# Lecture 11 & 14: Parameter Estimation & Bayesian Inference
# Notes

Konstantinos Chatzilygeroudis
costashatz@upatras.gr

December 22, 2025

## Contents

**1 Motivation for Parameter Estimation** **3**

**2 Linear Least Squares** **3**
2.1 Derivation . . . . . 4
2.2 Least Squares Estimation of AR Model Parameters . . . . . 5

**3 Overfitting and Regularization** **6**

**4 Nonlinear Least Squares** **6**
4.1 ARMA Parameter Estimation via Prediction Error Minimization . . . . . 6
4.2 Gauss–Newton for Nonlinear Least Squares . . . . . 8
4.3 Gauss–Newton in General Form . . . . . 9
4.4 Gauss–Newton Example: Estimating an ARMA(1,1) Model . . . . . 9
4.5 Worked Mini-Example (One Gauss–Newton Step) . . . . . 10
4.6 Practical View: Nonlinear LS via an Optimization Lens . . . . . 11

**5 Nonlinear Autoregressive Models and Linear Models on Features** **12**
5.1 NAR Models . . . . . 12
5.2 Linear Models on Features (Basis Expansion) . . . . . 13
5.3 NAR as Linear Regression on Features (Special Case) . . . . . 14
5.4 Worked Examples: Linear Models on Features . . . . . 14
5.5 Python Implemetation . . . . . 15

**6 Bayesian Inference** **19**
6.1 Core Idea . . . . . 19
6.2 Likelihood . . . . . 19
6.3 Bayes' Rule and the Posterior . . . . . 20
6.4 Interpretation of the Bayesian Quantities . . . . . 20
6.5 Bayesian Point Estimation via Risk Minimization . . . . . 21
6.5.1 MAP Estimator (Posterior Mode) . . . . . 21
6.5.2 MMSE Estimator (Posterior Mean) . . . . . 22

6.5.3 MMAE Estimator (Posterior Median) . . . . . 23
6.5.4 Summary: MAP vs. MMSE vs. MMAE . . . . . 23
6.6 Maximum Likelihood Estimation (MLE) . . . . . 24
6.6.1 Worked Example: MLE for a Gaussian Mean (Known Variance) . 25
6.6.2 Worked Example: MLE for Linear Regression with Unknown Variance . . . . . 26

**7 Bayesian Linear Regression** **28**
7.1 Model, Notation, and Goals . . . . . 28
7.2 Prior and Likelihood . . . . . 28
7.3 Posterior over Weights . . . . . 28
7.4 MAP Connection: BLR as Regularized Least Squares . . . . . 30
7.5 Predictive Distributions . . . . . 30
7.6 Practical Note: Sampling from the Prior (Function Space Intuition) . . . 31
7.7 Worked Example: Bayesian Linear Regression (1D, Closed Form) . . . . 32
7.8 Python Implementation . . . . . 34
7.9 Sequential Bayesian Linear Regression (Recursive Update) . . . . . 38
7.10 What Bayesian Linear Regression Adds Beyond LS / MLE . . . . . 39

# 1 Motivation for Parameter Estimation

Up until now, our focus has been on *signal analysis*: given a measured signal, we aimed to characterize its properties through spectral analysis, correlation functions, and filtering. We introduced non-parametric tools such as the periodogram and Welch method, discussed windowing effects, leakage, and resolution limits, and analyzed how stochastic processes and noise propagate through LTI systems and FIR/IIR filters.

While these methods are powerful, non-parametric descriptions can become unreliable when data records are short, measurements are noisy or colored, or high spectral resolution is required. In such cases, spectral estimates often exhibit high variance and fail to exploit any underlying structure in the signal generation process.

Parametric estimation addresses these limitations by explicitly assuming that the observed signal is generated by a model with a finite number of parameters. This view is already implicit in filtering, where signals are modeled as the output of a system driven by inputs and noise. Estimating the parameters of such models can lead to more compact representations, improved robustness to noise, and better generalization beyond the observed data.

The goal of this chapter is therefore to move from analyzing signals to **estimating the models that generate them**. We will introduce parametric model families, and study fundamental estimation principles including least squares, maximum likelihood, maximum a posteriori estimation, and Bayesian inference.

# 2 Linear Least Squares

Least Squares (LS) estimation is the basic tool for fitting linear parametric models to data. We assume a linear observation model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{w},$$

where $\boldsymbol{y} \in \mathbb{R}^N$ is the vector of measurements, $\boldsymbol{X} \in \mathbb{R}^{N\times p}$ is a known design (regressor) matrix constructed from inputs and/or past outputs, $\boldsymbol{\theta} \in \mathbb{R}^p$ is the unknown parameter vector, and $\boldsymbol{w} \in \mathbb{R}^N$ captures measurement noise and modeling error. The LS principle chooses parameters that minimize the total squared residual (prediction error),

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2.$$

Geometrically, $\boldsymbol{X}\hat{\boldsymbol{\theta}}$ is the orthogonal projection of $\boldsymbol{y}$ onto the column space of $\boldsymbol{X}$, and the residual $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\theta}}$ is orthogonal to that subspace. Setting the gradient of the cost to zero yields the normal equations,

$$\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\,\hat{\boldsymbol{\theta}} = \boldsymbol{X}^\mathsf{T}\boldsymbol{y}.$$

If $\boldsymbol{X}$ has full column rank (so $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is invertible), the unique minimizer is

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}.$$

If $\boldsymbol{X}$ does not have full column rank, there are infinitely many minimizers; the standard choice is the minimum-norm solution given by the Moore–Penrose pseudo-inverse,

$$\hat{\boldsymbol{\theta}} = \boldsymbol{X}^\dagger\boldsymbol{y}.$$

Under standard statistical assumptions, LS has important optimality properties. If the model is correct and $E[\boldsymbol{w}] = \mathbf{0}$, then $\hat{\boldsymbol{\theta}}$ is unbiased. If $\boldsymbol{w}$ is i.i.d. Gaussian with covariance $\sigma^2\boldsymbol{I}$, then LS coincides with the maximum likelihood estimator, and among all linear unbiased estimators it achieves minimum variance (Gauss–Markov theorem; in the Gaussian case it is also efficient in the class of unbiased estimators). These ideas form the foundation of linear regression, identification of AR/ARMA models (via linear-in-parameters formulations), and many estimation algorithms used in filtering and state-space modeling.

## 2.1 Derivation

We start from the linear observation model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{w},$$

where the vector

$$\boldsymbol{w} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}$$

represents the residual, capturing measurement noise and modeling error for a given choice of parameters $\boldsymbol{\theta}$.

The least-squares principle selects the parameter vector that minimizes the energy of this residual. This leads to the cost function

$$J(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}).$$

Expanding the quadratic form gives

$$J(\boldsymbol{\theta}) = \boldsymbol{y}^\mathsf{T}\boldsymbol{y} - 2\,\boldsymbol{\theta}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{\theta}.$$

Since this is a convex quadratic function of $\boldsymbol{\theta}$, its minimum is found by setting its gradient with respect to $\boldsymbol{\theta}$ equal to zero:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -2\,\boldsymbol{X}^\mathsf{T}\boldsymbol{y} + 2\,\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{\theta} = \mathbf{0}.$$

This yields the normal equations

$$\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\,\hat{\boldsymbol{\theta}} = \boldsymbol{X}^\mathsf{T}\boldsymbol{y}.$$

If $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is invertible (equivalently, if $\boldsymbol{X}$ has full column rank), the unique least-squares solution is

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}.$$

An important geometric interpretation follows directly from the normal equations. Defining the estimated residual

$$\hat{\boldsymbol{w}} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\theta}},$$

we obtain the orthogonality condition

$$\boldsymbol{X}^\mathsf{T}\hat{\boldsymbol{w}} = \mathbf{0}.$$

Thus, the least-squares residual is orthogonal to the column space of $\boldsymbol{X}$, meaning that the fitted vector $\boldsymbol{X}\hat{\boldsymbol{\theta}}$ is the orthogonal projection of $\boldsymbol{y}$ onto that subspace.

## 2.2 Least Squares Estimation of AR Model Parameters

Consider an autoregressive model of order $p$,

$$x[k] = -\sum_{i=1}^{p} a_i \, x[k-i] + w[k],$$

where $w[k]$ is the one-step-ahead prediction error (residual). The unknown parameters are the AR coefficients collected in

$$\boldsymbol{a} = [a_1, \ldots, a_p]^{\mathsf{T}}.$$

A key point is that the AR model is *linear in the parameters* $a_i$. For each time index $k$ we can write a linear regression equation

$$x[k] = -\boldsymbol{a}^{\mathsf{T}} \boldsymbol{x}_{\text{past}}[k] + w[k], \qquad \boldsymbol{x}_{\text{past}}[k] = \begin{bmatrix} x[k-1] \\ x[k-2] \\ \vdots \\ x[k-p] \end{bmatrix}.$$

Thus, each sample $x[k]$ (for $k \geq p$) provides one linear equation in the unknown vector $\boldsymbol{a}$.

To estimate $\boldsymbol{a}$ from a record $\{x[0], x[1], \ldots, x[N-1]\}$, we stack the equations for $k = p, \ldots, N-1$ into matrix–vector form:

$$\boldsymbol{y} = -\boldsymbol{X}\boldsymbol{a} + \boldsymbol{w},$$

with

$$\boldsymbol{y} = \begin{bmatrix} x[p] \\ x[p+1] \\ \vdots \\ x[N-1] \end{bmatrix}, \qquad \boldsymbol{X} = \begin{bmatrix} x[p-1] & x[p-2] & \cdots & x[p-p] \\ x[p] & x[p-1] & \cdots & x[p-p+1] \\ \vdots & \vdots & \ddots & \vdots \\ x[N-2] & x[N-3] & \cdots & x[N-1-p] \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} w[p] \\ w[p+1] \\ \vdots \\ w[N-1] \end{bmatrix}.$$

The least-squares estimate minimizes the sum of squared one-step prediction errors:

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \|\boldsymbol{y} + \boldsymbol{X}\boldsymbol{a}\|_2^2.$$

Solving the corresponding normal equations yields (when $\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X}$ is invertible)

$$\hat{\boldsymbol{a}} = -(\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{y},$$

and, more generally, $\hat{\boldsymbol{a}} = -\boldsymbol{X}^{\dagger}\boldsymbol{y}$ using the pseudo-inverse.

It is useful to contrast this with the Yule–Walker (YW) approach. LS estimates the coefficients by directly fitting sample-by-sample prediction equations, whereas YW estimates autocorrelations and then solves a Toeplitz linear system derived from the AR assumptions. When the data truly follow an AR($p$) model and are wide-sense stationary, the two approaches are asymptotically equivalent, but for short data records or mild model mismatch, LS can be more accurate because it avoids the intermediate step of estimating autocorrelations.

# 3 Overfitting and Regularization

In linear regression, increasing the number of parameters (or using a very rich set of regressors) increases the flexibility of the model. While this can reduce the training error, it also increases the risk of *overfitting*: the least-squares solution may start fitting not only the underlying signal structure but also the noise present in the measurements. Typical symptoms include large-magnitude coefficients, strong sensitivity to small perturbations in the data, and unstable predictions. As a result, a model that fits the available data extremely well can still generalize poorly to new data, producing unrealistic or highly oscillatory behavior.

A standard way to mitigate overfitting is to introduce *regularization*, which explicitly penalizes overly complex solutions. The most common choice in linear models is *ridge* (or *Tikhonov*) regularization, where we add an $\ell_2$ penalty on the parameter vector:

$$\hat{\boldsymbol{\theta}}_{\text{reg}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_2^2, \qquad \lambda > 0.$$

The parameter $\lambda$ controls the trade-off between fitting the data and keeping the parameter vector small. For $\lambda \to 0$ the solution approaches ordinary least squares, while larger $\lambda$ increasingly shrinks the coefficients toward zero.

Ridge regression admits a closed-form solution:

$$\hat{\boldsymbol{\theta}}_{\text{reg}} = \left(\boldsymbol{X}^\mathsf{T}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^\mathsf{T}\boldsymbol{y}.$$

Regularization has several important effects. First, it reduces the variance of the estimator by shrinking parameter magnitudes, often improving prediction performance when data are noisy or limited. Second, it improves numerical stability: even if $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is ill-conditioned (or singular), adding $\lambda \boldsymbol{I}$ makes the system better conditioned and typically invertible. Finally, regularization reduces sensitivity to noise and small modeling errors in the regressors.

Figure 1 illustrates the phenomenon on a toy example: a high-degree polynomial fitted by ordinary least squares oscillates strongly to interpolate noisy samples, while ridge regularization produces a smoother curve that better captures the underlying function.

# 4 Nonlinear Least Squares

## 4.1 ARMA Parameter Estimation via Prediction Error Minimization

Autoregressive–moving average (ARMA) models extend AR models by introducing a moving-average (MA) component that captures correlation structure in the residuals. An ARMA$(p, q)$ process can be written as

$$x[k] = -\sum_{i=1}^{p} a_i\, x[k-i] + \sum_{j=1}^{q} b_j\, w[k-j] + w[k], \qquad w[k] : \text{innovation/white noise.}$$

The unknown parameter vector is

$$\boldsymbol{\theta} = \begin{bmatrix} a_1 & \cdots & a_p & b_1 & \cdots & b_q \end{bmatrix}^\mathsf{T} \in \mathbb{R}^{p+q}.$$

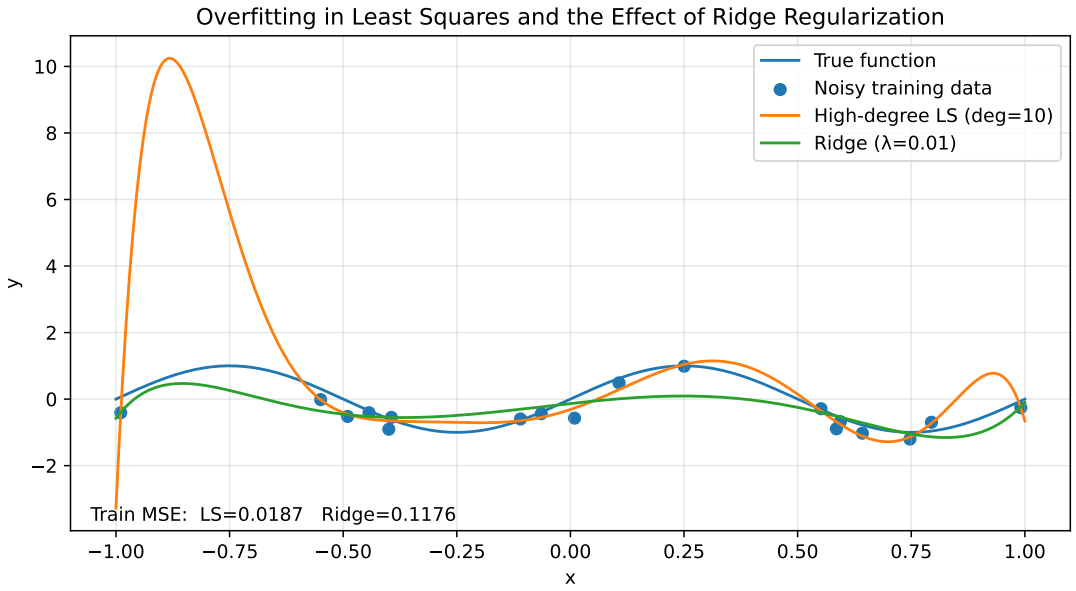



Figure 1: Overfitting in ordinary least squares (high-degree polynomial) and mitigation via ridge regularization. The LS fit tends to match noise and oscillates between samples, whereas ridge shrinks coefficients and yields a more stable, better-generalizing model.

**Why this becomes nonlinear.** For a pure AR($p$) model, the regressors $x[k-1], \ldots, x[k-p]$ are known from data, so the model is linear in the unknown coefficients and can be written in the form $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{a} + \boldsymbol{w}$, yielding a closed-form LS solution. In ARMA, the MA part depends on past innovations $w[k-j]$, which are not observable. If we try to rewrite the model in terms of measurable quantities, we obtain a recursion for the one-step prediction error (innovation estimate). Starting from

$$w[k] = x[k] + \sum_{i=1}^{p} a_i\, x[k-i] - \sum_{j=1}^{q} b_j\, w[k-j],$$

we define the prediction error (residual) as a function of the parameters,

$$\varepsilon(k, \boldsymbol{\theta}) = x[k] + \sum_{i=1}^{p} a_i\, x[k-i] - \sum_{j=1}^{q} b_j\, \varepsilon(k-j, \boldsymbol{\theta}),$$

with suitable initial conditions for $\varepsilon(k, \boldsymbol{\theta})$ for $k < 0$ (often set to zero) and with the first usable index

$$k_0 = \max(p, q),$$

so that all required past samples and past errors are available. The key point is that $\varepsilon(k, \boldsymbol{\theta})$ depends on previous $\varepsilon(k-j, \boldsymbol{\theta})$, which themselves depend on $\boldsymbol{\theta}$. Therefore, the residual sequence is a *nonlinear function* of the parameters, and we cannot cast the problem as a single linear system $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}$.

**Nonlinear least squares objective.** A standard approach is *prediction error minimization*, which chooses parameters that minimize the sum of squared one-step-ahead

prediction errors:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{k=k_0}^{N-1} \varepsilon(k, \boldsymbol{\theta})^2.$$

Defining the stacked residual vector

$$\boldsymbol{\varepsilon}(\boldsymbol{\theta}) = \begin{bmatrix} \varepsilon(k_0, \boldsymbol{\theta}) \\ \varepsilon(k_0 + 1, \boldsymbol{\theta}) \\ \vdots \\ \varepsilon(N-1, \boldsymbol{\theta}) \end{bmatrix} \in \mathbb{R}^M, \qquad M = N - k_0,$$

we can write

$$J(\boldsymbol{\theta}) = \frac{1}{2} \boldsymbol{\varepsilon}(\boldsymbol{\theta})^\mathsf{T} \boldsymbol{\varepsilon}(\boldsymbol{\theta}) = \frac{1}{2} \left\| \boldsymbol{\varepsilon}(\boldsymbol{\theta}) \right\|_2^2, \qquad \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}).$$

Unlike linear LS, this optimization problem generally has no closed-form solution and must be solved iteratively.

## 4.2 Gauss–Newton for Nonlinear Least Squares

Gauss–Newton is a classical method for minimizing least-squares objectives. It proceeds by locally linearizing the residual vector around the current estimate $\boldsymbol{\theta}_i$. Let the Jacobian of the residual vector be

$$\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{\varepsilon}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{M \times P}, \qquad P = p + q, \qquad \left[\boldsymbol{J}_\varepsilon(\boldsymbol{\theta})\right]_{m,\ell} = \frac{\partial \varepsilon(k_0 + m, \boldsymbol{\theta})}{\partial \theta_\ell}.$$

A first-order Taylor approximation gives

$$\boldsymbol{\varepsilon}(\boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}) \approx \boldsymbol{\varepsilon}(\boldsymbol{\theta}_i) + \boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)\, \Delta\boldsymbol{\theta}.$$

Gauss–Newton then chooses $\Delta\boldsymbol{\theta}_i$ as the minimizer of the resulting linear least-squares subproblem:

$$\Delta\boldsymbol{\theta}_i = \arg\min_{\Delta\boldsymbol{\theta}} \left\| \boldsymbol{\varepsilon}(\boldsymbol{\theta}_i) + \boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)\, \Delta\boldsymbol{\theta} \right\|_2^2.$$

This leads to the normal equations

$$\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)^\mathsf{T} \boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)\, \Delta\boldsymbol{\theta}_i = -\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)^\mathsf{T} \boldsymbol{\varepsilon}(\boldsymbol{\theta}_i),$$

followed by the parameter update

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i,$$

optionally combined with damping or line search to improve robustness.

**Interpretation.** At each iteration, Gauss–Newton replaces the nonlinear residual mapping by its local linear approximation and solves a linear least-squares problem. The method also admits a useful second-order interpretation: for least-squares objectives,

$$\nabla J(\boldsymbol{\theta}) = \boldsymbol{J}_\varepsilon(\boldsymbol{\theta})^\mathsf{T} \boldsymbol{\varepsilon}(\boldsymbol{\theta}), \qquad \nabla^2 J(\boldsymbol{\theta}) = \boldsymbol{J}_\varepsilon^\mathsf{T} \boldsymbol{J}_\varepsilon + \sum_{m=1}^{M} \varepsilon_m(\boldsymbol{\theta})\, \nabla^2 \varepsilon_m(\boldsymbol{\theta}),$$

and Gauss–Newton approximates the Hessian by dropping the second term, yielding

$$\nabla^2 J(\boldsymbol{\theta}) \approx \boldsymbol{J}_\varepsilon(\boldsymbol{\theta})^\mathsf{T} \boldsymbol{J}_\varepsilon(\boldsymbol{\theta}),$$

which avoids computing second derivatives while exploiting the least-squares structure.

## 4.3 Gauss–Newton in General Form

More generally, given residuals $\boldsymbol{r}(\boldsymbol{\theta}) \in \mathbb{R}^M$ and objective

$$J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{\theta})\|_2^2 = \boldsymbol{r}(\boldsymbol{\theta})^T \boldsymbol{r}(\boldsymbol{\theta}),$$

with Jacobian

$$\boldsymbol{J}_r(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{r}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{M \times P},$$

Gauss–Newton computes $\Delta\boldsymbol{\theta}_i$ from

$$\boldsymbol{J}_r(\boldsymbol{\theta}_i)^{\mathsf{T}} \boldsymbol{J}_r(\boldsymbol{\theta}_i)\, \Delta\boldsymbol{\theta}_i = -\boldsymbol{J}_r(\boldsymbol{\theta}_i)^{\mathsf{T}} \boldsymbol{r}(\boldsymbol{\theta}_i), \qquad \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i.$$

In ARMA estimation, $\boldsymbol{r}(\boldsymbol{\theta})$ corresponds to the stacked prediction errors $\boldsymbol{\varepsilon}(\boldsymbol{\theta})$, and the method iteratively refines the coefficients to minimize one-step-ahead prediction error energy.

## 4.4 Gauss–Newton Example: Estimating an ARMA(1,1) Model

Consider the ARMA(1,1) model

$$x[k] = -a\, x[k-1] + b\, w[k-1] + w[k], \qquad w[k] \text{ innovation.}$$

Because the past innovations $w[k-1]$ are not observable, we work with the *one-step-ahead prediction error* (innovation estimate)

$$\varepsilon(k, \boldsymbol{\theta}) = x[k] + a\, x[k-1] - b\, \varepsilon(k-1, \boldsymbol{\theta}), \qquad \boldsymbol{\theta} = \begin{bmatrix} a \\ b \end{bmatrix},$$

with a standard initialization such as $\varepsilon(0, \boldsymbol{\theta}) = 0$. The nonlinear least-squares (prediction error) cost is

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{k=1}^{N-1} \varepsilon(k, \boldsymbol{\theta})^2 = \frac{1}{2} \big\|\boldsymbol{\varepsilon}(\boldsymbol{\theta})\big\|_2^2, \qquad \boldsymbol{\varepsilon}(\boldsymbol{\theta}) = \begin{bmatrix} \varepsilon(1, \boldsymbol{\theta}) \\ \vdots \\ \varepsilon(N-1, \boldsymbol{\theta}) \end{bmatrix}.$$

Since $\varepsilon(k, \boldsymbol{\theta})$ depends recursively on past errors, it is a nonlinear function of $(a, b)$, and we minimize $J(\boldsymbol{\theta})$ iteratively using Gauss–Newton.

**Jacobian via recursions.** Gauss–Newton requires the Jacobian of the residual vector:

$$\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{\varepsilon}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \partial_a \varepsilon(1, \boldsymbol{\theta}) & \partial_b \varepsilon(1, \boldsymbol{\theta}) \\ \vdots & \vdots \\ \partial_a \varepsilon(N-1, \boldsymbol{\theta}) & \partial_b \varepsilon(N-1, \boldsymbol{\theta}) \end{bmatrix}.$$

Differentiate the recursion $\varepsilon(k) = x[k] + a\, x[k-1] - b\, \varepsilon(k-1)$ to obtain forward recursions for the partial derivatives:

$$\partial_a \varepsilon(k) = x[k-1] - b\, \partial_a \varepsilon(k-1),$$
$$\partial_b \varepsilon(k) = -\varepsilon(k-1) - b\, \partial_b \varepsilon(k-1),$$

with initial conditions

$$\varepsilon(0) = 0, \qquad \partial_a \varepsilon(0) = 0, \qquad \partial_b \varepsilon(0) = 0.$$

Thus, for a fixed parameter guess $(a, b)$, we can compute $\varepsilon(k)$, $\partial_a \varepsilon(k)$, and $\partial_b \varepsilon(k)$ in a single forward pass through the data.

**Gauss–Newton iteration.** At iteration $i$, given $\boldsymbol{\theta}_i = [a_i \;\; b_i]^\mathsf{T}$, we compute $\boldsymbol{\varepsilon}(\boldsymbol{\theta}_i)$ and $\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)$, then solve the linear least-squares step

$$\Delta\boldsymbol{\theta}_i = \arg\min_{\Delta\boldsymbol{\theta}} \|\boldsymbol{\varepsilon}(\boldsymbol{\theta}_i) + \boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)\Delta\boldsymbol{\theta}\|_2^2 ,$$

which yields the normal equations

$$\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)^\mathsf{T}\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)\,\Delta\boldsymbol{\theta}_i = -\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_i)^\mathsf{T}\boldsymbol{\varepsilon}(\boldsymbol{\theta}_i).$$

The update is

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i,$$

and we stop when $\|\Delta\boldsymbol{\theta}_i\|$ (or the decrease in $J$) is small. In practice, damping/line-search can be added if the plain step is unstable.

## 4.5 Worked Mini-Example (One Gauss–Newton Step)

Take three samples:

$$x[0] = 1, \quad x[1] = 0.5, \quad x[2] = -0.2,$$

and initialize at

$$\boldsymbol{\theta}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Rightarrow \quad a_0 = 0,\ b_0 = 0, \qquad \varepsilon(0) = 0.$$

**Step 1: residuals at $\boldsymbol{\theta}_0$.** Using $\varepsilon(k) = x[k] + a\,x[k-1] - b\,\varepsilon(k-1)$ with $a_0 = b_0 = 0$:

$$\varepsilon(1, \boldsymbol{\theta}_0) = x[1] = 0.5, \qquad \varepsilon(2, \boldsymbol{\theta}_0) = x[2] = -0.2.$$

So

$$\boldsymbol{\varepsilon}(\boldsymbol{\theta}_0) = \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}.$$

**Step 2: Jacobian at $\boldsymbol{\theta}_0$.** With $b_0 = 0$, the derivative recursions simplify to

$$\partial_a\varepsilon(k) = x[k-1], \qquad \partial_b\varepsilon(k) = -\varepsilon(k-1).$$

Therefore,

$$\partial_a\varepsilon(1) = x[0] = 1, \qquad \partial_a\varepsilon(2) = x[1] = 0.5,$$
$$\partial_b\varepsilon(1) = -\varepsilon(0) = 0, \qquad \partial_b\varepsilon(2) = -\varepsilon(1) = -0.5.$$

Hence,

$$\boldsymbol{J}_\varepsilon(\boldsymbol{\theta}_0) = \begin{bmatrix} 1 & 0 \\ 0.5 & -0.5 \end{bmatrix}.$$

**Step 3: Gauss–Newton step.** Compute

$$\boldsymbol{J}^\mathsf{T}\boldsymbol{J} = \begin{bmatrix} 1.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix}, \qquad \boldsymbol{J}^\mathsf{T}\boldsymbol{\varepsilon} = \begin{bmatrix} 0.4 \\ 0.1 \end{bmatrix}.$$

Since $\det\big(\boldsymbol{J}^\mathsf{T}\boldsymbol{J}\big) = 0.25$, the inverse is

$$(\boldsymbol{J}^\mathsf{T}\boldsymbol{J})^{-1} = \begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix}.$$

Thus,

$$\Delta\boldsymbol{\theta}_0 = -(\boldsymbol{J}^\mathsf{T}\boldsymbol{J})^{-1}\boldsymbol{J}^\mathsf{T}\boldsymbol{\varepsilon} = -\begin{bmatrix}1 & 1\\ 1 & 5\end{bmatrix}\begin{bmatrix}0.4\\ 0.1\end{bmatrix} = \begin{bmatrix}-0.5\\ -0.9\end{bmatrix},$$

and the updated parameters are

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_0 = \begin{bmatrix}-0.5\\ -0.9\end{bmatrix}.$$

This illustrates the mechanics of Gauss–Newton: compute residuals and Jacobian via forward recursions, then solve a small linear system to update the parameters.

## 4.6 Practical View: Nonlinear LS via an Optimization Lens

Nonlinear least squares is an optimization problem of the form

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{\theta})\|_2^2,$$

where $\boldsymbol{r}(\boldsymbol{\theta}) \in \mathbb{R}^M$ is the residual vector (e.g., stacked ARMA prediction errors). From the perspective of general optimization, Gauss–Newton simply constructs, at each iterate $\boldsymbol{\theta}_i$, a local quadratic model of $J$ based on a first-order linearization of $\boldsymbol{r}$, and then takes a descent step.

**Gradient and (approximate) Hessian.** Using the chain rule,

$$\nabla J(\boldsymbol{\theta}) = \boldsymbol{J}_r(\boldsymbol{\theta})^\mathsf{T}\boldsymbol{r}(\boldsymbol{\theta}), \qquad \boldsymbol{J}_r(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{r}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}.$$

The exact Hessian is

$$\nabla^2 J(\boldsymbol{\theta}) = \boldsymbol{J}_r(\boldsymbol{\theta})^\mathsf{T}\boldsymbol{J}_r(\boldsymbol{\theta}) + \sum_{m=1}^{M} r_m(\boldsymbol{\theta})\,\nabla^2 r_m(\boldsymbol{\theta}),$$

and Gauss–Newton uses the approximation

$$\boldsymbol{H}_{\mathrm{GN}}(\boldsymbol{\theta}) \triangleq \boldsymbol{J}_r(\boldsymbol{\theta})^\mathsf{T}\boldsymbol{J}_r(\boldsymbol{\theta}),$$

which is accurate when residuals are small near the optimum (or when the model is close to correct).

**Step computation as a linear solve.** The Gauss–Newton direction $\Delta\boldsymbol{\theta}_i$ solves

$$\boldsymbol{H}_{\mathrm{GN}}(\boldsymbol{\theta}_i)\,\Delta\boldsymbol{\theta}_i = -\nabla J(\boldsymbol{\theta}_i) = -\boldsymbol{J}_r(\boldsymbol{\theta}_i)^\mathsf{T}\boldsymbol{r}(\boldsymbol{\theta}_i).$$

This is directly analogous to a Newton step, but with an approximate Hessian that avoids second derivatives and exploits the least-squares structure.

**Line search (globalization).** As with general nonlinear optimization, the full step $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i$ may fail when the initial guess is poor or when the quadratic model is not accurate. A standard remedy is *line search*:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_i\,\Delta\boldsymbol{\theta}_i, \qquad \alpha_i \in (0, 1],$$

where $\alpha_i$ is chosen to ensure sufficient decrease in $J$ (e.g., Armijo condition) and to keep the iterates stable. In practice, a backtracking line search is often sufficient: start from $\alpha_i = 1$ and repeatedly reduce $\alpha_i \leftarrow \rho\alpha_i$ (with $\rho \in (0, 1)$) until $J(\boldsymbol{\theta}_{i+1})$ decreases adequately.

**Damping (Levenberg–Marquardt) as trust-region intuition.** Another widely used stabilization is *damping*, leading to the Levenberg–Marquardt (LM) step:

$$\big(\boldsymbol{J}_r(\boldsymbol{\theta}_i)^\mathsf{T}\boldsymbol{J}_r(\boldsymbol{\theta}_i) + \mu_i \boldsymbol{I}\big)\,\Delta\boldsymbol{\theta}_i = -\boldsymbol{J}_r(\boldsymbol{\theta}_i)^\mathsf{T}\boldsymbol{r}(\boldsymbol{\theta}_i), \qquad \mu_i > 0.$$

For large $\mu_i$, the step behaves like gradient descent (small, conservative, robust); for small $\mu_i$, it approaches Gauss–Newton (fast local convergence). This is closely related to a trust-region viewpoint: the algorithm restricts steps when the local model is unreliable and enlarges them when predictions match actual decrease.

**Practical checklist (what you actually implement).** Given a current estimate $\boldsymbol{\theta}_i$:

1. Compute residuals $\boldsymbol{r}(\boldsymbol{\theta}_i)$ (e.g., ARMA prediction errors via forward recursion).
2. Compute Jacobian $\boldsymbol{J}_r(\boldsymbol{\theta}_i)$ (analytically via recursions, or numerically if needed).
3. Compute a search direction by solving either:

   $$\boldsymbol{J}^\mathsf{T}\boldsymbol{J}\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}^\mathsf{T}\boldsymbol{r} \quad \text{(Gauss–Newton)}, \qquad \text{or} \qquad (\boldsymbol{J}^\mathsf{T}\boldsymbol{J} + \mu\boldsymbol{I})\,\Delta\boldsymbol{\theta} = -\boldsymbol{J}^\mathsf{T}\boldsymbol{r} \quad \text{(LM)}.$$

4. Update using a step length $\alpha_i$ (line search) and/or a damping parameter $\mu_i$:

   $$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_i \Delta\boldsymbol{\theta}_i.$$

5. Stop when $J(\boldsymbol{\theta}_i)$, $\|\nabla J(\boldsymbol{\theta}_i)\|$, or $\|\Delta\boldsymbol{\theta}_i\|$ is sufficiently small.

**Last remarks.** From an optimization viewpoint, Gauss–Newton is simply "Newton's method specialized to least squares", with a Hessian approximation that is cheap and often very effective. When combined with standard globalization tools (line search or trust-region/damping), it becomes a robust workhorse for practical parameter estimation problems such as ARMA prediction-error fitting, nonlinear regression, and many identification tasks in state-space models.

# 5 Nonlinear Autoregressive Models and Linear Models on Features

## 5.1 NAR Models

Many real-world signals cannot be accurately captured by linear autoregressive dynamics. Phenomena such as saturation, dead-zones, hysteresis, switching, and state-dependent noise introduce nonlinear dependencies on past values. A natural extension of an AR($p$) model is the *nonlinear autoregressive* (NAR) model

$$x[k] = f\big(x[k-1],\, x[k-2],\, \ldots,\, x[k-p];\boldsymbol{\theta}\big) + w[k],$$

where $f(\cdot;\boldsymbol{\theta})$ is a nonlinear mapping parameterized by an unknown vector $\boldsymbol{\theta}$, and $w[k]$ collects innovation noise and modeling error. The order $p$ determines the memory length, i.e., how many past samples influence the prediction.

From an estimation viewpoint, NAR identification is naturally posed as a (generally) *nonlinear least-squares* problem. For each $k \geq p$ we define the one-step prediction residual

$$\varepsilon(k, \boldsymbol{\theta}) = x[k] - f\big(x[k-1], \ldots, x[k-p]; \boldsymbol{\theta}\big),$$

and estimate the parameters by minimizing the sum of squared residuals:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{k=p}^{N-1} \varepsilon(k, \boldsymbol{\theta})^2.$$

In contrast to linear AR, there is typically no closed-form solution, so we use iterative optimization methods (gradient descent, Gauss–Newton, Levenberg–Marquardt). The practical cost of these methods is dominated by repeatedly (i) evaluating $f(\cdot; \boldsymbol{\theta})$ over the dataset and (ii) computing derivatives of the residuals with respect to $\boldsymbol{\theta}$.

Conceptually, NAR generalizes AR by replacing the linear predictor $-\sum_{i=1}^{p} a_i x[k-i]$ with a nonlinear function of the same past-sample vector.

## 5.2 Linear Models on Features (Basis Expansion)

A useful compromise between linear and nonlinear modeling is to keep the model *linear in the parameters* while allowing nonlinear dependence on the inputs through a feature map. Let $\boldsymbol{x} \in \mathbb{R}^d$ denote an input (or regressor) vector, and define a feature vector

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} \phi_1(\boldsymbol{x}) \\ \phi_2(\boldsymbol{x}) \\ \vdots \\ \phi_P(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^P.$$

A linear-in-features model is

$$y = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}) + w, \qquad \boldsymbol{\theta} \in \mathbb{R}^P.$$

Given data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$, we stack the feature vectors into the design matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathsf{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathsf{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathsf{T}} \end{bmatrix} \in \mathbb{R}^{N \times P}, \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}.$$

Then parameter estimation reduces to ordinary least squares:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2, \qquad \hat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{y},$$

assuming $\boldsymbol{\Phi}$ has full column rank (otherwise use $\boldsymbol{\Phi}^{\dagger}$ or ridge regularization).

This viewpoint is important because it allows nonlinear function approximation while preserving the simplicity of linear estimation tools (LS, ridge/Tikhonov, and their statistical interpretations). The nonlinearity is pushed into the choice of basis functions $\phi_i(\cdot)$, while the unknown parameters still enter linearly.

Common feature choices include polynomial basis expansions (e.g., $[1, x, x^2, \ldots]$), trigonometric/Fourier features (e.g., $\sin(\omega x)$ and $\cos(\omega x)$), radial basis functions, and learned features (e.g., the output of a fixed neural network layer). Model complexity is controlled by the number and type of features and, in practice, by regularization when $P$ is large relative to $N$.

## 5.3 NAR as Linear Regression on Features (Special Case)

An instructive special case is when the NAR function is expressed as a linear combination of chosen nonlinear features of the past-sample vector. Define the regressor

$$\boldsymbol{z}[k] = \begin{bmatrix} x[k-1] \\ x[k-2] \\ \vdots \\ x[k-p] \end{bmatrix},$$

choose a feature map $\boldsymbol{\phi}(\boldsymbol{z}[k])$, and model

$$x[k] = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{z}[k]) + w[k].$$

This is a nonlinear autoregressive model in terms of input–output behavior, but it remains *linear in the unknown parameters* $\boldsymbol{\theta}$, so we can estimate it by (regularized) least squares. This perspective is often called *basis-function NAR* and provides a practical bridge between linear AR estimation and fully nonlinear NAR identification.

## 5.4 Worked Examples: Linear Models on Features

**Example 1: Affine model.** Consider again the simple feature map

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \end{bmatrix}, \qquad y = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\phi}(x) + w, \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}.$$

Suppose we observe the following three samples:

$$(x_1, y_1) = (0, 0), \quad (x_2, y_2) = (1, 1), \quad (x_3, y_3) = (2, 3).$$

The design matrix and output vector are

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}.$$

Compute

$$\boldsymbol{\Phi}^{\mathsf{T}} \boldsymbol{\Phi} = \begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix}, \qquad \boldsymbol{\Phi}^{\mathsf{T}} \boldsymbol{y} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}.$$

Thus,

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^{\mathsf{T}} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^{\mathsf{T}} \boldsymbol{y} = \frac{1}{6} \begin{bmatrix} 5 & -3 \\ -3 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 7 \end{bmatrix} = \begin{bmatrix} -1/6 \\ 7/6 \end{bmatrix}.$$

The fitted model is therefore

$$\hat{y}(x) = -\frac{1}{6} + \frac{7}{6} x.$$

**Example 2: Quadratic feature expansion.** We now increase model flexibility by using a quadratic feature map:

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}, \qquad y = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\phi}(x) + w, \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}.$$

Consider four samples:

$$(x_1, y_1) = (-1, 1), \quad (x_2, y_2) = (0, 0), \quad (x_3, y_3) = (1, 1), \quad (x_4, y_4) = (2, 4).$$

The design matrix and output vector are

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 4 \end{bmatrix}.$$

Compute

$$\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} = \begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 8 \\ 6 & 8 & 18 \end{bmatrix}, \qquad \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y} = \begin{bmatrix} 6 \\ 8 \\ 18 \end{bmatrix}.$$

Solving $(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi})\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}$ gives

$$\hat{\boldsymbol{\theta}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Hence, the fitted model is

$$\hat{y}(x) = x^2.$$

**Takeaway.** By changing the feature map $\boldsymbol{\phi}(x)$, we can move from simple linear models to more expressive nonlinear predictors, while still relying on exactly the same least-squares machinery. Model complexity is determined entirely by the chosen features (and, in practice, by regularization when the feature dimension grows).

## 5.5 Python Implemetation

The following script demonstrates the "linear model on features" idea on a toy nonlinear regression problem. We generate a small set of noisy samples from a nonlinear ground-truth function

$$y_{\text{true}}(x) = 0.6\sin(2\pi x) + 0.3x^3 - 0.2x,$$

and then fit it using a model that is *linear in the parameters* but *nonlinear in the input* via a feature map

$$\boldsymbol{\phi}(x) = \left[1,\ x,\ x^2,\ x^3,\ \sin(2\pi x),\ \cos(2\pi x)\right]^\mathsf{T}.$$

Stacking these features over all samples yields a design matrix $\boldsymbol{\Phi}$ and the linear regression model

$$\boldsymbol{y} \approx \boldsymbol{\Phi}\boldsymbol{\theta}.$$

Two estimators are compared:

- **Least Squares (LS):** $\hat{\boldsymbol{\theta}}_{\text{LS}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2$ (implemented with `np.linalg.lstsq`).
- **Ridge (regularized LS):** $\hat{\boldsymbol{\theta}}_{\text{ridge}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2$, which has the closed form

$$\hat{\boldsymbol{\theta}}_{\text{ridge}} = (\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}.$$

Figure 2 plots the noisy samples, the true function, and the fitted curves from LS and ridge. Figure 3 visualizes the learned weights of each feature; ridge typically shrinks coefficients and reduces sensitivity to noise. For reference, the script also constructs the "true" coefficient vector (matching the data-generating function in this chosen basis) and compares it to the estimated weights.

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

# ------------------------------
# 1) Generate nonlinear data
# ------------------------------
rng = np.random.default_rng(0)

N = 10
x = np.linspace(-1, 1, N)
y_true = 0.6*np.sin(2*np.pi*x) + 0.3*x**3 - 0.2*x
y = y_true + 0.3*rng.standard_normal(N)  # noisy observations

# Dense grid for smooth plotting
x_plot = np.linspace(-1, 1, 500)
y_true_plot = 0.6*np.sin(2*np.pi*x_plot) + 0.3*x_plot**3 - 0.2*x_plot

# ------------------------------
# 2) Feature map phi(x)
#    (polynomials + sin/cos)
# ------------------------------
def phi(x):
    """
    Feature vector:
    [1, x, x^2, x^3, sin(2πx), cos(2πx)]
    """
    x = np.asarray(x)
    return np.column_stack([
        np.ones_like(x),
        x,
        x**2,
        x**3,
        np.sin(2*np.pi*x),
        np.cos(2*np.pi*x),
    ])

Phi = phi(x)
Phi_plot = phi(x_plot)

# ------------------------------
# 3) Least Squares fit
#    y ≈ Phi theta
# ------------------------------
theta_ls, *_ = np.linalg.lstsq(Phi, y, rcond=None)
y_ls_plot = Phi_plot @ theta_ls

# ------------------------------
# 4) Ridge fit (regularized LS)
#    min ||y-Phi theta||^2 + λ||theta||^2
# ------------------------------
lam = 1.
I = np.eye(Phi.shape[1])
```

```
theta_ridge = np.linalg.solve(Phi.T @ Phi + lam*I, Phi.T @ y)
y_ridge_plot = Phi_plot @ theta_ridge

# ------------------------------
# 5) Plot results
# ------------------------------
plt.figure(figsize=(9, 4))

plt.scatter(x, y, label="Noisy samples", marker="o")
plt.plot(x_plot, y_true_plot, linestyle="--", label="True function")
plt.plot(x_plot, y_ls_plot, label="LS on features")
plt.plot(x_plot, y_ridge_plot, label=f"Ridge on features (λ={lam})")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Model on Nonlinear Features")
plt.legend()
plt.tight_layout()
plt.show()

# ------------------------------
# 6) Plot feature weights
# ------------------------------
feature_names = ["1", "x", "x^2", "x^3", "sin(2πx)", "cos(2πx)"]
theta_true = np.zeros_like(theta_ridge)
theta_true[1] = -0.2
theta_true[3] = 0.3
theta_true[4] = 0.6

plt.figure(figsize=(7, 3))
idx = np.arange(len(feature_names))
plt.stem(idx, theta_ls, linefmt='b--', markerfmt='bo', basefmt=" ",
    label="LS")
plt.stem(idx, theta_ridge, linefmt="g--", markerfmt="gD", basefmt=" ",
    label="Ridge")
plt.stem(idx, theta_true, linefmt="k--", markerfmt="kD", basefmt=" ",
    label="True")

plt.xticks(idx, feature_names)
plt.xlabel("Feature")
plt.ylabel("Weight")
plt.title("Learned Feature Weights")
plt.legend()
plt.tight_layout()
plt.show()

print("theta_ls   =", theta_ls)
print("theta_ridge=", theta_ridge)
```

Listing 1: Linear in Features Regression Example

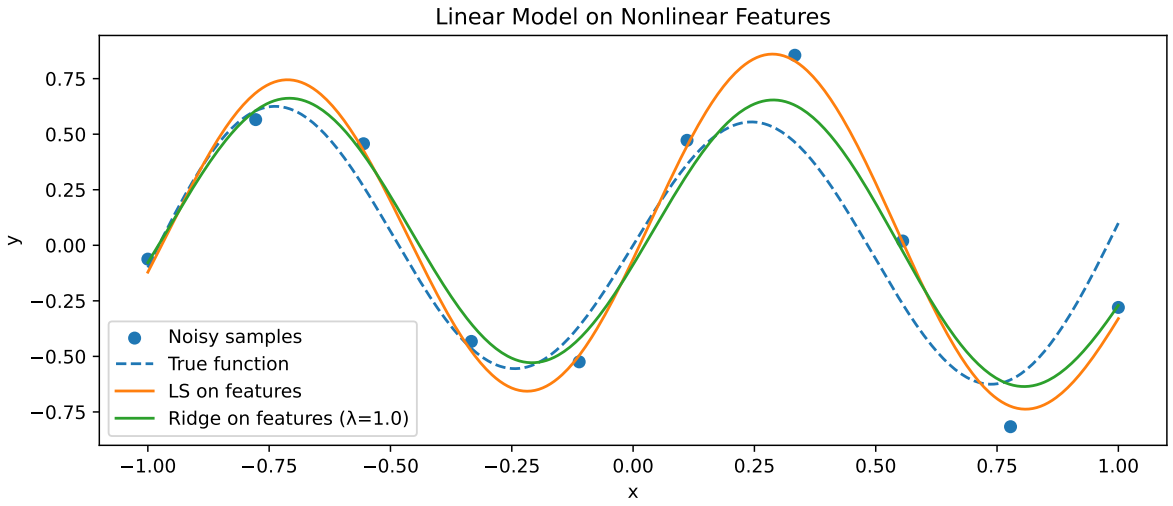



Figure 2: Noisy samples, true function, LS fit on features, and ridge-regularized fit.

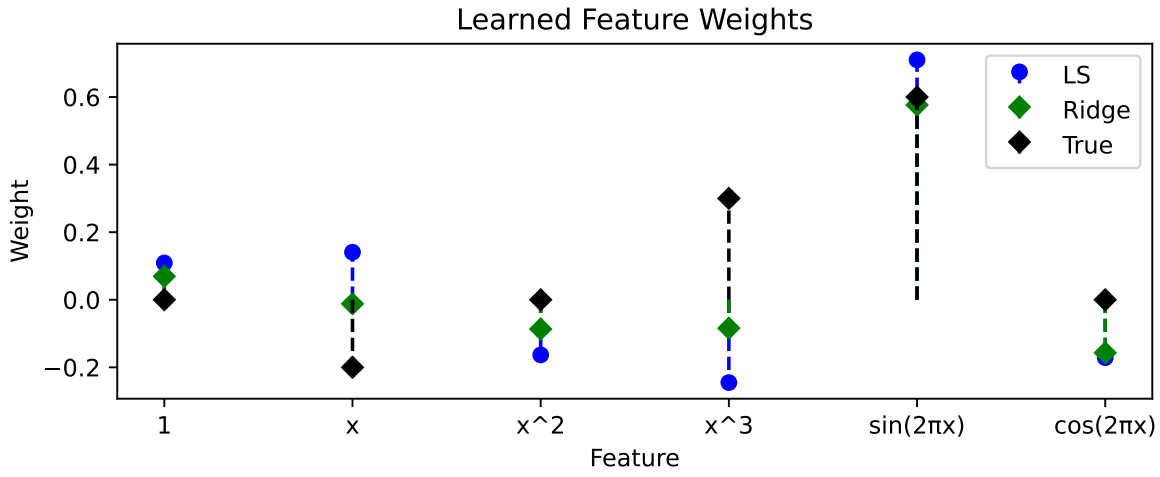



Figure 3: Estimated feature weights for LS vs. ridge.

# 6 Bayesian Inference

## 6.1 Core Idea

In Bayesian inference, unknown model parameters are treated as *random variables* rather than fixed but unknown constants. Instead of seeking a single best estimate of $\boldsymbol{\theta}$, we represent our uncertainty about its value using a probability distribution. This uncertainty before seeing any data is encoded through a *prior distribution* $p(\boldsymbol{\theta})$, which reflects prior knowledge, physical constraints, or modeling assumptions.

The observed data are also modeled probabilistically. Given inputs $\boldsymbol{X}$ and parameters $\boldsymbol{\theta}$, the outputs $\boldsymbol{Y}$ are assumed to be generated according to a stochastic data model. This explicitly accounts for measurement noise, unmodeled dynamics, and randomness in the data-generating process.

The central goal of Bayesian inference is therefore not to compute a single parameter vector, but to infer a *distribution over parameters* conditioned on the observed data. This distribution quantifies both what parameter values are plausible and how uncertain we are about them. Bayesian methods are particularly attractive because they provide principled uncertainty estimates, incorporate regularization naturally through priors, and enable systematic comparison between competing models.

## 6.2 Likelihood

The likelihood

$$p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})$$

describes how the data are generated given the parameters. It is a probabilistic model of the observations conditioned on $\boldsymbol{\theta}$ (and possibly inputs $\boldsymbol{X}$).

The likelihood can be interpreted in two complementary ways.

1. **For fixed parameters $\boldsymbol{\theta}$**, it is a probability distribution over possible datasets, describing how likely different observations are under the assumed model.

2. **For fixed observed data $(\boldsymbol{X}, \boldsymbol{Y})$**, it becomes a function of $\boldsymbol{\theta}$ that measures how compatible different parameter values are with the data.

The choice of likelihood depends on the assumed noise model and data-generating process.

A common example is additive Gaussian noise:

$$y_i = f(x_i; \boldsymbol{\theta}) + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2).$$

Under the assumption that noise samples are independent, the likelihood factorizes as

$$p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}\big(y_i \,;\, f(x_i; \boldsymbol{\theta}), \sigma^2\big)\,.$$

This choice directly connects Bayesian inference with least-squares estimation, since maximizing the likelihood under Gaussian noise is equivalent to minimizing a sum of squared residuals.

## 6.3 Bayes' Rule and the Posterior

Bayes' rule combines prior knowledge and data through

$$p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}) = \frac{p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{Y} \mid \boldsymbol{X})}.$$

The resulting distribution $p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X})$ is the *posterior*, representing our updated belief about the parameters after observing the data.

The denominator

$$p(\boldsymbol{Y} \mid \boldsymbol{X}) = \int p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}$$

is called the *evidence* or *marginal likelihood*. It ensures that the posterior integrates to one and plays a key role in model comparison. Importantly, it does not depend on $\boldsymbol{\theta}$ when performing parameter inference.

A useful summary is

$$\text{posterior} \;\propto\; \text{likelihood} \times \text{prior}.$$

The likelihood pulls the posterior toward parameter values that explain the data well, while the prior pulls it toward values considered plausible before seeing the data. The posterior represents a balance between these two sources of information.

## 6.4 Interpretation of the Bayesian Quantities

Each component of Bayes' rule has a clear interpretation. The prior $p(\boldsymbol{\theta})$ encodes assumptions or domain knowledge available before observing data. The likelihood $p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})$ measures how well a given parameter setting explains the observed data. The posterior $p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X})$ is the updated belief that combines both. Finally, the evidence $p(\boldsymbol{Y} \mid \boldsymbol{X})$ quantifies how plausible the observed data are under the assumed model class and prior, and is therefore useful for comparing different models.

From an estimation perspective, Bayesian inference generalizes the methods studied so far: least squares and regularization appear as special cases corresponding to particular likelihood–prior combinations, while Bayesian methods additionally provide principled uncertainty quantification and a unified probabilistic interpretation.

**Evidence and Model Comparison** Beyond parameter estimation, Bayesian inference also provides a principled mechanism for comparing different models. This is done through the *evidence*, or *marginal likelihood*,

$$p(\boldsymbol{Y} \mid \boldsymbol{X}) = \int p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}.$$

Unlike the likelihood, which evaluates how well a specific parameter value explains the data, the evidence measures how well an entire model class explains the data *before* knowing the parameters.

The evidence can be interpreted as an average of the likelihood over the prior distribution. Models that assign high likelihood to the observed data for many plausible parameter values achieve high evidence, whereas models that require fine-tuned parameter choices to explain the data are penalized. In this sense, the evidence naturally balances data fit and model complexity.

This quantity enables Bayesian *model comparison*. Given two competing models $\mathcal{M}_1$ and $\mathcal{M}_2$, their relative support from the data is given by the *Bayes factor*:

$$\frac{p(\boldsymbol{Y} \mid \boldsymbol{X}, \mathcal{M}_1)}{p(\boldsymbol{Y} \mid \boldsymbol{X}, \mathcal{M}_2)}.$$

If this ratio is greater than one, the data favor $\mathcal{M}_1$ over $\mathcal{M}_2$; if it is less than one, $\mathcal{M}_2$ is preferred.

A key conceptual consequence is the built-in *Occam's razor* effect of Bayesian inference. More complex models typically spread probability mass over a larger parameter space and therefore tend to have lower evidence unless the data provide strong support for that added complexity. As a result, Bayesian model comparison automatically trades off goodness of fit against model complexity, without the need for ad hoc penalties or cross-validation heuristics.

## 6.5 Bayesian Point Estimation via Risk Minimization

Bayesian inference yields a full posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X})$, which quantifies uncertainty about the unknown parameters. In many applications, however, we still want to report a single parameter vector $\hat{\boldsymbol{\theta}}$ for prediction, control, or interpretation. Bayesian point estimators are best understood through a decision-theoretic viewpoint: we choose an estimate that minimizes *posterior expected loss* (posterior risk). Given a loss function $\mathcal{L}(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}})$ that penalizes estimating $\tilde{\boldsymbol{\theta}}$ when the (latent) parameter is $\boldsymbol{\theta}$, the optimal Bayesian decision rule is

$$\hat{\boldsymbol{\theta}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\Big[\mathcal{L}(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \;\Big|\; \boldsymbol{Y}, \boldsymbol{X}\Big] = \arg\min_{\tilde{\boldsymbol{\theta}}} \int \mathcal{L}(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}})\, p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X})\, d\boldsymbol{\theta}.$$

Different loss functions correspond to different notions of "best" estimate, and therefore lead to different point estimators.

### 6.5.1 MAP Estimator (Posterior Mode)

The maximum a posteriori (MAP) estimator selects the parameter value at which the posterior density is maximized:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}).$$

Using Bayes' rule,

$$p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}) \propto p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta}),$$

so equivalently

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta}).$$

In optimization form, MAP is obtained by minimizing the negative log-posterior:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \Big( -\log p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) \Big).$$

Thus, MAP estimation can be interpreted as minimizing a *data misfit term* (from the likelihood) plus a *regularization term* (from the prior). Because MAP chooses the posterior *mode*, it selects the single most probable parameter value; if the posterior is multimodal, there may be multiple MAP solutions (multiple local maxima).

**MAP, Priors, and Regularization** The connection between priors and regularization becomes explicit in common likelihood–prior combinations. For example, a zero-mean isotropic Gaussian prior

$$p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\boldsymbol{I})$$

implies

$$-\log p(\boldsymbol{\theta}) = \text{const} + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2,$$

i.e., an $\ell_2$ penalty on the parameters. If the likelihood corresponds to Gaussian observation noise,

$$y_i = f(x_i; \boldsymbol{\theta}) + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

then

$$-\log p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \text{const} + \frac{1}{2\sigma^2}\|\boldsymbol{Y} - f(\boldsymbol{X}; \boldsymbol{\theta})\|_2^2.$$

Therefore MAP becomes a regularized least-squares problem:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma^2}\|\boldsymbol{Y} - f(\boldsymbol{X}; \boldsymbol{\theta})\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2.$$

This provides a precise probabilistic interpretation of ridge (Tikhonov) regularization: it corresponds to a Gaussian prior on the parameters.

**MLE as a Special Case of MAP** If the prior is flat (uninformative) over the parameter region of interest,

$$p(\boldsymbol{\theta}) \propto \text{const},$$

then the MAP estimator reduces to the maximum likelihood estimator (MLE):

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \hat{\boldsymbol{\theta}}_{\text{MLE}}.$$

In words, MAP uses both prior information and data, whereas MLE uses data only. As the amount of data increases, the influence of the prior typically diminishes and MAP and MLE often become close.

### 6.5.2 MMSE Estimator (Posterior Mean)

The minimum mean square error (MMSE) estimator arises from squared-error loss:

$$\hat{\boldsymbol{\theta}}_{\text{MMSE}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\Big[\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 \;\Big|\; \boldsymbol{Y}, \boldsymbol{X}\Big].$$

The unique minimizer is the posterior mean:

$$\hat{\boldsymbol{\theta}}_{\text{MMSE}} = \mathbb{E}[\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}] = \int \boldsymbol{\theta}\, p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X})\, d\boldsymbol{\theta}.$$

Intuitively, MMSE averages all plausible parameter values, weighted by their posterior probability. This can be more robust than MAP when the posterior is skewed or multimodal, because the mean reflects the full mass of the distribution rather than only its peak.

**Derivation** Let $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}]$. Consider the posterior risk for a candidate estimate $\tilde{\boldsymbol{\theta}}$:

$$\mathbb{E}\Big[\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 \mid \boldsymbol{Y}, \boldsymbol{X}\Big].$$

Add and subtract $\boldsymbol{\mu}$:

$$\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 = \|(\boldsymbol{\theta} - \boldsymbol{\mu}) + (\boldsymbol{\mu} - \tilde{\boldsymbol{\theta}})\|_2^2 = \|\boldsymbol{\theta} - \boldsymbol{\mu}\|_2^2 + \|\boldsymbol{\mu} - \tilde{\boldsymbol{\theta}}\|_2^2 + 2(\boldsymbol{\theta} - \boldsymbol{\mu})^{\mathsf{T}}(\boldsymbol{\mu} - \tilde{\boldsymbol{\theta}}).$$

Taking the posterior expectation makes the cross-term vanish because $\mathbb{E}[\boldsymbol{\theta} - \boldsymbol{\mu} \mid \boldsymbol{Y}, \boldsymbol{X}] = \mathbf{0}$. Hence

$$\mathbb{E}\Big[\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 \mid \cdot\Big] = \underbrace{\mathbb{E}\big[\|\boldsymbol{\theta} - \boldsymbol{\mu}\|_2^2 \mid \cdot\big]}_{\text{independent of } \tilde{\boldsymbol{\theta}}} + \|\boldsymbol{\mu} - \tilde{\boldsymbol{\theta}}\|_2^2,$$

which is minimized when $\tilde{\boldsymbol{\theta}} = \boldsymbol{\mu}$.

### 6.5.3 MMAE Estimator (Posterior Median)

If we instead use absolute-error loss (componentwise $\ell_1$ loss),

$$\hat{\boldsymbol{\theta}}_{\text{MMAE}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\Big[\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_1 \;\Big|\; \boldsymbol{Y}, \boldsymbol{X}\Big], \qquad \|\boldsymbol{v}\|_1 = \sum_i |v_i|,$$

then the optimal estimator is the posterior median. In practice, a common choice is the *componentwise posterior median*:

$$\hat{\boldsymbol{\theta}}_{\text{MMAE}} = \big[\text{median}(p(\theta_1 \mid \boldsymbol{Y}, \boldsymbol{X})), \ldots, \text{median}(p(\theta_P \mid \boldsymbol{Y}, \boldsymbol{X}))\big]^{\mathsf{T}}.$$

MMAE is often more robust to heavy-tailed posteriors and outliers than MMSE, because medians are less influenced by extreme values than means.

### 6.5.4 Summary: MAP vs. MMSE vs. MMAE

The three most common Bayesian point estimators correspond to different summaries of the posterior:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}) \quad \text{(mode)}, \qquad \hat{\boldsymbol{\theta}}_{\text{MMSE}} = \mathbb{E}[\boldsymbol{\theta} \mid \boldsymbol{Y}, \boldsymbol{X}] \quad \text{(mean)},$$

$$\hat{\boldsymbol{\theta}}_{\text{MMAE}} = \arg\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\Big[\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_1 \mid \boldsymbol{Y}, \boldsymbol{X}\Big] \quad \text{(median)}.$$

If the posterior is Gaussian (or, more generally, symmetric and unimodal), then

$$\text{mean} = \text{median} = \text{mode} \quad \Rightarrow \quad \hat{\boldsymbol{\theta}}_{\text{MMSE}} = \hat{\boldsymbol{\theta}}_{\text{MMAE}} = \hat{\boldsymbol{\theta}}_{\text{MAP}}.$$

For skewed, heavy-tailed, or multimodal posteriors, these estimators can differ significantly, and the appropriate choice depends on the loss function that best matches the application's notion of error.

**1D Geometric Intuition** Figure 4 illustrates the geometric intuition behind MAP, MMSE, and MMAE in one dimension. The MAP estimate corresponds to the peak of the posterior density (mode), the MMSE estimate corresponds to the center of mass (mean), and the MMAE estimate corresponds to the point that divides the posterior into two equal probability halves (median). The key result is that different estimators reflect different optimality criteria (different losses).

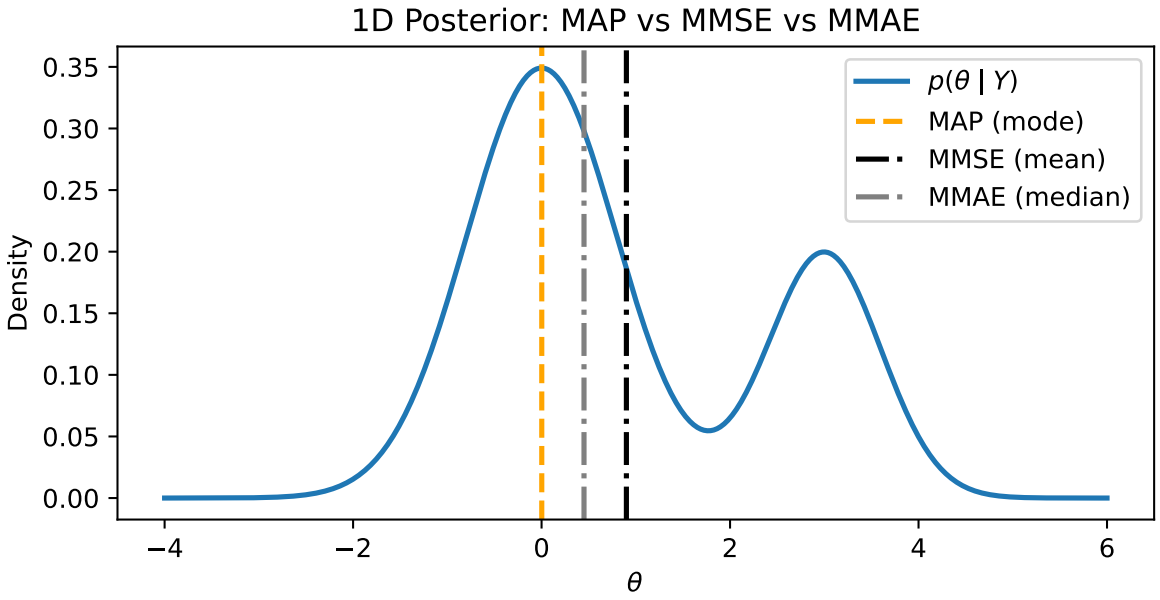



Figure 4: Geometric intuition for Bayesian point estimators in one dimension. For a posterior $p(\theta \mid Y)$: the MAP estimate $\hat{\theta}_{\text{MAP}}$ is the posterior *mode* (peak), the MMSE estimate $\hat{\theta}_{\text{MMSE}}$ is the posterior *mean* (center of mass), and the MMAE estimate $\hat{\theta}_{\text{MMAE}}$ is the posterior *median* (50% probability mass on each side). Different estimators correspond to different loss functions and can differ substantially for skewed or multimodal posteriors.

## 6.6 Maximum Likelihood Estimation (MLE)

Maximum likelihood estimation is a frequentist approach in which the unknown parameter vector $\boldsymbol{\theta}$ is treated as a fixed (but unknown) quantity. We assume an observation model that specifies a likelihood

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}),$$

i.e., the probability (density) of observing the data $\boldsymbol{y}$ under a given parameter value.

The MLE selects the parameter value that makes the observed data most probable:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{y} \mid \boldsymbol{\theta}).$$

In practice we almost always work with the log-likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = \log p(\boldsymbol{y} \mid \boldsymbol{\theta}), \qquad \hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}),$$

because the logarithm turns products into sums and improves numerical stability without changing the maximizer.

**i.i.d. data.** If the samples are independent and identically distributed (i.i.d.), then the likelihood factorizes:

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \prod_{k=1}^{N} p(y_k \mid \boldsymbol{\theta}),$$

so the log-likelihood becomes a sum:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^{N} \log p(y_k \mid \boldsymbol{\theta}).$$

More generally, for i.i.d. input–output pairs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ with a conditional model,

$$p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}), \qquad \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}),$$

and MLE is

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}).$$

The crucial point is that MLE depends entirely on the assumed data/noise model through $p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$: changing the distributional assumption changes the estimator.

**How is the MLE computed?** If the log-likelihood is concave in $\boldsymbol{\theta}$ and has a simple form, the maximizer may be available in closed form (or via a single linear solve). Otherwise, we solve an optimization problem numerically, typically using the tools already seen in optimization: gradient ascent/descent on $-\mathcal{L}$, Newton's method, Gauss–Newton (when the objective has least-squares structure), and line-search or trust-region strategies for robustness.

**Example: linear model with Gaussian noise ⇒ least squares.** Consider the linear regression model

$$y_i = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}_i + w_i, \qquad \boldsymbol{\theta} \in \mathbb{R}^p,$$

with i.i.d. Gaussian noise $w_i \sim \mathcal{N}(0, \sigma^2)$. Then

$$p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) = \mathcal{N}\big(y_i; \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}_i, \sigma^2\big)\,,$$

and the log-likelihood is

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left[ -\frac{1}{2} \log\big(2\pi\sigma^2\big) - \frac{1}{2\sigma^2} \big(y_i - \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}_i\big)^2 \right].$$

Maximizing $\mathcal{L}(\boldsymbol{\theta})$ is equivalent to minimizing the sum of squared residuals:

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \iff \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \big(y_i - \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}_i\big)^2.$$

Therefore, under a linear model with i.i.d. Gaussian noise, MLE coincides exactly with ordinary linear least squares. This provides a probabilistic justification for the LS criterion and clarifies how the assumed noise distribution determines the estimation objective.

### 6.6.1 Worked Example: MLE for a Gaussian Mean (Known Variance)

Assume we observe i.i.d. samples

$$y_1, \ldots, y_N \quad \text{with} \quad y_k \sim \mathcal{N}(\mu, \sigma^2),$$

where the variance $\sigma^2$ is known and the unknown parameter is the mean $\theta = \mu$.

**Likelihood.** Because the samples are i.i.d.,

$$p(\boldsymbol{y} \mid \mu) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_k - \mu)^2}{2\sigma^2}\right).$$

**Log-likelihood.** Taking the logarithm,

$$\mathcal{L}(\mu) = \log p(\boldsymbol{y} \mid \mu) = -\frac{N}{2}\log\big(2\pi\sigma^2\big) - \frac{1}{2\sigma^2}\sum_{k=1}^{N}(y_k - \mu)^2.$$

Since the first term is constant in $\mu$, maximizing $\mathcal{L}(\mu)$ is equivalent to minimizing $\sum_{k=1}^{N}(y_k - \mu)^2$.

**Compute the maximizer.** Differentiate $\mathcal{L}(\mu)$ (or the equivalent sum-of-squares objective) and set to zero:

$$\frac{d\mathcal{L}}{d\mu} = -\frac{1}{2\sigma^2}\frac{d}{d\mu}\sum_{k=1}^{N}(y_k - \mu)^2 = -\frac{1}{2\sigma^2}\sum_{k=1}^{N}2(y_k - \mu)(-1) = \frac{1}{\sigma^2}\sum_{k=1}^{N}(y_k - \mu).$$

Set $\frac{d\mathcal{L}}{d\mu} = 0$:

$$\sum_{k=1}^{N}(y_k - \mu) = 0 \quad \Rightarrow \quad N\mu = \sum_{k=1}^{N} y_k \quad \Rightarrow \quad \boxed{\hat{\mu}_{\mathrm{MLE}} = \frac{1}{N}\sum_{k=1}^{N} y_k}.$$

Thus, the MLE of the Gaussian mean (with known variance) is the sample average.

**Numeric mini-example.** Let

$$y_1 = 1.2, \quad y_2 = 0.7, \quad y_3 = 1.5, \quad y_4 = 0.6, \quad y_5 = 1.0, \qquad (N = 5).$$

Then

$$\hat{\mu}_{\mathrm{MLE}} = \frac{1}{5}(1.2 + 0.7 + 1.5 + 0.6 + 1.0) = \frac{1}{5}\cdot 5.0 = 1.0.$$

**Connection to least squares.** In this example, maximizing the Gaussian likelihood is equivalent to minimizing $\sum_{k=1}^{N}(y_k - \mu)^2$, i.e., fitting a constant model to data in the least-squares sense. This mirrors the general fact that Gaussian-noise MLE leads to squared-error objectives.

### 6.6.2 Worked Example: MLE for Linear Regression with Unknown Variance

Assume the standard linear model with Gaussian noise:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{w}, \qquad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I}),$$

where both the regression coefficients $\boldsymbol{\theta} \in \mathbb{R}^p$ and the noise variance $\sigma^2 > 0$ are unknown.

**Likelihood.** The conditional density of $\boldsymbol{y}$ given $(\boldsymbol{\theta}, \sigma^2)$ is multivariate Gaussian:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2\right).$$

**Log-likelihood.** Taking the log:

$$\mathcal{L}(\boldsymbol{\theta}, \sigma^2) = -\frac{N}{2}\log\bigl(2\pi\sigma^2\bigr) - \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2.$$

**MLE for $\boldsymbol{\theta}$.** For fixed $\sigma^2$, maximizing $\mathcal{L}$ w.r.t. $\boldsymbol{\theta}$ is equivalent to minimizing $\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\|_2^2$, so the MLE for $\boldsymbol{\theta}$ is the ordinary least-squares solution:

$$\boxed{\hat{\boldsymbol{\theta}}_{\text{MLE}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}} \quad \text{(assuming } \boldsymbol{X} \text{ has full column rank).}$$

**MLE for $\sigma^2$.** Plug $\hat{\boldsymbol{\theta}}_{\text{MLE}}$ into the log-likelihood and maximize w.r.t. $\sigma^2$. Differentiate $\mathcal{L}$ w.r.t. $\sigma^2$:

$$\frac{\partial\mathcal{L}}{\partial\sigma^2} = -\frac{N}{2}\frac{1}{\sigma^2} + \frac{1}{2}\frac{1}{(\sigma^2)^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2.$$

Setting this to zero gives

$$-N\sigma^2 + \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 = 0 \quad \Rightarrow \quad \sigma^2 = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2.$$

Therefore,

$$\boxed{\hat{\sigma}^2_{\text{MLE}} = \frac{1}{N}\bigl\|\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\theta}}_{\text{MLE}}\bigr\|_2^2}$$

(note: the unbiased variance estimator uses $\frac{1}{N-p}$ instead of $\frac{1}{N}$).

**Numeric mini-example (small dataset).** Fit a line $y = \theta_0 + \theta_1 x + w$ from three samples:

$$(x_1, y_1) = (0, 1), \quad (x_2, y_2) = (1, 2), \quad (x_3, y_3) = (2, 2).$$

Then

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}.$$

Compute

$$\boldsymbol{X}^\top\boldsymbol{X} = \begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix}, \qquad \boldsymbol{X}^\top\boldsymbol{y} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}, \qquad (\boldsymbol{X}^\top\boldsymbol{X})^{-1} = \frac{1}{6}\begin{bmatrix} 5 & -3 \\ -3 & 3 \end{bmatrix}.$$

Thus

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \frac{1}{6}\begin{bmatrix} 5 & -3 \\ -3 & 3 \end{bmatrix}\begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 7/6 \\ 1/2 \end{bmatrix}, \quad \Rightarrow \quad \hat{y}(x) = \frac{7}{6} + \frac{1}{2}x.$$

Residuals:

$$\hat{\boldsymbol{r}} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\theta}}_{\text{MLE}} = \begin{bmatrix} 1 - 7/6 \\ 2 - (7/6 + 1/2) \\ 2 - (7/6 + 1) \end{bmatrix} = \begin{bmatrix} -1/6 \\ 1/3 \\ -1/6 \end{bmatrix}, \qquad \|\hat{\boldsymbol{r}}\|_2^2 = \frac{1}{36} + \frac{1}{9} + \frac{1}{36} = \frac{1}{6}.$$

Therefore, with $N = 3$,

$$\hat{\sigma}^2_{\text{MLE}} = \frac{1}{N}\|\hat{\boldsymbol{r}}\|_2^2 = \frac{1}{3}\cdot\frac{1}{6} = \frac{1}{18}.$$

# 7 Bayesian Linear Regression

## 7.1 Model, Notation, and Goals

Bayesian linear regression (BLR) combines a linear-in-parameters model with a probabilistic description of uncertainty in the parameters. We observe training data

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

choose a feature map

$$\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^P, \qquad \boldsymbol{\phi}_i \equiv \boldsymbol{\phi}(\boldsymbol{x}_i),$$

and assume the observation model

$$y_i = \boldsymbol{\phi}_i^{\mathsf{T}} \boldsymbol{\theta} + w_i, \qquad w_i \sim \mathcal{N}(0, \sigma_w^2), \quad \text{i.i.d.}$$

Stacking the observations yields

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \qquad \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1^{\mathsf{T}} \\ \vdots \\ \boldsymbol{\phi}_N^{\mathsf{T}} \end{bmatrix} \in \mathbb{R}^{N \times P}, \qquad \boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\theta} + \boldsymbol{w}, \quad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma_w^2 \boldsymbol{I}).$$

The Bayesian goal is twofold:

- infer a posterior distribution over weights, $p(\boldsymbol{\theta} \mid \mathcal{D})$,
- predict at a new input $\boldsymbol{x}_*$ with uncertainty: $p(y_* \mid \boldsymbol{x}_*, \mathcal{D})$.

## 7.2 Prior and Likelihood

**Prior on weights (conjugate Gaussian).** We encode uncertainty and prior knowledge about the weights by a Gaussian prior

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0).$$

Here $\boldsymbol{\mu}_0$ is the prior mean (our best guess of the weights before data) and $\boldsymbol{\Sigma}_0$ is the prior covariance (uncertainty and correlations). A common special case is an isotropic prior $\boldsymbol{\Sigma}_0 = \alpha^{-1}\boldsymbol{I}$, which shrinks weights toward $\boldsymbol{\mu}_0$ (often $\boldsymbol{0}$).

**Likelihood.** Conditioned on $\boldsymbol{\theta}$, the outputs are Gaussian:

$$p(\boldsymbol{y} \mid \boldsymbol{\Phi}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma_w^2 \boldsymbol{I}).$$

Equivalently, the log-likelihood is

$$\log p(\boldsymbol{y} \mid \boldsymbol{\Phi}, \boldsymbol{\theta}) = -\frac{N}{2} \log\bigl(2\pi\sigma_w^2\bigr) - \frac{1}{2\sigma_w^2} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2.$$

Thus, Gaussian noise connects directly to least squares (MLE $\Leftrightarrow$ LS).

## 7.3 Posterior over Weights

Bayes' rule gives

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) \propto p(\boldsymbol{y} \mid \boldsymbol{\Phi}, \boldsymbol{\theta})\, p(\boldsymbol{\theta}).$$

We derive the posterior explicitly and show it is Gaussian (conjugacy).

**Step 1: Write the unnormalized log posterior.** Ignoring constants independent of $\boldsymbol{\theta}$,

$$\begin{aligned}\log p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) &= \log p(\boldsymbol{y} \mid \boldsymbol{\Phi}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const} \\ &= -\frac{1}{2\sigma_w^2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^\mathsf{T}\boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_0) + \text{const}.\end{aligned}$$

**Step 2: Expand both quadratic forms.** First,

$$\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 = (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) = \boldsymbol{y}^\mathsf{T}\boldsymbol{y} - 2\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\boldsymbol{\theta}.$$

Second,

$$(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^\mathsf{T}\boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_0) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \boldsymbol{\mu}_0^\mathsf{T}\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0.$$

Substitute into the log posterior and collect terms in $\boldsymbol{\theta}$. All terms that do not depend on $\boldsymbol{\theta}$ are absorbed into "const". We obtain

$$\log p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) = -\frac{1}{2}\left[\boldsymbol{\theta}^\mathsf{T}\left(\boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\right)\boldsymbol{\theta} - 2\boldsymbol{\theta}^\mathsf{T}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}\right)\right] + \text{const}.$$

**Step 3: Identify the Gaussian by completing the square.** Define the *posterior precision* matrix and the corresponding linear term:

$$\boldsymbol{\Lambda}_N \triangleq \boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}, \qquad \boldsymbol{\eta}_N \triangleq \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}.$$

Then the log posterior becomes

$$\log p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) = -\frac{1}{2}\left(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Lambda}_N\boldsymbol{\theta} - 2\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\eta}_N\right) + \text{const}.$$

Complete the square:

$$\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\Lambda}_N\boldsymbol{\theta} - 2\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\eta}_N = (\boldsymbol{\theta} - \boldsymbol{\mu}_N)^\mathsf{T}\boldsymbol{\Lambda}_N(\boldsymbol{\theta} - \boldsymbol{\mu}_N) - \boldsymbol{\mu}_N^\mathsf{T}\boldsymbol{\Lambda}_N\boldsymbol{\mu}_N,$$

where the shift $\boldsymbol{\mu}_N$ must satisfy

$$\boldsymbol{\Lambda}_N\boldsymbol{\mu}_N = \boldsymbol{\eta}_N \quad \Rightarrow \quad \boldsymbol{\mu}_N = \boldsymbol{\Lambda}_N^{-1}\boldsymbol{\eta}_N.$$

Hence,

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) = \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N), \qquad \boxed{\boldsymbol{\Sigma}_N = \boldsymbol{\Lambda}_N^{-1}} \qquad \boxed{\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N\,\boldsymbol{\eta}_N}.$$

**Final posterior formulas (mean/covariance).** Explicitly,

$$\boxed{\boldsymbol{\Sigma}_N = \left(\boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\right)^{-1}}, \qquad \boxed{\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}\right)}.$$

**Interpretation via "precision addition".** It is often clearest to look at precisions:

$$\boldsymbol{\Sigma}_N^{-1} = \boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}.$$

The data contribute a precision term $\frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}$: more data (larger $N$) or cleaner data (smaller $\sigma_w^2$) increase precision and shrink posterior uncertainty.

## 7.4 MAP Connection: BLR as Regularized Least Squares

The MAP estimator is the posterior mode. Since the posterior is Gaussian, the mode equals the mean:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \boldsymbol{\mu}_N.$$

Equivalently, maximize the posterior or minimize the negative log posterior:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_w^2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_0).$$

For $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Sigma}_0 = \alpha^{-1}\boldsymbol{I}$ this becomes ridge regression:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2, \qquad \lambda = \sigma_w^2\alpha.$$

This makes the "prior $\leftrightarrow$ regularizer" correspondence explicit.

## 7.5 Predictive Distributions

### Prior predictive (before observing data)

For a new input $\boldsymbol{x}_*$ with feature vector $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\boldsymbol{x}_*)$, the conditional model is

$$p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\phi}_*^\top \boldsymbol{\theta}, \sigma_w^2).$$

Before seeing data, the predictive distribution averages over the prior:

$$p(y_* \mid \boldsymbol{x}_*) = \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}.$$

Let $z_* = \boldsymbol{\phi}_*^\top \boldsymbol{\theta}$. Since $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and $z_*$ is linear,

$$z_* \sim \mathcal{N}\big(\boldsymbol{\phi}_*^\top \boldsymbol{\mu}_0,\ \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_0 \boldsymbol{\phi}_*\big)\,.$$

With independent additive noise $y_* = z_* + w_*$, $w_* \sim \mathcal{N}(0, \sigma_w^2)$,

$$\boxed{p(y_* \mid \boldsymbol{x}_*) = \mathcal{N}\big(\boldsymbol{\phi}_*^\top \boldsymbol{\mu}_0,\ \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_0 \boldsymbol{\phi}_* + \sigma_w^2\big)\,.}$$

### Posterior predictive (after observing data)

After observing $\mathcal{D}$, we average over the posterior:

$$p(y_* \mid \boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{\Phi}) = \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi})\, d\boldsymbol{\theta}.$$

Using $p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{\Phi}) = \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$, the same linear-Gaussian argument yields

$$\boxed{p(y_* \mid \boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{\Phi}) = \mathcal{N}\big(\boldsymbol{\phi}_*^\top \boldsymbol{\mu}_N,\ \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_N \boldsymbol{\phi}_* + \sigma_w^2\big)\,.}$$

**Mean vs. variance decomposition.** The predictive mean is

$$\mathbb{E}[y_* \mid \cdot] = \boldsymbol{\phi}_*^\top \boldsymbol{\mu}_N,$$

and the predictive variance splits into two terms:

$$\text{Var}(y_* \mid \cdot) = \underbrace{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma}_N \boldsymbol{\phi}_*}_{\text{parameter uncertainty}} + \underbrace{\sigma_w^2}_{\text{measurement noise}}.$$

As more informative data arrive, $\boldsymbol{\Sigma}_N$ shrinks and the uncertainty bands tighten; the irreducible term $\sigma_w^2$ remains as observation noise.

## 7.6 Practical Note: Sampling from the Prior (Function Space Intuition)

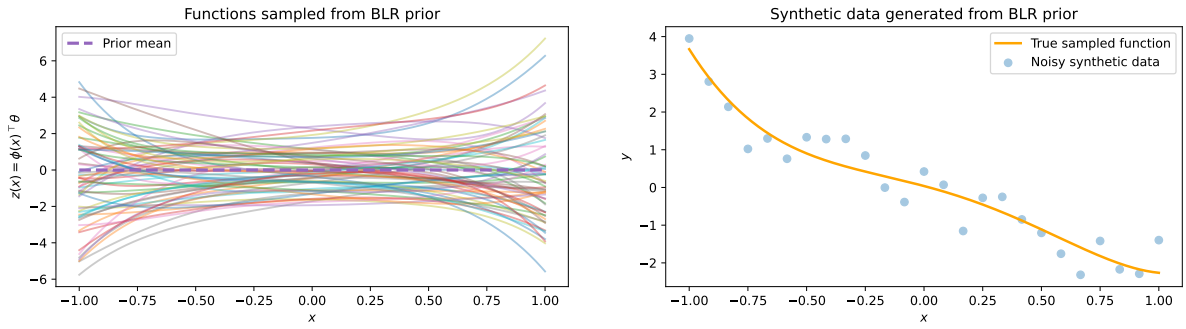
Before observing data, BLR defines a distribution over functions via sampling weights:

$$\boldsymbol{\theta}^{(s)} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \qquad z^{(s)}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\theta}^{(s)}.$$

Adding observation noise yields synthetic datasets:

$$y_i^{(s)} = z^{(s)}(\boldsymbol{x}_i) + w_i^{(s)}, \qquad w_i^{(s)} \sim \mathcal{N}(0, \sigma_w^2).$$

This is a useful way to visualize and debug priors/features: it shows what functions the model believes are plausible *before* seeing any data.




Prior function samples

One synthetic dataset

Figure 5: Bayesian linear regression prior intuition via sampling. Left: sample weights $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and plot the corresponding noiseless functions $z(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\theta}$ to visualize what the prior considers plausible. Right: sample one such function and add Gaussian noise $w \sim \mathcal{N}(0, \sigma_w^2)$ to generate a synthetic training dataset.

Figure 5 illustrates an important intuition behind Bayesian linear regression: before observing any data, the model already defines a *distribution over functions*. This distribution is induced by the prior $p(\boldsymbol{\theta})$ together with the chosen feature map $\boldsymbol{\phi}(\cdot)$.

The left panel shows multiple function samples obtained by first drawing $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0$ and then evaluating the corresponding noiseless prediction $z(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\theta}$ over the input domain. These curves represent the set of functions that the prior considers plausible *before* seeing any data. The choice of features and the prior covariance $\boldsymbol{\Sigma}_0$ strongly influence their smoothness, amplitude, and variability.

The right panel demonstrates how synthetic datasets can be generated from this prior. A single function is sampled from the left panel and Gaussian observation noise $w \sim \mathcal{N}(0, \sigma_w^2)$ is added to produce training points. This process exactly matches the assumed generative model of Bayesian linear regression.

Such prior sampling is useful for several reasons. It provides a sanity check on feature design and prior choices, allowing us to verify that the model encodes reasonable assumptions before fitting data. It also helps build intuition about uncertainty: even in the absence of data, predictions vary because many parameter values are plausible. Finally, it is a convenient way to generate controlled toy datasets for experiments, demos, or assignments under the same assumptions used by the inference procedure.

## 7.7 Worked Example: Bayesian Linear Regression (1D, Closed Form)

We work through a complete Bayesian linear regression example for a simple 1D linear model (intercept + slope). The goal is to compute the posterior over weights and the posterior predictive distribution at a new input.

**Model and features.** Let

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \end{bmatrix}, \qquad y_i = \boldsymbol{\phi}(x_i)^\mathsf{T}\boldsymbol{\theta} + w_i, \qquad w_i \sim \mathcal{N}(0, \sigma_w^2), \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}.$$

Assume noise variance

$$\sigma_w^2 = 1.$$

**Prior.** Use a zero-mean isotropic Gaussian prior

$$p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\boldsymbol{I}), \qquad \alpha = 1 \quad \Rightarrow \quad \boldsymbol{\Sigma}_0 = \boldsymbol{I},\ \boldsymbol{\mu}_0 = \mathbf{0}.$$

**Training data.** Use three observations:

$$(x_1, y_1) = (0, 1), \quad (x_2, y_2) = (1, 2), \quad (x_3, y_3) = (2, 2).$$

Then

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}.$$

**Step 1: Compute posterior covariance $\boldsymbol{\Sigma}_N$**

BLR posterior covariance is

$$\boldsymbol{\Sigma}_N = \left( \boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2} \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} \right)^{-1}.$$

Compute

$$\boldsymbol{\Sigma}_0^{-1} = \boldsymbol{I}, \qquad \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} = \begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix}.$$

Since $\sigma_w^2 = 1$,

$$\boldsymbol{\Sigma}_N^{-1} = \boldsymbol{I} + \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} = \begin{bmatrix} 4 & 3 \\ 3 & 6 \end{bmatrix}.$$

Invert:

$$\det\left(\boldsymbol{\Sigma}_N^{-1}\right) = 4 \cdot 6 - 3 \cdot 3 = 24 - 9 = 15,$$

$$\Rightarrow \quad \boldsymbol{\Sigma}_N = (\boldsymbol{\Sigma}_N^{-1})^{-1} = \frac{1}{15} \begin{bmatrix} 6 & -3 \\ -3 & 4 \end{bmatrix}.$$

**Step 2: Compute posterior mean $\boldsymbol{\mu}_N$**

The posterior mean is

$$\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N \left( \boldsymbol{\Sigma}_0^{-1} \boldsymbol{\mu}_0 + \frac{1}{\sigma_w^2} \boldsymbol{\Phi}^\mathsf{T} \boldsymbol{y} \right).$$

Here $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\sigma_w^2 = 1$, so

$$\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N \boldsymbol{\Phi}^\mathsf{T} \boldsymbol{y}.$$

Compute

$$\boldsymbol{\Phi}^\mathsf{T} \boldsymbol{y} = \begin{bmatrix} 1+2+2 \\ 0 \cdot 1 + 1 \cdot 2 + 2 \cdot 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}.$$

Thus

$$\boldsymbol{\mu}_N = \frac{1}{15} \begin{bmatrix} 6 & -3 \\ -3 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 30-18 \\ -15+24 \end{bmatrix} = \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix}.$$

Therefore,

$$\boxed{p(\boldsymbol{\theta} \mid \mathcal{D}) = \mathcal{N}\left( \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix}, \frac{1}{15} \begin{bmatrix} 6 & -3 \\ -3 & 4 \end{bmatrix} \right).}$$

**Step 3: Posterior predictive at a new input**

Take a test input $x_* = 1.5$, so

$$\boldsymbol{\phi}_* = \boldsymbol{\phi}(1.5) = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}.$$

The posterior predictive distribution is Gaussian:

$$p(y_* \mid x_*, \mathcal{D}) = \mathcal{N}\big(m_*,\ s_*^2\big),$$

with

$$m_* = \boldsymbol{\phi}_*^\mathsf{T} \boldsymbol{\mu}_N, \qquad s_*^2 = \boldsymbol{\phi}_*^\mathsf{T} \boldsymbol{\Sigma}_N \boldsymbol{\phi}_* + \sigma_w^2.$$

Compute the predictive mean:

$$m_* = \begin{bmatrix} 1 & 1.5 \end{bmatrix} \begin{bmatrix} 4/5 \\ 3/5 \end{bmatrix} = \frac{4}{5} + \frac{4.5}{5} = \frac{8.5}{5} = 1.7.$$

Compute the predictive variance term from parameter uncertainty:

$$\boldsymbol{\Sigma}_N \boldsymbol{\phi}_* = \frac{1}{15} \begin{bmatrix} 6 & -3 \\ -3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1.5 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 6-4.5 \\ -3+6 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 1.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}.$$

Then

$$\boldsymbol{\phi}_*^\mathsf{T} \boldsymbol{\Sigma}_N \boldsymbol{\phi}_* = \begin{bmatrix} 1 & 1.5 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = 0.1 + 0.3 = 0.4.$$

Finally,

$$s_*^2 = 0.4 + \sigma_w^2 = 0.4 + 1 = 1.4.$$

So

$$\boxed{p(y_* \mid x_* = 1.5, \mathcal{D}) = \mathcal{N}(1.7,\ 1.4).}$$

**Interpretation.**

- The posterior mean $\boldsymbol{\mu}_N$ is the MAP estimate (Gaussian posterior).
- The posterior covariance $\boldsymbol{\Sigma}_N$ quantifies weight uncertainty and typically shrinks as more data arrive.
- The predictive variance splits into $\boldsymbol{\phi}_*^\mathsf{T} \boldsymbol{\Sigma}_N \boldsymbol{\phi}_*$ (uncertainty about the weights) plus $\sigma_w^2$ (irreducible measurement noise).

## 7.8 Python Implementation

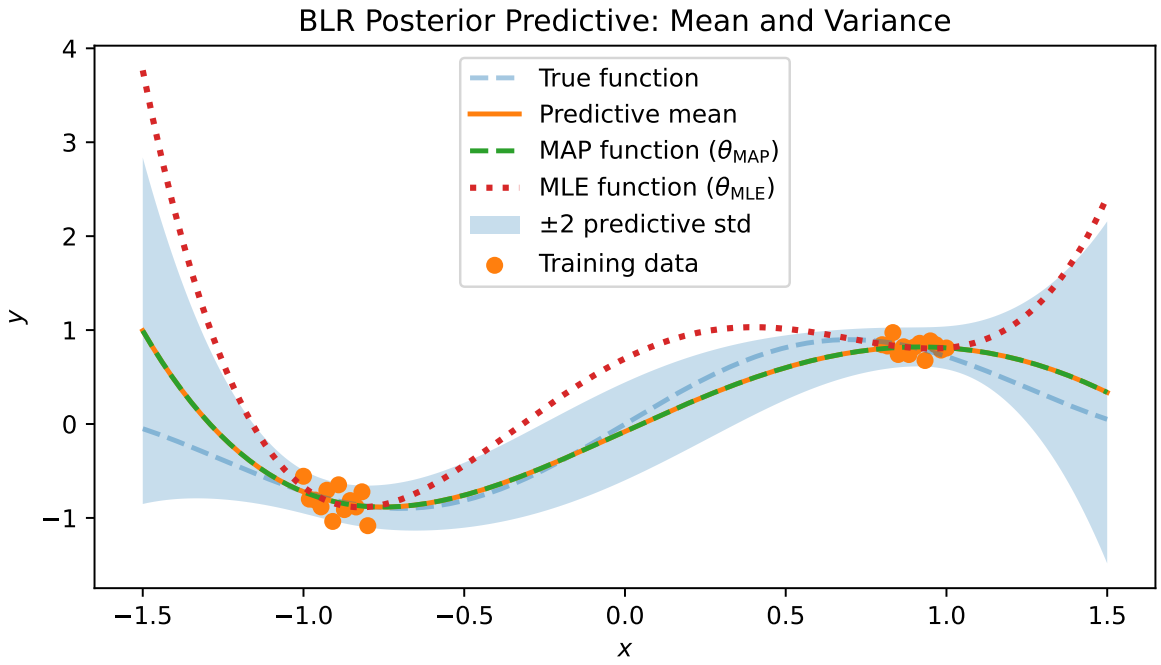



Figure 6: Posterior predictive behavior in Bayesian linear regression (BLR), compared to MLE and MAP point estimates. The training data are concentrated in two separated input regions (near $x \approx -1$ and $x \approx +1$), leaving a gap with no observations. The figure shows: (i) the BLR *predictive mean* (posterior predictive mean), (ii) an uncertainty band of $\pm 2$ predictive standard deviations, (iii) the *MAP function* induced by the MAP weights $\boldsymbol{\theta}_{\text{MAP}}$ (for a Gaussian posterior, this equals the posterior mean), and (iv) the *MLE function* induced by $\boldsymbol{\theta}_{\text{MLE}}$ (no prior).

The code constructs a synthetic regression problem where the underlying function is nonlinear, but we deliberately fit a *polynomial feature model* of degree $P$:

$$\phi(x) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^P \end{bmatrix}^\mathsf{T}, \qquad y \approx \boldsymbol{\theta}^\mathsf{T} \phi(x) + w, \qquad w \sim \mathcal{N}(0, \sigma_w^2).$$

This is a typical "linear model on nonlinear features" setting: the model is linear in $\boldsymbol{\theta}$, but nonlinear in $x$.

**Dataset design: why the gap matters.** The inputs are sampled from two disjoint intervals:

$$x \in [-1, -0.8] \ \cup \ [0.8, 1],$$

so the model sees no data in the middle region. This is precisely where Bayesian predictive uncertainty becomes informative: BLR should be confident where data exist and less confident where they do not.

**Three different predictors in the plot.**

- **MLE function** (point estimate, no prior). The code computes

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y} \mid \boldsymbol{\theta}) \quad \Longleftrightarrow \quad \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2,$$

  implemented as $\boldsymbol{\theta}_{\text{MLE}} = \text{pinv}(\boldsymbol{\Phi})\, \boldsymbol{y}$. This yields a single curve $\hat{y}(x) = \phi(x)^\top \boldsymbol{\theta}_{\text{MLE}}$ with *no uncertainty*.

- **MAP function** (point estimate with prior). The prior is Gaussian,

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \qquad \boldsymbol{\Sigma}_0 = \alpha^{-1}\boldsymbol{I},$$

  which leads to a MAP estimator equivalent to ridge regression:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_w^2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_0).$$

  The code computes $\boldsymbol{\theta}_{\text{MAP}}$ explicitly in closed form. This again gives a single curve, typically smoother / less extreme than MLE due to shrinkage.

- **BLR posterior predictive** (distribution over outputs). BLR does not return a single function only; it returns a predictive distribution:

$$p(y_* \mid x_*, \mathcal{D}) = \mathcal{N}\Big( \underbrace{\phi(x_*)^\top \boldsymbol{\mu}_N}_{\text{predictive mean}} , \underbrace{\phi(x_*)^\top \boldsymbol{\Sigma}_N \phi(x_*)}_{\text{parameter uncertainty}} + \underbrace{\sigma_w^2}_{\text{noise}} \Big).$$

  The shaded band in the plot corresponds to $\pm 2$ predictive standard deviations, i.e. roughly a 95% interval under the Gaussian predictive model.

**How to read the figure (main takeaways).**

- **Point estimators (MLE/MAP) produce curves but no uncertainty.** They cannot express that the prediction in the gap is less reliable than near the observed regions.
- **BLR uncertainty grows away from the data.** In regions with few/no training points, $\phi(x)^\top \boldsymbol{\Sigma}_N \phi(x)$ increases, widening the predictive band.
- **MAP vs MLE differs most when data are limited or features are flexible.** The prior shrinks coefficients and prevents extreme polynomial behavior, so the MAP curve is often less oscillatory than MLE (especially outside the observed range).

**Implementation details mirrored in the code.** The code uses the closed-form posterior:

$$\boldsymbol{\Sigma}_N^{-1} = \boldsymbol{\Sigma}_0^{-1} + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}, \qquad \boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \frac{1}{\sigma_w^2}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{y}\right),$$

then computes predictive mean and variance on a grid using

$$\mathrm{Var}(y_* \mid \cdot) = \mathrm{diag}(\boldsymbol{\Phi}_*\boldsymbol{\Sigma}_N\boldsymbol{\Phi}_*^\mathsf{T}) + \sigma_w^2,$$

implemented via the efficient diagonal trick

$$\mathrm{diag}(\boldsymbol{\Phi}_*\boldsymbol{\Sigma}_N\boldsymbol{\Phi}_*^\mathsf{T}) = \sum_j (\boldsymbol{\Phi}_*\boldsymbol{\Sigma}_N)_{\cdot j} \odot (\boldsymbol{\Phi}_*)_{\cdot j}.$$

This is why the code computes `param_var = np.sum(Phi_grid @ SigmaN * Phi_grid, axis=1)`.

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

# ---------------------------
# 1) Synthetic dataset
# ---------------------------
np.random.seed(1)

# N = 25
# x_data = np.linspace(-1., 1., N)
N = 25
N_left = N // 2
N_right = N - N_left

x_left  = np.linspace(-1.0, -0.8, N_left, endpoint=True)
x_right = np.linspace(0.8,  1.0, N_right, endpoint=True)

x_data = np.concatenate([x_left, x_right])
sigma_w = 0.1    # noise std

# True underlying function (just for data gen)
def f_true(x):
    return 0.7*np.sin(2.5*x) + 0.3*x

y_data = f_true(x_data) + sigma_w*np.random.randn(N)

# ---------------------------
# 2) Feature map (polynomial)
# ---------------------------
P = 4   # order -> feature dim = P+1

def phi(x, P):
    return np.vstack([x**k for k in range(P+1)]).T

Phi_data = phi(x_data, P)   # (N, P+1)

# Grid for prediction
x_grid = np.linspace(-1.5, 1.5, 300)
Phi_grid = phi(x_grid, P)

y_grid = f_true(x_grid)
```

```
42
43  # ---------------------------
44  # 3) Prior over theta
45  # ---------------------------
46  alpha = 2.0
47  mu0 = np.zeros(P+1)
48  Sigma0 = (1/alpha) * np.eye(P+1)
49
50  # ---------------------------
51  # 4) Posterior (closed form)
52  # ---------------------------
53  SigmaN_inv = np.linalg.inv(Sigma0) + (1/sigma_w**2) * (Phi_data.T @
        Phi_data)
54  SigmaN = np.linalg.inv(SigmaN_inv)
55
56  muN = SigmaN @ (np.linalg.inv(Sigma0) @ mu0 + (1/sigma_w**2) * Phi_data
        .T @ y_data)
57
58  # ---------------------------
59  # 4b) MAP estimate for theta
60  # ---------------------------
61  # For Gaussian posterior, MAP = posterior mean
62  # theta_map = muN.copy()
63
64  # (Optional explicit ridge/MAP form; should match muN)
65  theta_map = np.linalg.inv(Phi_data.T @ Phi_data + sigma_w**2 * np.
        linalg.inv(Sigma0)) @ (Phi_data.T @ y_data + sigma_w**2 * np.linalg.
        inv(Sigma0) @ mu0)
66
67  # ---------------------------
68  # 4c) MLE estimate for theta (no prior)
69  # ---------------------------
70  # With N < P+1, use pseudoinverse for stability
71  theta_mle = np.linalg.pinv(Phi_data) @ y_data
72  y_mle = Phi_grid @ theta_mle
73
74  # Function induced by MAP parameters
75  y_map = Phi_grid @ theta_map
76
77  # ---------------------------
78  # 5) Posterior predictive
79  # ---------------------------
80  # Mean
81  y_mean = Phi_grid @ muN
82
83  # Variance = parameter uncertainty + noise
84  param_var = np.sum(Phi_grid @ SigmaN * Phi_grid, axis=1)  # diag(Phi Σ
        Phi^T)
85  pred_var = param_var + sigma_w**2
86  pred_std = np.sqrt(pred_var)
87
88  # ---------------------------
89  # 6) Plot
90  # ---------------------------
91  plt.figure(figsize=(6.8, 4.0))
92
93  # the actual function
94  plt.plot(x_grid, y_grid, "--", linewidth=2, alpha=0.4, label="True
```

```
    function")

# predictive mean
plt.plot(x_grid, y_mean, linewidth=2, label="Predictive mean")

# MAP function
plt.plot(x_grid, y_map, "--", linewidth=2, label=r"MAP function ($\
    theta_{\mathrm{MAP}}$)")

# MLE function
plt.plot(x_grid, y_mle, ":", linewidth=2.5, label=r"MLE function ($\
    theta_{\mathrm{MLE}}$)")

# uncertainty band ±2 std
plt.fill_between(
    x_grid,
    y_mean - 2*pred_std,
    y_mean + 2*pred_std,
    alpha=0.25,
    label=r"$\pm 2$ predictive std"
)

# data
plt.scatter(x_data, y_data, s=35, label="Training data")

plt.xlabel(r"$x$")
plt.ylabel(r"$y$")
plt.title("BLR Posterior Predictive: Mean and Variance")
plt.legend()
plt.tight_layout()
plt.show()

print("theta_MAP =", theta_map)
print("theta_MLE =", theta_mle)
```

Listing 2: Bayesian Linear Regression

## 7.9 Sequential Bayesian Linear Regression (Recursive Update)

A major practical advantage of Bayesian linear regression is that it can be updated *online*: when a new sample arrives, we update the posterior without re-processing the whole dataset. Suppose that after observing

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

our posterior is

$$p(\boldsymbol{\theta} \mid \mathcal{D}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

A new data point $(\boldsymbol{x}_{k+1}, y_{k+1})$ induces a feature vector

$$\boldsymbol{\phi}_{k+1} \equiv \boldsymbol{\phi}(\boldsymbol{x}_{k+1}), \qquad y_{k+1} = \boldsymbol{\phi}_{k+1}^{\mathsf{T}}\boldsymbol{\theta} + w_{k+1}, \qquad w_{k+1} \sim \mathcal{N}(0, \sigma_w^2).$$

Bayes' rule says the updated posterior is proportional to the previous posterior times the new likelihood:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{k+1}) \propto p(y_{k+1} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D}_k).$$

Because both factors are Gaussian in $\boldsymbol{\theta}$, the posterior remains Gaussian:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{k+1}) = \mathcal{N}(\boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1}).$$

It is especially clean to express the recursion in *information form*. Define the precision (information) matrix

$$\boldsymbol{\Lambda}_k \triangleq \boldsymbol{\Sigma}_k^{-1}.$$

Then each new sample contributes a rank-1 information increment:

$$\boxed{\boldsymbol{\Lambda}_{k+1} = \boldsymbol{\Lambda}_k + \frac{1}{\sigma_w^2}\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_{k+1}^{\mathsf{T}}.}$$

Similarly, define the information vector

$$\boldsymbol{\eta}_k \triangleq \boldsymbol{\Lambda}_k \boldsymbol{\mu}_k.$$

The mean update becomes a simple additive rule in information space:

$$\boxed{\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \frac{1}{\sigma_w^2}\boldsymbol{\phi}_{k+1} y_{k+1}, \qquad \boldsymbol{\mu}_{k+1} = \boldsymbol{\Lambda}_{k+1}^{-1}\boldsymbol{\eta}_{k+1}.}$$

This representation highlights the interpretation of sequential BLR: we keep accumulating information about $\boldsymbol{\theta}$. When the observation noise variance $\sigma_w^2$ is small, each sample is trusted more and adds more information; when $\sigma_w^2$ is large, the update is smaller.

The same recursion can also be written in a Kalman-filter-like form, which may be more intuitive if you have seen recursive least squares or Kalman updates. The one-step predictive distribution of the new output is

$$y_{k+1} \mid \mathcal{D}_k \sim \mathcal{N}\left(\boldsymbol{\phi}_{k+1}^{\mathsf{T}}\boldsymbol{\mu}_k,\ s_{k+1}\right), \qquad s_{k+1} \triangleq \boldsymbol{\phi}_{k+1}^{\mathsf{T}}\boldsymbol{\Sigma}_k\boldsymbol{\phi}_{k+1} + \sigma_w^2.$$

Define the innovation (prediction error)

$$e_{k+1} \triangleq y_{k+1} - \boldsymbol{\phi}_{k+1}^{\mathsf{T}}\boldsymbol{\mu}_k,$$

and the gain vector

$$\boldsymbol{K}_{k+1} \triangleq \frac{\boldsymbol{\Sigma}_k\boldsymbol{\phi}_{k+1}}{s_{k+1}}.$$

Then the posterior mean and covariance update as

$$\boxed{\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \boldsymbol{K}_{k+1}e_{k+1}, \qquad \boldsymbol{\Sigma}_{k+1} = \boldsymbol{\Sigma}_k - \boldsymbol{K}_{k+1}\boldsymbol{\phi}_{k+1}^{\mathsf{T}}\boldsymbol{\Sigma}_k.}$$

This is exactly the measurement update of a Kalman filter for a *static* state $\boldsymbol{\theta}$ (no dynamics), with measurement matrix $\boldsymbol{\phi}_{k+1}^{\mathsf{T}}$.

## 7.10 What Bayesian Linear Regression Adds Beyond LS / MLE

Least squares and maximum likelihood estimation provide a *single* best-fit parameter vector. In contrast, BLR produces a full posterior distribution

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N),$$

which directly quantifies how uncertain we are about each parameter and how parameters are correlated.

A second benefit is that BLR incorporates regularization in a principled way through the prior. For a Gaussian prior, the MAP estimate coincides with the posterior mean and is equivalent to ridge regression:

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_w^2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2^2 + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^{\mathsf{T}}\boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_0).$$

Thus, the prior controls model complexity and improves numerical stability, especially when data are scarce or the feature dimension is large.

Most importantly, BLR yields *predictive uncertainty*. For a new input $\boldsymbol{x}_*$ with features $\boldsymbol{\phi}_*$, the posterior predictive distribution is

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}\big(\boldsymbol{\phi}_*^{\mathsf{T}}\boldsymbol{\mu}_N,\ \boldsymbol{\phi}_*^{\mathsf{T}}\boldsymbol{\Sigma}_N\boldsymbol{\phi}_* + \sigma_w^2\big)\,.$$

The predictive variance decomposes into two intuitive parts:

$$\mathrm{Var}(y_* \mid \cdot) = \underbrace{\boldsymbol{\phi}_*^{\mathsf{T}}\boldsymbol{\Sigma}_N\boldsymbol{\phi}_*}_{\text{uncertainty in } \boldsymbol{\theta}} + \underbrace{\sigma_w^2}_{\text{observation noise}}\,.$$

The first term becomes small when the data strongly constrain the weights (posterior concentrates), while the second term is irreducible measurement noise.

Finally, BLR connects naturally to the asymptotic behavior of MLE: as $N$ grows and the data are informative, the posterior covariance shrinks and the posterior mean approaches the MLE under standard regularity conditions. In that sense, BLR can be viewed as LS/MLE augmented with prior knowledge and a coherent uncertainty quantification mechanism.