation-Based Calibration for Gaussian Processes Classification using Laplace and EP approximations (synthetic data second experiment)

```
import numpy as np
from scipy.stats import norm
from scipy.linalg import cho_factor, cho_solve
import matplotlib.pyplot as plt
rng = np.random.default_rng(0)

# Probit link
def phi(z):
    z = np.clip(z, -10, 10)
    return norm.cdf(z)

# Squared Exponential (RBF) kernel
def se_kernel(X1, X2, lengthscale=1.0, variance=1.0):
    dists = np.sum((X1[:,None,:] - X2[None,:,:])**2, axis=2)
    return variance * np.exp(-0.5 * dists / lengthscale**2)

# Generate synthetic training and test inputs
def get_data():
    X = np.linspace(-30, 30, 25)[:,None]
    x_star = np.array([[0.0]])  # test location at 0
    return X, x_star

# Laplace approximation predictive mean and variance
def laplace_predictive(X, x_star, y, ell=1.0, sf2=1.0):
    K = se_kernel(X, X, ell, sf2) + 1e-6*np.eye(len(X))
    f = np.zeros(len(y))
    for _ in range(20):
        t = (2*y-1)*f
        r = norm.pdf(np.clip(t,-10,10))/np.maximum(phi(t), 1e-12)
        g = r*(2*y-1)
        h = -r*(t + r)
        W = -h
        sqrtW = np.sqrt(np.maximum(W, 1e-12))
        B = np.eye(len(X)) + (sqrtW[:,None]*K)*sqrtW[None,:]
        L = cho_factor(B)
        b = g - W*f
        Kb = K @ b
        v = cho_solve(L, sqrtW*Kb)
        f += Kb - (K*(sqrtW*v)[:,None]).sum(axis=1)
        if np.linalg.norm(Kb) < 1e-6:
            break
    B = np.eye(len(X)) + (sqrtW[:,None]*K)*sqrtW[None,:]
    L = cho_factor(B)
    tmp = cho_solve(L, (sqrtW[:,None]*K).T)
    Sigma = K - (K*sqrtW) @ tmp
    Kxs = se_kernel(X, x_star, ell, sf2)
    Kss = se_kernel(x_star, x_star, ell, sf2)
    Kinv = cho_solve(cho_factor(K), np.eye(len(X)))
```

```
    m_star = (Kxs.T @ (Kinv @ f)).ravel()
    v_star = (Kss - Kxs.T @ (Kinv - Kinv @ Sigma @ Kinv) @ Kxs).ravel()
    return m_star, np.maximum(v_star, 1e-12)

# Expectation Propagation predictive mean and variance
def ep_predictive(X, x_star, y, ell=1.0, sf2=1.0):
    K = se_kernel(X, X, ell, sf2) + 1e-6*np.eye(len(X))
    n = len(y)
    tau = np.zeros(n)
    nu = np.zeros(n)
    Kinv = np.linalg.inv(K)
    for _ in range(50):
        Sigma = np.linalg.inv(Kinv + np.diag(tau))
        mu = Sigma @ nu
        for i in range(n):
            tau_cav = 1.0/np.maximum(Sigma[i,i], 1e-12) - tau[i]
            nu_cav = mu[i]/np.maximum(Sigma[i,i], 1e-12) - nu[i]
            tau_cav = np.maximum(tau_cav, 1e-12)
            v_cav = 1.0/tau_cav
            m_cav = v_cav*nu_cav
            t = (2*y[i]-1)*m_cav/np.sqrt(1+v_cav)
            Z = np.maximum(phi(t), 1e-12)
            ratio = norm.pdf(np.clip(t,-10,10))/Z
            dm = (2*y[i]-1)*v_cav/np.sqrt(1+v_cav)*ratio
            dv = v_cav - v_cav**2 * ratio * ((t+ratio)/(1+v_cav))
            v_hat = np.maximum(dv, 1e-12)
            m_hat = m_cav + dm
            tau[i] = 1.0/v_hat - tau_cav
            nu[i] = m_hat/v_hat - nu_cav
    Sigma = np.linalg.inv(Kinv + np.diag(tau))
    mu = Sigma @ nu
    Kxs = se_kernel(X, x_star, ell, sf2)
    Kss = se_kernel(x_star, x_star, ell, sf2)
    m_star = (Kxs.T @ (Kinv @ mu)).ravel()
    v_star = (Kss - Kxs.T @ (Kinv - Kinv @ Sigma @ Kinv) @ Kxs).ravel()
    return m_star, np.maximum(v_star, 1e-12)

# SBC procedure with synthetic GP data
def sbc(method, N=200, L=50):
    X, x_star = get_data()
    ranks = []
    for _ in range(N):
        # Sample joint prior (f_X, f_star)
        K = se_kernel(X, X)
        K_joint = np.block([[K, se_kernel(X,x_star)],
                            [se_kernel(x_star,X), se_kernel(x_star,x_star)]])
        Lchol = np.linalg.cholesky(K_joint + 1e-9*np.eye(len(X)+1))
        f_joint = Lchol @ rng.standard_normal(len(X)+1)
        fX, f_star_true = f_joint[:-1], f_joint[-1]
        # Generate synthetic labels
        y = rng.binomial(1, np.clip(phi(fX), 1e-9, 1-1e-9))
        # Posterior predictive
        m_star, v_star = method(X, x_star, y)
        samples = rng.normal(m_star, np.sqrt(v_star), size=L)
        r = np.sum(samples < f_star_true)
        ranks.append(r)
```

```
    return np.array(ranks)

# Run SBC with Laplace and EP
for name, method in [('Laplace', laplace_predictive), ('EP', ep_predictive)]:
    ranks = sbc(method, N=200, L=50)
    np.save(f"ranks_{name.lower()}_synthetic.npy", ranks)
    plt.hist(ranks, bins=np.arange(-0.5, 51.5, 1), edgecolor='k')
    plt.title(f"SBC Rank Histogram     {name}")
    plt.xlabel("Rank")
    plt.ylabel("Frequency")
    plt.savefig(f"sbc_hist_synthetic_{name.lower()}_v2.png", dpi=300,
        bbox_inches='tight')
    plt.close()
    print(f"{name}: mean rank = {np.mean(ranks):.2f} (theoretical 25.00)")
```

References

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. URL: https://www.springer.com/gp/book/9780387310732.

McLeod, John and Fergus Simpson (2021). *Validating Gaussian Process Models with Simulation-Based Calibration*. arXiv: 2110.15049 [stat.ML]. URL: https://arxiv.org/abs/2110.15049.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. URL: https://mitpress.mit.edu/9780262018029/machine-learning/.

Rasmussen, Carl E. and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press. URL: http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

Talts, Sean et al. (2018). *Validating Bayesian Inference Algorithms with Simulation-Based Calibration*. arXiv: 1804.06788 [stat.ME]. URL: https://arxiv.org/abs/1804.06788.