Lecture 12 & 13: Optimal Filters Notes

Konstantinos Chatzilygeroudis
costashatz@upatras.gr

December 21, 2025

Contents

1 From Hand-Crafted to Optimal Filters

Classical filtering is often introduced through *hand-crafted* designs such as FIR/IIR filters, moving averages, and standard bandpass or notch filters. In this paradigm, we first *choose* a filter structure and then *tune* its parameters (e.g., cutoff frequencies, filter order, window length) based on intuition about the spectrum. This works well when the spectral separation between signal and noise is clear and stable.

In many real-world problems, however, the assumptions behind manual tuning are fragile. Noise may be colored rather than white, and it may vary over time (nonstationary behavior). Moreover, signal and noise spectra may overlap, so there is rarely a uniquely "right" cutoff frequency. Different design choices therefore lead to different tradeoffs between *noise suppression* and *signal distortion*, and it is not always obvious how to select the best compromise.

This motivates a natural question:

> *Can we design a filter automatically, in an optimal way, once we state what "optimal" means?*

The *optimal filtering* viewpoint answers this by turning filter design into an estimation problem. Instead of choosing a filter heuristically, we specify (i) a **goal** (e.g., estimate or reconstruct a clean signal), (ii) a **criterion** (most commonly, minimize mean squared error), and (iii) a **model** for the signal and noise (typically via second-order statistics). The resulting filter is then derived as the one that achieves the *best possible performance* under these assumptions.

In **Wiener filtering**, we consider a static (non-recursive) optimal linear estimator: given a noisy observation process, we seek the linear filter that minimizes the mean squared estimation error. This leads to the Wiener–Hopf equations and yields a closed-form optimal solution in terms of correlation functions (or, equivalently, power spectral densities). Wiener filtering is particularly useful when we can assume stationarity and when a batch/offline filter is acceptable.

In **Kalman filtering**, we move from static estimation to *dynamic* estimation. Here the quantity of interest evolves over time according to a *state-space model*, and measurements arrive sequentially. Rather than computing a single batch solution, we aim for a *recursive* estimator that updates its belief about the state whenever a new measurement becomes available. Under linear dynamics, linear measurements, and Gaussian noise assumptions, the Kalman filter provides the optimal *minimum mean squared error* (MMSE) estimate of the state, and it does so efficiently by propagating only the current estimate and its uncertainty (covariance).

Thus, Wiener and Kalman filtering can be seen as two classical instances of optimal filtering: Wiener filtering addresses optimal linear MMSE estimation in (typically) stationary settings, while Kalman filtering generalizes the idea to time-varying systems with a principled recursive update. Together, they illustrate the central theme of this module: once we define a model and an optimality criterion, the filter follows from the mathematics rather than from ad hoc tuning.

2 Wiener Filtering: The Optimal Linear MMSE Estimator

We now derive the Wiener filter, i.e., the classical optimal linear estimator that minimizes mean squared error under standard second-order (and, optionally, Gaussian) assumptions.

2.1 Assumptions and Modeling

Wiener filtering can be interpreted as the Bayesian MMSE solution (posterior mean) in a linear–Gaussian setting, and equivalently as the *linear MMSE* (LMMSE) solution when only second-order statistics are assumed. Concretely, we adopt the following model.

1) Linear observation model. We assume that the measurements $\boldsymbol{x}[n] \in \mathbb{R}^p$ are generated from the unknown (clean) signal $\boldsymbol{d}[n] \in \mathbb{R}^q$ via

$$\boldsymbol{x}[n] = \boldsymbol{A}\,\boldsymbol{d}[n] + \boldsymbol{v}[n], \tag{2.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{p\times q}$ is a known linear operator. For denoising, $\boldsymbol{A} = \boldsymbol{I}$; for deconvolution or deblurring, $\boldsymbol{A}$ is often a convolution operator.

2) Second-order signal model (WSS). The clean signal $\boldsymbol{d}[n]$ is modeled as a (wide-sense) stationary random process, characterized by its autocorrelation $\boldsymbol{R}_{dd}[k]$ (or, equivalently, its power spectral density $\boldsymbol{S}_{dd}(\omega)$ in the scalar/WSS setting).

3) Additive noise model (WSS) and signal–noise uncorrelatedness. The noise $\boldsymbol{v}[n]$ is modeled as WSS with autocorrelation $\boldsymbol{R}_{vv}[k]$ (or PSD $\boldsymbol{S}_{vv}(\omega)$), and we assume it is uncorrelated with the signal:

$$\boldsymbol{R}_{dv}[k] = \boldsymbol{0}. \tag{2.2}$$

4) Gaussianity (Bayesian justification). If, additionally, $\boldsymbol{d}$ and $\boldsymbol{v}$ are modeled as Gaussian processes, then $p(\boldsymbol{d} \mid \boldsymbol{x})$ is Gaussian and the posterior mean is linear in $\boldsymbol{x}$. This implies that the MMSE estimator coincides with the optimal linear estimator.

5) Restriction to linear estimators. We search for an estimator within the class of linear mappings, written (at a fixed time index n) as

$$\hat{\boldsymbol{d}}[n] = \boldsymbol{W}^T \boldsymbol{x}[n], \tag{2.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{p\times q}$ (so that $\boldsymbol{W}^T\boldsymbol{x} \in \mathbb{R}^q$). In the LTI/WSS setting, this linear mapping corresponds to convolution with an impulse response (i.e., an LTI filter). The Wiener filter is the particular choice of $\boldsymbol{W}$ that minimizes mean squared error.

2.2 Linear MMSE Objective

We seek the linear estimator that minimizes the expected squared reconstruction error:

$$\boldsymbol{W}_\star = \arg\min_{\boldsymbol{W}} \ \mathbb{E}\Big[\big\|\boldsymbol{d}[n] - \boldsymbol{W}^T\boldsymbol{x}[n]\big\|_2^2\Big]. \tag{2.4}$$

For notational simplicity, we drop $[n]$ and write $\boldsymbol{x}, \boldsymbol{d}$ when no confusion is possible.

Expanding the quadratic form yields

$$\begin{aligned} J(\boldsymbol{W}) &= \mathbb{E}\big[(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})^T(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})\big] \\ &= \operatorname{tr}\big(\mathbb{E}\{\boldsymbol{d}\boldsymbol{d}^T\}\big) - 2\operatorname{tr}\big(\boldsymbol{W}^T\mathbb{E}\{\boldsymbol{x}\boldsymbol{d}^T\}\big) + \operatorname{tr}\big(\boldsymbol{W}^T\mathbb{E}\{\boldsymbol{x}\boldsymbol{x}^T\}\boldsymbol{W}\big). \end{aligned} \tag{2.5}$$

Define the (cross-)correlation matrices

$$\boldsymbol{R}_{dd} \triangleq \mathbb{E}\{\boldsymbol{d}\boldsymbol{d}^T\}, \qquad \boldsymbol{R}_{xd} \triangleq \mathbb{E}\{\boldsymbol{x}\boldsymbol{d}^T\}, \qquad \boldsymbol{R}_{xx} \triangleq \mathbb{E}\{\boldsymbol{x}\boldsymbol{x}^T\}. \tag{2.6}$$

Then (2.5) can be written compactly as

$$J(\boldsymbol{W}) = \operatorname{tr}(\boldsymbol{R}_{dd}) - 2\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xd}\big) + \operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xx}\boldsymbol{W}\big). \tag{2.7}$$

2.2.1 Wiener–Hopf (Normal) Equations and Closed-Form Solution

Differentiating $J(\boldsymbol{W})$ with respect to $\boldsymbol{W}$ and setting the gradient to zero yields the normal (Wiener–Hopf) equations:

$$\frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{0} \quad \Longrightarrow \quad \boldsymbol{R}_{xx}\,\boldsymbol{W}_\star = \boldsymbol{R}_{xd}. \tag{2.8}$$

Assuming $\boldsymbol{R}_{xx}$ is invertible, we obtain the Wiener/LMMSE solution

$$\boxed{\boldsymbol{W}_\star = \boldsymbol{R}_{xx}^{-1}\,\boldsymbol{R}_{xd}.} \tag{2.9}$$

Using the measurement model (2.1) and the assumption that $\boldsymbol{v} \perp \boldsymbol{d}$, we can express $\boldsymbol{R}_{xd}$ and $\boldsymbol{R}_{xx}$ in terms of the model covariances. Indeed,

$$\boldsymbol{R}_{xd} = \mathbb{E}\{(\boldsymbol{A}\boldsymbol{d} + \boldsymbol{v})\boldsymbol{d}^T\} = \boldsymbol{A}\,\mathbb{E}\{\boldsymbol{d}\boldsymbol{d}^T\} + \mathbb{E}\{\boldsymbol{v}\boldsymbol{d}^T\} = \boldsymbol{A}\boldsymbol{R}_{dd}, \tag{2.10}$$

$$\begin{aligned} \boldsymbol{R}_{xx} &= \mathbb{E}\{(\boldsymbol{A}\boldsymbol{d} + \boldsymbol{v})(\boldsymbol{A}\boldsymbol{d} + \boldsymbol{v})^T\} \\ &= \boldsymbol{A}\boldsymbol{R}_{dd}\boldsymbol{A}^T + \boldsymbol{R}_{vv}, \end{aligned} \tag{2.11}$$

where $\boldsymbol{R}_{vv} \triangleq \mathbb{E}\{\boldsymbol{v}\boldsymbol{v}^T\}$. Substituting (2.10)–(2.11) into (2.9) gives the commonly used closed form:

$$\boxed{\boldsymbol{W}_\star = \big(\boldsymbol{A}\boldsymbol{R}_{dd}\boldsymbol{A}^T + \boldsymbol{R}_{vv}\big)^{-1}\boldsymbol{A}\boldsymbol{R}_{dd}.} \tag{2.12}$$

Remark (correlation vs. covariance). Under the standard assumption of zero-mean processes, $\mathbb{E}\{\boldsymbol{x}\} = \boldsymbol{0}$ and $\mathbb{E}\{\boldsymbol{d}\} = \boldsymbol{0}$, the correlation and covariance matrices coincide:

$$\operatorname{Cov}[\boldsymbol{x}, \boldsymbol{x}] = \mathbb{E}\big[(\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{x} - \boldsymbol{\mu}_x)^T\big] = \mathbb{E}\{\boldsymbol{x}\boldsymbol{x}^T\} = \boldsymbol{R}_{xx}, \tag{2.13}$$

and similarly

$$\operatorname{Cov}[\boldsymbol{x}, \boldsymbol{d}] = \mathbb{E}\big[(\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{d} - \boldsymbol{\mu}_d)^T\big] = \mathbb{E}\{\boldsymbol{x}\boldsymbol{d}^T\} = \boldsymbol{R}_{xd}. \tag{2.14}$$

2.3 Detailed Derivation of the Wiener Filter

We now provide a detailed derivation of the Wiener filter by explicitly minimizing the mean squared error over the class of linear estimators. Throughout, we assume real-valued random vectors and zero-mean processes.

Let $\boldsymbol{x} \in \mathbb{R}^p$ denote the observation vector, $\boldsymbol{d} \in \mathbb{R}^q$ the desired (clean) signal, and $\boldsymbol{W} \in \mathbb{R}^{p \times q}$ the linear estimator matrix such that $\hat{\boldsymbol{d}} = \boldsymbol{W}^T\boldsymbol{x} \in \mathbb{R}^q$.

2.3.1 Objective Function

The linear MMSE objective is

$$J(\boldsymbol{W}) = \mathbb{E}\big[\|\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x}\|_2^2\big] = \mathbb{E}\big[(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})^T(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})\big] . \tag{2.15}$$

Using the trace operator, which allows convenient manipulation of matrix-valued expressions, we can write

$$\begin{aligned} J(\boldsymbol{W}) &= \mathbb{E}\big[\operatorname{tr}\big((\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})^T(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})\big)\big] \\ &= \operatorname{tr}\big(\mathbb{E}\big[(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})^T(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})\big]\big) , \end{aligned} \tag{2.16}$$

where linearity of the trace and expectation has been used.

2.3.2 Expanding the Quadratic Form

We first expand the quadratic term:

$$(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x})^T(\boldsymbol{d} - \boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{d}^T\boldsymbol{d} - \boldsymbol{d}^T\boldsymbol{W}^T\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{d} + \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x}. \tag{2.17}$$

Substituting this expansion into the objective yields

$$J(\boldsymbol{W}) = \operatorname{tr}\big(\mathbb{E}[\boldsymbol{d}^T\boldsymbol{d}]\big) - \operatorname{tr}\big(\mathbb{E}[\boldsymbol{d}^T\boldsymbol{W}^T\boldsymbol{x}]\big) - \operatorname{tr}\big(\mathbb{E}[\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{d}]\big) + \operatorname{tr}\big(\mathbb{E}[\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x}]\big). \tag{2.18}$$

2.3.3 Evaluating Each Term

First term. Using $\boldsymbol{d}^T\boldsymbol{d} = \operatorname{tr}\big(\boldsymbol{d}\boldsymbol{d}^T\big)$, we obtain

$$\operatorname{tr}\big(\mathbb{E}[\boldsymbol{d}^T\boldsymbol{d}]\big) = \operatorname{tr}\big(\mathbb{E}[\boldsymbol{d}\boldsymbol{d}^T]\big) = \operatorname{tr}(\boldsymbol{R}_{dd}), \tag{2.19}$$

where $\boldsymbol{R}_{dd} \triangleq \mathbb{E}[\boldsymbol{d}\boldsymbol{d}^T]$.

Second term. Using cyclic permutation of the trace,

$$\begin{aligned} \operatorname{tr}\big(\mathbb{E}[\boldsymbol{d}^T\boldsymbol{W}^T\boldsymbol{x}]\big) &= \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{d}^T\boldsymbol{W}^T\boldsymbol{x}\big)\big] = \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{x}\boldsymbol{d}^T\big)\big] \\ &= \operatorname{tr}\big(\boldsymbol{W}^T\,\mathbb{E}[\boldsymbol{x}\boldsymbol{d}^T]\big) = \operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xd}\big), \end{aligned} \tag{2.20}$$

where $\boldsymbol{R}_{xd} \triangleq \mathbb{E}[\boldsymbol{x}\boldsymbol{d}^T]$.

Third term. Similarly,

$$\begin{aligned} \operatorname{tr}\big(\mathbb{E}[\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{d}]\big) &= \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{d}\big)\big] = \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{W}\boldsymbol{d}\boldsymbol{x}^T\big)\big] \\ &= \operatorname{tr}\big(\boldsymbol{W}\,\mathbb{E}[\boldsymbol{d}\boldsymbol{x}^T]\big) = \operatorname{tr}(\boldsymbol{W}\boldsymbol{R}_{dx}), \end{aligned} \tag{2.21}$$

with $\boldsymbol{R}_{dx} = \boldsymbol{R}_{xd}^T$. Using $\operatorname{tr}(\boldsymbol{A}) = \operatorname{tr}\big(\boldsymbol{A}^T\big)$, this becomes

$$\operatorname{tr}(\boldsymbol{W}\boldsymbol{R}_{dx}) = \operatorname{tr}\big(\boldsymbol{R}_{xd}\boldsymbol{W}^T\big) = \operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xd}\big). \tag{2.22}$$

Fourth term. Finally,

$$\begin{aligned}\operatorname{tr}\big(\mathbb{E}[\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x}]\big) &= \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x}\big)\big] \\ &= \mathbb{E}\big[\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}\big)\big] \\ &= \operatorname{tr}\big(\boldsymbol{W}^T\,\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]\,\boldsymbol{W}\big) = \operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xx}\boldsymbol{W}\big),\end{aligned} \tag{2.23}$$

where $\boldsymbol{R}_{xx} \triangleq \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$.

2.3.4 Final Cost Function

Collecting all terms, the objective function becomes

$$\boxed{J(\boldsymbol{W}) = \operatorname{tr}(\boldsymbol{R}_{dd}) - 2\,\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xd}\big) + \operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{R}_{xx}\boldsymbol{W}\big)}. \tag{2.24}$$

2.3.5 Gradient and Optimality Condition

We now differentiate $J(\boldsymbol{W})$ with respect to $\boldsymbol{W}$. The required matrix calculus identities are:

$$\frac{\partial}{\partial \boldsymbol{W}}\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{A}\big) = \boldsymbol{A}, \qquad \frac{\partial}{\partial \boldsymbol{W}}\operatorname{tr}\big(\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\big) = (\boldsymbol{A}+\boldsymbol{A}^T)\boldsymbol{W}.$$

Since $\boldsymbol{R}_{xx}$ is symmetric, $\boldsymbol{R}_{xx} = \boldsymbol{R}_{xx}^T$, we obtain

$$\frac{\partial J}{\partial \boldsymbol{W}} = -2\boldsymbol{R}_{xd} + 2\boldsymbol{R}_{xx}\boldsymbol{W}. \tag{2.25}$$

Setting the gradient to zero yields the Wiener–Hopf equations

$$\boxed{\boldsymbol{R}_{xx}\boldsymbol{W}_\star = \boldsymbol{R}_{xd},} \tag{2.26}$$

which lead directly to the Wiener filter solution $\boldsymbol{W}_\star = \boldsymbol{R}_{xx}^{-1}\boldsymbol{R}_{xd}$.

2.3.6 Computing $\boldsymbol{R}_{xd}$ and $\boldsymbol{R}_{xx}$ from the Observation Model

We now explicitly compute the cross-covariance $\boldsymbol{R}_{xd}$ and the observation covariance $\boldsymbol{R}_{xx}$ from the assumed linear measurement model. Recall that the observations are generated according to

$$\boldsymbol{x}[n] = \boldsymbol{A}\,\boldsymbol{d}[n] + \boldsymbol{v}[n], \tag{2.27}$$

where $\mathbb{E}[\boldsymbol{d}] = \boldsymbol{0}$, $\mathbb{E}[\boldsymbol{v}] = \boldsymbol{0}$, and the noise is uncorrelated with the signal, i.e., $\boldsymbol{v} \perp \boldsymbol{d}$.

Cross-covariance between observation and signal. By definition,

$$\boldsymbol{R}_{xd} \triangleq \mathbb{E}\{\boldsymbol{x}\boldsymbol{d}^T\}. \tag{2.28}$$

Substituting the observation model (2.27) gives

$$\begin{aligned}\boldsymbol{R}_{xd} &= \mathbb{E}\{(\boldsymbol{A}\boldsymbol{d}+\boldsymbol{v})\boldsymbol{d}^T\} \\ &= \boldsymbol{A}\,\mathbb{E}\{\boldsymbol{d}\boldsymbol{d}^T\} + \mathbb{E}\{\boldsymbol{v}\boldsymbol{d}^T\}.\end{aligned} \tag{2.29}$$

Since $\boldsymbol{v}$ and $\boldsymbol{d}$ are uncorrelated, the second term vanishes: $\mathbb{E}\{\boldsymbol{v}\boldsymbol{d}^T\} = \boldsymbol{0}$. Therefore,

$$\boxed{\boldsymbol{R}_{xd} = \boldsymbol{A}\boldsymbol{R}_{dd}.} \tag{2.30}$$

Auto-covariance of the observations. Similarly, the observation covariance is

$$\boldsymbol{R}_{xx} \triangleq \mathbb{E}\{\boldsymbol{x}\boldsymbol{x}^T\} = \mathbb{E}\{(\boldsymbol{A}\boldsymbol{d} + \boldsymbol{v})(\boldsymbol{A}\boldsymbol{d} + \boldsymbol{v})^T\}. \tag{2.31}$$

Expanding the product yields

$$\boldsymbol{R}_{xx} = \boldsymbol{A}\mathbb{E}\{\boldsymbol{d}\boldsymbol{d}^T\}\boldsymbol{A}^T + \boldsymbol{A}\mathbb{E}\{\boldsymbol{d}\boldsymbol{v}^T\} + \mathbb{E}\{\boldsymbol{v}\boldsymbol{d}^T\}\boldsymbol{A}^T + \mathbb{E}\{\boldsymbol{v}\boldsymbol{v}^T\}. \tag{2.32}$$

Again, uncorrelatedness implies that the cross terms vanish:

$$\mathbb{E}\{\boldsymbol{d}\boldsymbol{v}^T\} = \mathbb{E}\{\boldsymbol{v}\boldsymbol{d}^T\} = \boldsymbol{0}.$$

Thus, the observation covariance reduces to

$$\boxed{\boldsymbol{R}_{xx} = \boldsymbol{A}\boldsymbol{R}_{dd}\boldsymbol{A}^T + \boldsymbol{R}_{vv},} \tag{2.33}$$

where $\boldsymbol{R}_{vv} \triangleq \mathbb{E}\{\boldsymbol{v}\boldsymbol{v}^T\}$.

Remark. Equations (2.30) and (2.33) show that the Wiener filter depends only on the linear operator $\boldsymbol{A}$ and on second-order statistics of the signal and noise. No higher-order moments are required, and Gaussianity is only needed if one wishes to interpret the solution as the full Bayesian MMSE estimator rather than the linear MMSE solution.

2.4 Using the Wiener Filter in Practice: Offline Targets vs. Second-Order Statistics

The Wiener solution is written in terms of correlation matrices such as $\boldsymbol{R}_{xx}$ and $\boldsymbol{R}_{xd}$, which are expectations over the (unknown) data-generating process. In practice, we rarely have access to these expectations directly, and we must approximate them from data. There are two common regimes:

2.4.1 Case 1: When the desired signal $\boldsymbol{d}[n]$ is available (supervised / training mode)

In some applications we can obtain (at least occasionally) pairs of observations and ground truth,

$$(\boldsymbol{x}[n], \boldsymbol{d}[n]), \qquad n = 1, \ldots, N,$$

for example from calibration experiments, high-quality sensors, offline post-processing, or simulated data. In this setting, the Wiener filter can be learned by *empirical risk minimization*: we minimize the sample mean squared error

$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \frac{1}{N}\sum_{n=1}^{N} \left\|\boldsymbol{d}[n] - \boldsymbol{W}^T\boldsymbol{x}[n]\right\|_2^2. \tag{2.34}$$

This is simply multivariate least squares (linear regression) with inputs $\boldsymbol{x}[n]$ and targets $\boldsymbol{d}[n]$.

Define the data matrices

$$\boldsymbol{X} \triangleq [\boldsymbol{x}[1] \ \cdots \ \boldsymbol{x}[N]] \in \mathbb{R}^{p\times N}, \qquad \boldsymbol{D} \triangleq [\boldsymbol{d}[1] \ \cdots \ \boldsymbol{d}[N]] \in \mathbb{R}^{q\times N}.$$

Then (2.34) can be written as $\min_{\boldsymbol{W}} \|\boldsymbol{D} - \boldsymbol{W}^T\boldsymbol{X}\|_F^2$, whose normal equations yield

$$\boxed{\widehat{\boldsymbol{W}} = \left(\boldsymbol{X}\boldsymbol{X}^T\right)^{-1}\boldsymbol{X}\boldsymbol{D}^T} \qquad \text{(assuming } \boldsymbol{X}\boldsymbol{X}^T \text{ is invertible).} \tag{2.35}$$

Equivalently, introducing the sample correlation estimates

$$\widehat{\boldsymbol{R}}_{xx} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}[n]\boldsymbol{x}[n]^T = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^T, \qquad \widehat{\boldsymbol{R}}_{xd} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}[n]\boldsymbol{d}[n]^T = \frac{1}{N}\boldsymbol{X}\boldsymbol{D}^T,$$

we recover the familiar Wiener form

$$\boxed{\widehat{\boldsymbol{W}} = \widehat{\boldsymbol{R}}_{xx}^{-1}\widehat{\boldsymbol{R}}_{xd}.} \tag{2.36}$$

Online use after training. Once $\widehat{\boldsymbol{W}}$ is learned, deployment is fully online and extremely cheap: for each incoming observation $\boldsymbol{x}[n]$ we compute

$$\hat{\boldsymbol{d}}[n] = \widehat{\boldsymbol{W}}^T\boldsymbol{x}[n].$$

If the environment drifts, we may update $\widehat{\boldsymbol{W}}$ online using recursive least squares (RLS) or stochastic gradient descent on the instantaneous squared error $\|\boldsymbol{d}[n] - \boldsymbol{W}^T\boldsymbol{x}[n]\|_2^2$ whenever ground truth becomes available.

2.4.2 Case 2: When only correlations are known or can be estimated (unsupervised / model-based mode)

Often, the clean signal $\boldsymbol{d}[n]$ is never observed directly, so we cannot form $\widehat{\boldsymbol{R}}_{xd}$ from paired samples. Nevertheless, Wiener filtering remains usable if we know (or can estimate) the second-order statistics implied by the model.

A common situation is additive noise with $\boldsymbol{A} = \boldsymbol{I}$:

$$\boldsymbol{x}[n] = \boldsymbol{d}[n] + \boldsymbol{v}[n], \qquad \boldsymbol{v} \perp \boldsymbol{d}.$$

From the identities derived earlier,

$$\boldsymbol{R}_{xx} = \boldsymbol{R}_{dd} + \boldsymbol{R}_{vv}, \qquad \boldsymbol{R}_{xd} = \boldsymbol{R}_{dd}.$$

Thus the Wiener solution becomes

$$\boldsymbol{W}_\star = (\boldsymbol{R}_{dd} + \boldsymbol{R}_{vv})^{-1}\boldsymbol{R}_{dd}. \tag{2.37}$$

In the scalar LTI/WSS case, this corresponds in the frequency domain to the well-known PSD form

$$H(\omega) = \frac{S_{dd}(\omega)}{S_{dd}(\omega) + S_{vv}(\omega)}. \tag{2.38}$$

How do we obtain the needed statistics? In practice, one typically proceeds by one of the following routes:

- **Noise-only segments / calibration:** estimate $\boldsymbol{R}_{vv}$ (or $S_{vv}(\omega)$) from time intervals where the signal is absent.
- **Signal modeling:** assume a parametric prior for $\boldsymbol{d}$ (e.g., AR/ARMA) and estimate its parameters from data, which then determines $\boldsymbol{R}_{dd}$ (or $S_{dd}(\omega)$).
- **Domain knowledge:** use a known or expected PSD shape for the signal and/or noise (e.g., $1/f$ noise, band-limited signals).
- **Nonstationary / online adaptation:** estimate correlations over a sliding window so that $S_{vv}(\omega)$ or $S_{xx}(\omega)$ can adapt to changing conditions.

Online implementation viewpoint. Even when correlations are estimated, online operation typically has a two-stage structure: (i) update the second-order statistics (or their parametric models) from incoming data, and (ii) update or apply the filter to produce $\hat{\boldsymbol{d}}[n]$. In stationary settings, the filter can be designed once and then applied online. In slowly varying settings, one can re-estimate statistics and redesign the Wiener filter periodically (blockwise) or continuously (adaptive filtering).

Summary. If $\boldsymbol{d}[n]$ is available (even occasionally), the Wiener filter can be learned from data via least squares and then deployed online. If $\boldsymbol{d}[n]$ is not available, Wiener filtering is still possible provided that the relevant second-order statistics (correlations/PSDs) are known or can be estimated from the measurement stream under modeling assumptions.

2.5 Worked Examples

To make the Wiener filter concrete, we now work through small, low-dimensional examples. We use 2-dimensional signals so that every matrix and step is easy to inspect by hand.

2.5.1 Example 1: Pure Denoising with $\boldsymbol{A} = \boldsymbol{I}$ (Known Covariances)

Setup. Assume the observation model

$$\boldsymbol{x} = \boldsymbol{d} + \boldsymbol{v}, \qquad \mathbb{E}[\boldsymbol{d}] = \boldsymbol{0},\ \mathbb{E}[\boldsymbol{v}] = \boldsymbol{0},\ \boldsymbol{v} \perp \boldsymbol{d},$$

with

$$\boldsymbol{R}_{dd} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \boldsymbol{R}_{vv} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}.$$

Then

$$\boldsymbol{R}_{xx} = \boldsymbol{R}_{dd} + \boldsymbol{R}_{vv} = \begin{bmatrix} 3 & 1 \\ 1 & 5 \end{bmatrix}, \qquad \boldsymbol{R}_{xd} = \mathbb{E}[\boldsymbol{x}\boldsymbol{d}^T] = \boldsymbol{R}_{dd}.$$

Wiener solution. The Wiener matrix is

$$\boldsymbol{W}_\star = \boldsymbol{R}_{xx}^{-1}\boldsymbol{R}_{xd} = \begin{bmatrix} 3 & 1 \\ 1 & 5 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

Compute the inverse:

$$\det(\boldsymbol{R}_{xx}) = 3 \cdot 5 - 1 \cdot 1 = 14, \qquad \boldsymbol{R}_{xx}^{-1} = \frac{1}{14}\begin{bmatrix} 5 & -1 \\ -1 & 3 \end{bmatrix}.$$

Therefore,

$$\boldsymbol{W}_\star = \frac{1}{14}\begin{bmatrix} 5 & -1 \\ -1 & 3 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = \frac{1}{14}\begin{bmatrix} 9 & 3 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} \frac{9}{14} & \frac{3}{14} \\ \frac{1}{14} & \frac{5}{14} \end{bmatrix}.$$

Using the filter. Given a measured vector $\boldsymbol{x} \in \mathbb{R}^2$, the estimate is

$$\hat{\boldsymbol{d}} = \boldsymbol{W}_\star^T \boldsymbol{x}.$$

For instance, if $\boldsymbol{x} = [1 \;\; 2]^T$ then

$$\hat{\boldsymbol{d}} = \begin{bmatrix} \frac{9}{14} & \frac{1}{14} \\ \frac{3}{14} & \frac{5}{14} \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{11}{14} \\ \frac{13}{14} \end{bmatrix}.$$

Interpretation. The second component has larger noise variance (3 vs. 1), so the filter shrinks and mixes components asymmetrically. The estimate is not simply $\alpha\boldsymbol{x}$; it exploits correlation in $\boldsymbol{d}$ (off-diagonal terms of $\boldsymbol{R}_{dd}$) to "borrow strength" from the cleaner channel.

2.5.2 Example 2: Denoising When $\boldsymbol{d}[n]$ is Known During Training (Learning $\boldsymbol{W}$)

Setup. Assume we have N supervised training samples $\{(\boldsymbol{x}[n], \boldsymbol{d}[n])\}_{n=1}^N$, with $\boldsymbol{x}[n], \boldsymbol{d}[n] \in \mathbb{R}^2$. Define

$$\boldsymbol{X} = [\boldsymbol{x}[1] \;\cdots\; \boldsymbol{x}[N]] \in \mathbb{R}^{2\times N}, \qquad \boldsymbol{D} = [\boldsymbol{d}[1] \;\cdots\; \boldsymbol{d}[N]] \in \mathbb{R}^{2\times N}.$$

The least-squares estimate of the Wiener matrix is

$$\widehat{\boldsymbol{W}} = (\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{D}^T.$$

A tiny numeric example ($N = 3$). Consider the training set

$$\boldsymbol{x}[1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; \boldsymbol{d}[1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \boldsymbol{x}[2] = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; \boldsymbol{d}[2] = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \boldsymbol{x}[3] = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; \boldsymbol{d}[3] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Then

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

Compute

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

Since

$$(\boldsymbol{X}\boldsymbol{X}^T)^{-1} = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix},$$

we obtain

$$\widehat{\boldsymbol{W}} = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 3 & -1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{3} \\ 0 & \frac{2}{3} \end{bmatrix}.$$

Using the learned filter. For a new measurement $\boldsymbol{x}$, the estimate is $\hat{\boldsymbol{d}} = \widehat{\boldsymbol{W}}^T \boldsymbol{x}$.

2.5.3 Example 3: Deconvolution / Mixing with a Nontrivial $\boldsymbol{A}$

Setup. Assume a linear mixing ("blur") operator

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{d} + \boldsymbol{v}, \qquad \boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{R}_{dd} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{R}_{vv} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then

$$\boldsymbol{R}_{xd} = \boldsymbol{A}\boldsymbol{R}_{dd} = \boldsymbol{A}, \qquad \boldsymbol{R}_{xx} = \boldsymbol{A}\boldsymbol{R}_{dd}\boldsymbol{A}^T + \boldsymbol{R}_{vv} = \boldsymbol{A}\boldsymbol{A}^T + \boldsymbol{I}.$$

Compute

$$\boldsymbol{A}\boldsymbol{A}^T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \boldsymbol{R}_{xx} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}.$$

Invert:

$$\det(\boldsymbol{R}_{xx}) = 3 \cdot 2 - 1 \cdot 1 = 5, \qquad \boldsymbol{R}_{xx}^{-1} = \frac{1}{5} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}.$$

Thus

$$\boldsymbol{W}_\star = \boldsymbol{R}_{xx}^{-1}\boldsymbol{R}_{xd} = \frac{1}{5} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}.$$

Hence $\hat{\boldsymbol{d}} = \boldsymbol{W}_\star^T \boldsymbol{x}$ with

$$\boldsymbol{W}_\star^T = \frac{1}{5} \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}.$$

Interpretation. Because $\boldsymbol{A}$ mixes the two components, the optimal estimator must both (i) undo the mixing (like an inverse) and (ii) regularize against noise. The Wiener filter performs exactly this tradeoff automatically through the covariance terms.

2.6 M-tap FIR Wiener Filtering (Time-Domain Wiener–Hopf Design)

So far, we have considered a *memoryless* linear estimator of the form

$$\hat{\boldsymbol{d}}[n] = \boldsymbol{W}^T \boldsymbol{x}[n], \qquad \boldsymbol{x}[n] \in \mathbb{R}^p, \; \boldsymbol{d}[n] \in \mathbb{R}^q.$$

This is appropriate when $\boldsymbol{x}[n]$ contains all relevant information at time n. However, in many signal processing settings the desired estimate at time n should depend on a *window* of past observations, leading naturally to an M-tap causal FIR filter.

2.6.1 Stacked Data Vector and FIR Parameterization

To obtain an M-tap causal FIR estimator, we form an augmented data vector by stacking the current and past measurements:

$$\boldsymbol{x}_n \triangleq \begin{bmatrix} \boldsymbol{x}[n] \\ \boldsymbol{x}[n-1] \\ \vdots \\ \boldsymbol{x}[n-M+1] \end{bmatrix} \in \mathbb{R}^{pM}. \tag{2.39}$$

We then apply a linear estimator to $\boldsymbol{x}_n$:

$$\hat{\boldsymbol{d}}[n] = \boldsymbol{W}^T \boldsymbol{x}_n, \qquad \boldsymbol{W} \in \mathbb{R}^{(pM)\times q}. \tag{2.40}$$

Partition $\boldsymbol{W}$ into M blocks,

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_0 \\ \boldsymbol{W}_1 \\ \vdots \\ \boldsymbol{W}_{M-1} \end{bmatrix}, \qquad \boldsymbol{W}_k \in \mathbb{R}^{p\times q}.$$

Then (2.40) can be written as

$$\hat{\boldsymbol{d}}[n] = \begin{bmatrix} \boldsymbol{W}_0^T & \boldsymbol{W}_1^T & \cdots & \boldsymbol{W}_{M-1}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{x}[n] \\ \boldsymbol{x}[n-1] \\ \vdots \\ \boldsymbol{x}[n-M+1] \end{bmatrix}. \tag{2.41}$$

Equivalently, this is precisely a causal FIR convolution:

$$\boxed{\hat{\boldsymbol{d}}[n] = \sum_{k=0}^{M-1} \boldsymbol{W}_k^T \, \boldsymbol{x}[n-k].} \tag{2.42}$$

Thus, the usual LMMSE form $\hat{\boldsymbol{d}}[n] = \boldsymbol{W}^T \boldsymbol{x}_n$ becomes an M-tap FIR Wiener filter simply by choosing $\boldsymbol{x}_n$ to include M past samples.

2.6.2 Designing the FIR Wiener Filter from Second-Order Statistics

The optimal FIR Wiener filter is obtained by minimizing the MSE over the stacked estimator (2.40), i.e.,

$$\boldsymbol{W}_\star = \arg\min_{\boldsymbol{W}} \, \mathbb{E}\left[\left\| \boldsymbol{d}[n] - \boldsymbol{W}^T \boldsymbol{x}_n \right\|_2^2 \right].$$

The solution has the same algebraic form as before:

$$\boxed{\boldsymbol{W}_\star = \boldsymbol{R}_{x_n x_n}^{-1} \, \boldsymbol{R}_{x_n d},} \tag{2.43}$$

where

$$\boldsymbol{R}_{x_n x_n} \triangleq \mathbb{E}\{\boldsymbol{x}_n \boldsymbol{x}_n^T\} \in \mathbb{R}^{pM\times pM}, \qquad \boldsymbol{R}_{x_n d} \triangleq \mathbb{E}\{\boldsymbol{x}_n \boldsymbol{d}[n]^T\} \in \mathbb{R}^{pM\times q}.$$

In the wide-sense stationary (WSS) case, these matrices have a block-Toeplitz structure. Define the lag-k observation autocorrelation matrices

$$\boldsymbol{R}_{xx}[k] \triangleq \mathbb{E}\{\boldsymbol{x}[n]\boldsymbol{x}[n-k]^T\}, \qquad k \in \mathbb{Z},$$

and the cross-correlation matrices

$$\boldsymbol{R}_{xd}[k] \triangleq \mathbb{E}\{\boldsymbol{x}[n]\boldsymbol{d}[n-k]^T\}.$$

Then

$$\boldsymbol{R}_{x_n x_n} = \begin{bmatrix} \boldsymbol{R}_{xx}[0] & \boldsymbol{R}_{xx}[1] & \cdots & \boldsymbol{R}_{xx}[M-1] \\ \boldsymbol{R}_{xx}[-1] & \boldsymbol{R}_{xx}[0] & \cdots & \boldsymbol{R}_{xx}[M-2] \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{R}_{xx}[-M+1] & \boldsymbol{R}_{xx}[-M+2] & \cdots & \boldsymbol{R}_{xx}[0] \end{bmatrix} \in \mathbb{R}^{pM\times pM}, \tag{2.44}$$

and

$$\boldsymbol{R}_{x_n d} = \begin{bmatrix} \boldsymbol{R}_{xd}[0] \\ \boldsymbol{R}_{xd}[1] \\ \vdots \\ \boldsymbol{R}_{xd}[M-1] \end{bmatrix} \in \mathbb{R}^{pM\times q}. \tag{2.45}$$

For WSS $\boldsymbol{x}[n]$, we also have the symmetry relation

$$\boldsymbol{R}_{xx}[-k] = \boldsymbol{R}_{xx}[k]^T.$$

2.6.3 Practical Wiener Filtering on Data (FIR Case)

Suppose we are given measurements $\{\boldsymbol{x}[n]\}_{n=0}^{N-1}$ and either (i) a clean reference $\boldsymbol{d}[n]$ (supervised / calibration) or (ii) a model that allows us to form the required correlations (model-based).

Step 1: Choose the filter form. We select an FIR length M and use the stacked vector $\boldsymbol{x}_n$ as in (2.39).

Step 2: Estimate the required second-order statistics. A common unbiased sample estimate for the observation autocorrelation matrices is

$$\widehat{\boldsymbol{R}}_{xx}[k] = \frac{1}{N-k}\sum_{n=k}^{N-1} \boldsymbol{x}[n]\boldsymbol{x}[n-k]^T, \qquad k = 0,\dots,M-1. \tag{2.46}$$

If a clean reference exists, the cross-correlation may be estimated as

$$\widehat{\boldsymbol{R}}_{xd}[k] = \frac{1}{N-k}\sum_{n=k}^{N-1} \boldsymbol{x}[n]\boldsymbol{d}[n-k]^T, \qquad k = 0,\dots,M-1. \tag{2.47}$$

If $\boldsymbol{d}$ is not observed, one instead uses a model (e.g. $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{d} + \boldsymbol{v}$) together with estimates of signal/noise statistics to form $\widehat{\boldsymbol{R}}_{xx}[k]$ and $\widehat{\boldsymbol{R}}_{xd}[k]$.

Step 3: Build the Wiener–Hopf linear system. Using (2.44)–(2.45) with the estimated correlations, form

$$\widehat{\boldsymbol{R}}_{x_n x_n} \in \mathbb{R}^{pM\times pM}, \qquad \widehat{\boldsymbol{R}}_{x_n d} \in \mathbb{R}^{pM\times q}.$$

Step 4: Solve for the optimal tap matrix.

$$\boxed{\boldsymbol{W}^\star = \widehat{\boldsymbol{R}}_{x_n x_n}^{-1}\,\widehat{\boldsymbol{R}}_{x_n d}.} \tag{2.48}$$

(Equivalently, solve the linear system $\widehat{\boldsymbol{R}}_{x_n x_n}\boldsymbol{W}^\star = \widehat{\boldsymbol{R}}_{x_n d}$.)

Step 5: Filter the data. Finally, partition $\boldsymbol{W}^\star$ into taps $\boldsymbol{W}_0^\star, \dots, \boldsymbol{W}_{M-1}^\star$ and apply

$$\boxed{\hat{\boldsymbol{d}}[n] = \sum_{k=0}^{M-1} \boldsymbol{W}_k^{\star T} \boldsymbol{x}[n-k].} \tag{2.49}$$

Remark (causality and delay). The above construction yields a causal FIR filter that uses only current and past samples. If noncausal smoothing is allowed (e.g. offline processing), one can include future samples to obtain a two-sided Wiener smoother, often achieving lower MSE at the cost of delay.

2.6.4 Numeric Example (1D, $M = 2$): Supervised vs. Model-Based

We illustrate the complete Wiener–Hopf construction for a two-tap causal FIR filter

$$\hat{d}[n] = w_0 x[n] + w_1 x[n-1], \qquad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}.$$

For $M = 2$, the Wiener–Hopf system is

$$\underbrace{\begin{bmatrix} r_{xx}[0] & r_{xx}[1] \\ r_{xx}[1] & r_{xx}[0] \end{bmatrix}}_{\boldsymbol{R}} \boldsymbol{w} = \underbrace{\begin{bmatrix} r_{xd}[0] \\ r_{xd}[1] \end{bmatrix}}_{\boldsymbol{p}}, \qquad \boldsymbol{w} = \boldsymbol{R}^{-1}\boldsymbol{p}. \tag{2.50}$$

A) Supervised case (sample-based correlations). Assume we have $N = 5$ paired samples $\{x[n], d[n]\}$:

$$d = [1,\ 0,\ -1,\ 0,\ 1], \qquad v = [0.2,\ -0.1,\ 0.1,\ -0.2,\ 0], \qquad x = d+v = [1.2,\ -0.1,\ -0.9,\ -0.2,\ 1].$$

Step 1: compute $\hat{r}_{xx}[0]$.

$$\hat{r}_{xx}[0] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]^2 = \frac{1}{5}\Big(1.2^2 + (-0.1)^2 + (-0.9)^2 + (-0.2)^2 + 1^2\Big).$$

Compute each term:

$$1.2^2 = 1.44, \quad (-0.1)^2 = 0.01, \quad (-0.9)^2 = 0.81, \quad (-0.2)^2 = 0.04, \quad 1^2 = 1.$$

Sum:

$$1.44 + 0.01 + 0.81 + 0.04 + 1 = 3.30 \quad \Rightarrow \quad \boxed{\hat{r}_{xx}[0] = 3.30/5 = 0.66.}$$

Step 2: compute $\hat{r}_{xx}[1]$.

$$\hat{r}_{xx}[1] = \frac{1}{N-1}\sum_{n=1}^{N-1} x[n]x[n-1] = \frac{1}{4}\Big(x[1]x[0] + x[2]x[1] + x[3]x[2] + x[4]x[3]\Big).$$

Compute each product:

$$x[1]x[0] = (-0.1)(1.2) = -0.12, \quad x[2]x[1] = (-0.9)(-0.1) = 0.09,$$
$$x[3]x[2] = (-0.2)(-0.9) = 0.18, \quad x[4]x[3] = (1)(-0.2) = -0.20.$$

Sum:

$$-0.12 + 0.09 + 0.18 - 0.20 = -0.05 \quad \Rightarrow \quad \boxed{\hat{r}_{xx}[1] = -0.05/4 = -0.0125.}$$

Step 3: compute $\hat{r}_{xd}[0]$.

$$\hat{r}_{xd}[0] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]d[n] = \frac{1}{5}\Big(x[0]d[0] + x[1]d[1] + x[2]d[2] + x[3]d[3] + x[4]d[4]\Big).$$

Compute each product:

$$x[0]d[0] = 1.2 \cdot 1 = 1.2, \quad x[1]d[1] = (-0.1)\cdot 0 = 0, \quad x[2]d[2] = (-0.9)\cdot(-1) = 0.9,$$
$$x[3]d[3] = (-0.2)\cdot 0 = 0, \quad x[4]d[4] = (1)\cdot 1 = 1.$$

Sum:

$$1.2 + 0 + 0.9 + 0 + 1 = 3.1 \quad \Rightarrow \quad \boxed{\hat{r}_{xd}[0] = 3.1/5 = 0.62.}$$

Step 4: compute $\hat{r}_{xd}[1] = \mathbb{E}\{x[n]d[n-1]\}$.

$$\hat{r}_{xd}[1] = \frac{1}{N-1}\sum_{n=1}^{N-1} x[n]d[n-1] = \frac{1}{4}\Big(x[1]d[0] + x[2]d[1] + x[3]d[2] + x[4]d[3]\Big).$$

Compute each product:

$$x[1]d[0] = (-0.1)\cdot 1 = -0.1, \quad x[2]d[1] = (-0.9)\cdot 0 = 0,$$
$$x[3]d[2] = (-0.2)\cdot(-1) = 0.2, \quad x[4]d[3] = 1\cdot 0 = 0.$$

Sum:

$$-0.1 + 0 + 0.2 + 0 = 0.1 \quad \Rightarrow \quad \boxed{\hat{r}_{xd}[1] = 0.1/4 = 0.025.}$$

Step 5: build and solve the Wiener–Hopf system. Using (2.50),

$$\hat{\boldsymbol{R}} = \begin{bmatrix} 0.66 & -0.0125 \\ -0.0125 & 0.66 \end{bmatrix}, \qquad \hat{\boldsymbol{p}} = \begin{bmatrix} 0.62 \\ 0.025 \end{bmatrix}.$$

The inverse of a 2×2 matrix $\begin{bmatrix} a & b \\ b & a \end{bmatrix}$ is

$$\begin{bmatrix} a & b \\ b & a \end{bmatrix}^{-1} = \frac{1}{a^2 - b^2}\begin{bmatrix} a & -b \\ -b & a \end{bmatrix}.$$

Here $a = 0.66$, $b = -0.0125$, hence

$$a^2 - b^2 = 0.66^2 - (-0.0125)^2 = 0.4356 - 0.00015625 = 0.43544375.$$

Therefore,

$$\hat{\boldsymbol{R}}^{-1} = \frac{1}{0.43544375}\begin{bmatrix} 0.66 & 0.0125 \\ 0.0125 & 0.66 \end{bmatrix}.$$

Now compute $\hat{\boldsymbol{w}} = \hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{p}}$. First multiply:

$$\begin{bmatrix} 0.66 & 0.0125 \\ 0.0125 & 0.66 \end{bmatrix}\begin{bmatrix} 0.62 \\ 0.025 \end{bmatrix} = \begin{bmatrix} 0.66\cdot 0.62 + 0.0125\cdot 0.025 \\ 0.0125\cdot 0.62 + 0.66\cdot 0.025 \end{bmatrix} = \begin{bmatrix} 0.4092 + 0.0003125 \\ 0.00775 + 0.0165 \end{bmatrix} = \begin{bmatrix} 0.4095125 \\ 0.02425 \end{bmatrix}.$$

Then divide by 0.43544375:

$$\boxed{\hat{w}_0 = 0.4095125/0.43544375 \approx 0.9404, \qquad \hat{w}_1 = 0.02425/0.43544375 \approx 0.0557.}$$

Step 6: apply the filter on a sample. For example, at $n = 4$ (using $x[4] = 1$ and $x[3] = -0.2$),

$$\hat{d}[4] = \hat{w}_0 x[4] + \hat{w}_1 x[3] \approx 0.9404 \cdot 1 + 0.0557 \cdot (-0.2) \approx 0.9293.$$

B) Model-based case (AR(1) prior + white noise). Assume

$$x[n] = d[n] + v[n], \qquad v[n] \text{ white with } \mathrm{Var}(v) = \sigma_v^2, \qquad v \perp d,$$

and an AR(1) prior

$$d[n] = a\, d[n-1] + e[n], \qquad e[n] \sim \mathcal{N}(0, \sigma_e^2), \quad |a| < 1.$$

For AR(1), the stationary autocorrelations are

$$r_{dd}[0] = \frac{\sigma_e^2}{1-a^2}, \qquad r_{dd}[1] = a\, r_{dd}[0].$$

Using $x = d + v$ with white noise,

$$r_{xx}[0] = r_{dd}[0] + \sigma_v^2, \quad r_{xx}[1] = r_{dd}[1], \qquad r_{xd}[0] = r_{dd}[0], \quad r_{xd}[1] = r_{dd}[1].$$

Step 1: compute $r_{dd}[0], r_{dd}[1]$. Choose

$$a = 0.9, \qquad \sigma_e = 0.5 \Rightarrow \sigma_e^2 = 0.25, \qquad \sigma_v = 0.8 \Rightarrow \sigma_v^2 = 0.64.$$

Then

$$r_{dd}[0] = \frac{0.25}{1-0.9^2} = \frac{0.25}{1-0.81} = \frac{0.25}{0.19} \approx 1.315789,$$
$$r_{dd}[1] = 0.9 \cdot 1.315789 \approx 1.184210.$$

Step 2: compute $r_{xx}[0], r_{xx}[1], r_{xd}[0], r_{xd}[1]$.

$$r_{xx}[0] = r_{dd}[0] + \sigma_v^2 \approx 1.315789 + 0.64 = 1.955789, \qquad r_{xx}[1] = r_{dd}[1] \approx 1.184210,$$

$$r_{xd}[0] = r_{dd}[0] \approx 1.315789, \qquad r_{xd}[1] = r_{dd}[1] \approx 1.184210.$$

Step 3: build and solve the Wiener–Hopf system. Thus

$$\boldsymbol{R} = \begin{bmatrix} 1.955789 & 1.184210 \\ 1.184210 & 1.955789 \end{bmatrix}, \qquad \boldsymbol{p} = \begin{bmatrix} 1.315789 \\ 1.184210 \end{bmatrix}.$$

Again using the 2×2 inverse formula with $a = 1.955789$, $b = 1.184210$:

$$a^2 - b^2 \approx (1.955789)^2 - (1.184210)^2 \approx 3.825113 - 1.402357 = 2.422756.$$

So

$$\boldsymbol{R}^{-1} = \frac{1}{a^2 - b^2} \begin{bmatrix} a & -b \\ -b & a \end{bmatrix} \approx \frac{1}{2.422756} \begin{bmatrix} 1.955789 & -1.184210 \\ -1.184210 & 1.955789 \end{bmatrix}.$$

Compute $\boldsymbol{w} = \boldsymbol{R}^{-1}\boldsymbol{p}$. First multiply:

$$\begin{bmatrix} 1.955789 & -1.184210 \\ -1.184210 & 1.955789 \end{bmatrix} \begin{bmatrix} 1.315789 \\ 1.184210 \end{bmatrix} = \begin{bmatrix} 1.955789 \cdot 1.315789 - 1.184210 \cdot 1.184210 \\ -1.184210 \cdot 1.315789 + 1.955789 \cdot 1.184210 \end{bmatrix}.$$

Compute each product:

$$1.955789 \cdot 1.315789 \approx 2.573407, \qquad 1.184210 \cdot 1.184210 \approx 1.402357,$$

$$1.184210 \cdot 1.315789 \approx 1.557092, \qquad 1.955789 \cdot 1.184210 \approx 2.315552.$$

So the vector becomes

$$\begin{bmatrix} 2.573407 - 1.402357 \\ -1.557092 + 2.315552 \end{bmatrix} = \begin{bmatrix} 1.171050 \\ 0.758460 \end{bmatrix}.$$

Divide by 2.422756:

$$\boxed{w_0 \approx 1.171050/2.422756 \approx 0.4834, \qquad w_1 \approx 0.758460/2.422756 \approx 0.3129.}$$

Step 4: apply the filter on a sample. For instance, at time n, the estimate is

$$\hat{d}[n] = 0.4834\, x[n] + 0.3129\, x[n-1].$$

Interpretation. The supervised example above uses very few samples ($N = 5$), so the correlation estimates and therefore the taps $(\hat{w}_0, \hat{w}_1)$ are noisy. The model-based taps use exact second-order statistics from the assumed AR(1)+white-noise model, so they are "clean" and stable. In practice, with enough training data (or with regularization), the supervised approach approaches the model-based solution when the model is correct.

2.6.5 Python Implementation

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

def make_ar1(N, a, sigma_e, seed=0):
    """Generate AR(1): d[n] = a d[n-1] + e[n], e~N(0,sigma_e^2)."""
    rng = np.random.default_rng(seed)
    e = rng.normal(0.0, sigma_e, size=N)
    d = np.zeros(N, dtype=float)
    for n in range(1, N):
        d[n] = a * d[n-1] + e[n]
    return d

def wiener_m2_from_correlations(rxx0, rxx1, rxd0, rxd1):
    """Solve M=2 Wiener-Hopf system for w=[w0,w1]."""
    R = np.array([[rxx0, rxx1],
                  [rxx1, rxx0]], dtype=float)
    p = np.array([rxd0, rxd1], dtype=float)
    w = np.linalg.solve(R, p)
    return w  # [w0, w1]

def estimate_corrs_supervised(x, d):
    """Estimate rxx[0], rxx[1], rxd[0], rxd[1] from paired data."""
    N = len(x)
    rxx0 = np.mean(x * x)
    rxx1 = np.mean(x[1:] * x[:-1])  # lag 1
    rxd0 = np.mean(x * d)
    rxd1 = np.mean(x[1:] * d[:-1])  # E[x[n] d[n-1]]
    return rxx0, rxx1, rxd0, rxd1
```

```python

def apply_m2_filter(x, w0, w1):
    """Apply causal 2-tap FIR: d_hat[n] = w0 x[n] + w1 x[n-1]."""
    d_hat = np.zeros_like(x, dtype=float)
    d_hat[0] = w0 * x[0]   # assume x[-1]=0
    d_hat[1:] = w0 * x[1:] + w1 * x[:-1]
    return d_hat

# --------------------------
# 1) Generate synthetic data
# --------------------------
N = 800
a = 0.9
sigma_e = 0.5
sigma_v = 0.8

d = make_ar1(N, a=a, sigma_e=sigma_e, seed=1)
rng = np.random.default_rng(2)
v = rng.normal(0.0, sigma_v, size=N)
x = d + v  # denoising setting

# Split into training/test
N_train = N // 2
x_tr, d_tr = x[:N_train], d[:N_train]
x_te, d_te = x[N_train:], d[N_train:]

# -----------------------------------------
# 2) Supervised M=2 Wiener filter (training)
# -----------------------------------------
rxx0_hat, rxx1_hat, rxd0_hat, rxd1_hat = estimate_corrs_supervised(x_tr
    , d_tr)
w_sup = wiener_m2_from_correlations(rxx0_hat, rxx1_hat, rxd0_hat,
   rxd1_hat)
w0_sup, w1_sup = w_sup

# -----------------------------------------
# 3) Model-based M=2 Wiener filter (AR(1))
# -----------------------------------------
# For AR(1): r_dd[0] = sigma_e^2 / (1-a^2), r_dd[1] = a r_dd[0]
rdd0 = (sigma_e**2) / (1.0 - a**2)
rdd1 = a * rdd0

# For x=d+v with white noise var sigma_v^2:
rxx0_model = rdd0 + sigma_v**2
rxx1_model = rdd1
rxd0_model = rdd0
rxd1_model = rdd1

w_mod = wiener_m2_from_correlations(rxx0_model, rxx1_model, rxd0_model,
    rxd1_model)
w0_mod, w1_mod = w_mod

# --------------------------
# 4) Apply filters on test
# --------------------------
dhat_sup = apply_m2_filter(x_te, w0_sup, w1_sup)
dhat_mod = apply_m2_filter(x_te, w0_mod, w1_mod)

```

```python
# ---------------------------
# 5) Evaluate and plot
# ---------------------------
mse_noisy = np.mean((x_te - d_te)**2)
mse_sup = np.mean((dhat_sup - d_te)**2)
mse_mod = np.mean((dhat_mod - d_te)**2)

print("Supervised Wiener (M=2) taps:  w0 = %.4f, w1 = %.4f" % (w0_sup,
    w1_sup))
print("Model-based Wiener (M=2) taps: w0 = %.4f, w1 = %.4f" % (w0_mod,
    w1_mod))
print("Test MSE: noisy x vs d:         %.4f" % mse_noisy)
print("Test MSE: supervised estimate: %.4f" % mse_sup)
print("Test MSE: model-based estimate:%.4f" % mse_mod)

# Plot a short window for readability
L = min(250, N - N_train)
t = np.arange(L)

plt.figure()
plt.plot(t, d_te[:L], label="true d[n]")
plt.plot(t, x_te[:L], label="observed x[n]")
plt.plot(t, dhat_sup[:L], label="Wiener M=2 (supervised)")
plt.plot(t, dhat_mod[:L], label="Wiener M=2 (model-based)")
plt.xlabel("n (test segment)")
plt.ylabel("amplitude")
plt.title("2-tap FIR Wiener filtering (supervised vs model-based)")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1: Wiener Filter implementation

This simulation implements a complete end-to-end example of Wiener filtering for a scalar (1D) signal using a two-tap ($M = 2$) causal FIR estimator. We now describe each step of the experiment in mathematical terms.

Signal and noise model. The clean signal $\{d[n]\}$ is generated as a stationary autoregressive process of order one:

$$d[n] = a\, d[n-1] + e[n], \qquad e[n] \sim \mathcal{N}(0, \sigma_e^2), \qquad |a| < 1. \tag{2.51}$$

This defines a zero-mean, wide-sense stationary random process with autocorrelation

$$r_{dd}[0] = \frac{\sigma_e^2}{1-a^2}, \qquad r_{dd}[1] = a\, r_{dd}[0]. \tag{2.52}$$

The observed signal is obtained through additive noise:

$$x[n] = d[n] + v[n], \qquad v[n] \sim \mathcal{N}(0, \sigma_v^2), \qquad v \perp d. \tag{2.53}$$

The goal is to estimate $d[n]$ from $x[n]$ using a causal FIR Wiener filter.

Estimator structure ($M = 2$). The estimator is restricted to the class of two-tap causal FIR filters:

$$\hat{d}[n] = w_0\, x[n] + w_1\, x[n-1], \tag{2.54}$$

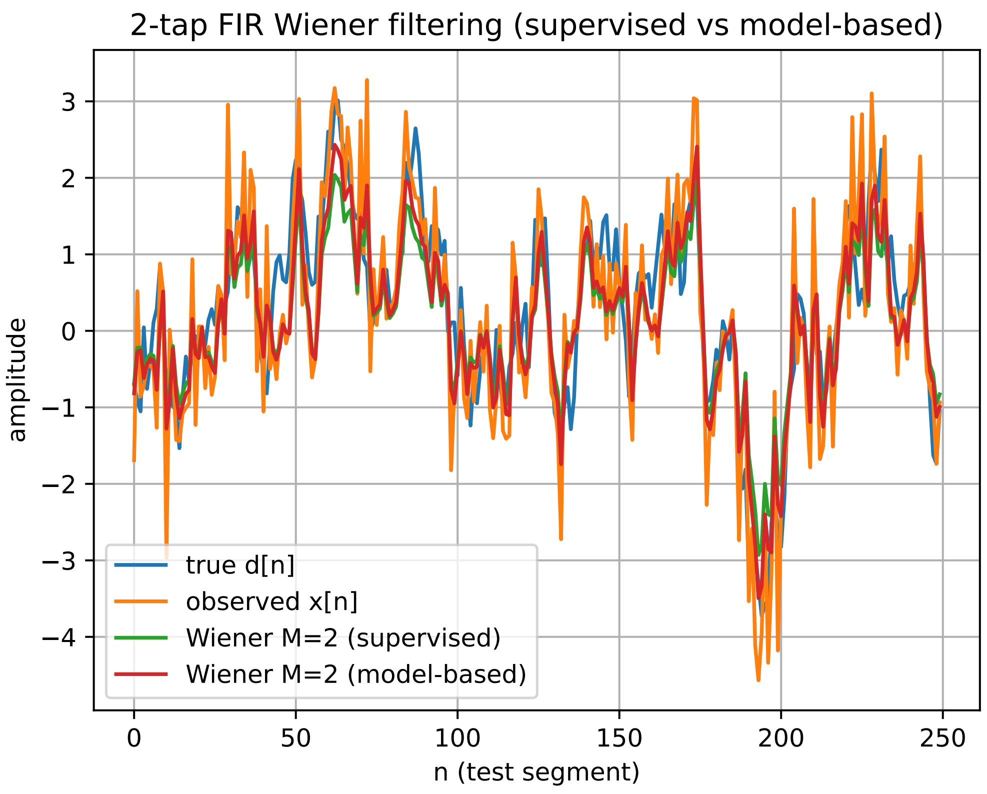

Figure 1: **Two-tap FIR Wiener filtering ($M = 2$) in 1D.** We show the clean signal $d[n]$, the noisy observation $x[n] = d[n] + v[n]$, and two Wiener estimates $\hat{d}[n] = w_0 x[n] + w_1 x[n-1]$. The *supervised* filter learns (w_0, w_1) from paired training data $(x[n], d[n])$ via sample correlations, while the *model-based* filter computes (w_0, w_1) from assumed second-order statistics (e.g., AR(1) prior for $d[n]$ and known noise variance).

or, equivalently,

$$\hat{d}[n] = \boldsymbol{w}^T \boldsymbol{x}_n, \qquad \boldsymbol{x}_n = \begin{bmatrix} x[n] \\ x[n-1] \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}.$$

Optimality criterion. The coefficients (w_0, w_1) are chosen to minimize the mean squared error

$$\mathbb{E}\left[(d[n] - \hat{d}[n])^2\right].$$

For $M = 2$, the Wiener–Hopf equations reduce to

$$\begin{bmatrix} r_{xx}[0] & r_{xx}[1] \\ r_{xx}[1] & r_{xx}[0] \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} r_{xd}[0] \\ r_{xd}[1] \end{bmatrix}, \tag{2.55}$$

where

$$r_{xx}[k] = \mathbb{E}\{x[n]x[n-k]\}, \qquad r_{xd}[k] = \mathbb{E}\{x[n]d[n-k]\}.$$

Supervised correlation estimation. In the supervised part of the experiment, paired samples $\{x[n], d[n]\}$ are available over a training interval. The expectations in (2.55) are approximated by sample averages:

$$\hat{r}_{xx}[0] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]^2, \tag{2.56}$$

$$\hat{r}_{xx}[1] = \frac{1}{N-1}\sum_{n=1}^{N-1} x[n]x[n-1], \tag{2.57}$$

$$\hat{r}_{xd}[0] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]d[n], \tag{2.58}$$

$$\hat{r}_{xd}[1] = \frac{1}{N-1}\sum_{n=1}^{N-1} x[n]d[n-1]. \tag{2.59}$$

These empirical correlations are inserted into (2.55), yielding a data-driven estimate $\hat{\boldsymbol{w}}_{\text{sup}}$. This corresponds to solving a least-squares problem over the chosen FIR structure.

Model-based correlation computation. In the model-based part of the experiment, the clean signal $d[n]$ is *not* used directly. Instead, the assumed probabilistic model (2.51)–(2.53) is used to compute correlations analytically. Using $x = d+v$ with $v \perp d$, we obtain

$$r_{xx}[0] = r_{dd}[0] + \sigma_v^2, \qquad r_{xx}[1] = r_{dd}[1], \tag{2.60}$$

and

$$r_{xd}[0] = r_{dd}[0], \qquad r_{xd}[1] = r_{dd}[1]. \tag{2.61}$$

Substituting the AR(1) autocorrelations from (2.52) produces a closed-form Wiener–Hopf system whose solution $\boldsymbol{w}_{\text{model}}$ is the optimal two-tap LMMSE estimator under the assumed statistics.

Filtering and evaluation. Both filters are applied to a disjoint test segment using (2.54). Performance is quantified by the empirical mean squared error

$$\text{MSE} = \frac{1}{N_{\text{test}}}\sum_{n} \big(d[n] - \hat{d}[n]\big)^2,$$

and compared against the baseline MSE of the noisy observation $x[n]$.

Key interpretation. The supervised Wiener filter approximates the optimal solution using finite-sample estimates of second-order statistics, and is therefore subject to estimation variance. The model-based Wiener filter uses exact (oracle) second-order statistics derived from the generating model, and is therefore closer to the true LMMSE solution when the model is correct. As the amount of training data increases, the supervised solution converges to the model-based one; conversely, under model mismatch, the supervised approach can outperform the model-based design.

3 Kalman Filtering

3.1 Motivation: Why Kalman Filtering?

In the previous lecture we studied *Wiener filtering*, which provides an optimal linear estimator (LMMSE/MMSE under Gaussianity) for *stationary* settings, often interpreted in a batch or frequency-domain viewpoint. In many applications, however, the quantity we wish to estimate is inherently *dynamic*: it evolves over time, and measurements arrive sequentially. Typical examples include position and velocity in navigation, slowly varying channel gains in communications, or temperature in a thermal system.

This motivates the need for an estimator with three properties. First, it should be **recursive** (online): rather than recomputing an estimate from scratch when new data arrive, it should update the current estimate efficiently using the new measurement. Second, it should be **model-based**: it should exploit knowledge of how the state evolves over time via a dynamical model. Third, it should be **uncertainty-aware**: it should track not only a point estimate but also its confidence, typically represented by a covariance matrix that quantifies estimation error.

The Kalman filter addresses precisely this need. For linear state-space models with Gaussian noise, it yields the **optimal linear MMSE** estimator and, under Gaussian assumptions, coincides with the exact Bayesian posterior mean. Its conceptual structure is simple and powerful: *predict* the state forward using the model, then *correct* that prediction using the new measurement.

3.2 State-Space Modeling

Many dynamical systems can be described through a **latent (hidden) state** that evolves over time and generates noisy observations. In a generic (possibly nonlinear) form, this is written as

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1},\, \boldsymbol{u}_{k-1},\, k-1) + \boldsymbol{w}_{k-1}, \tag{3.1}$$

$$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k,\, k) + \boldsymbol{v}_k. \tag{3.2}$$

Here,

- $\boldsymbol{x}_k \in \mathbb{R}^n$ is the latent/hidden state (the quantity to be estimated),
- $\boldsymbol{y}_k \in \mathbb{R}^m$ is the measurement (observed data),
- $\boldsymbol{u}_k \in \mathbb{R}^r$ is a known input/control (may be absent),
- $\boldsymbol{w}_{k-1}$ is the process noise capturing model uncertainty and unmodeled effects,
- $\boldsymbol{v}_k$ is the measurement noise capturing sensor uncertainty.

The Bayesian filtering objective is:

$$\text{infer } \boldsymbol{x}_k \text{ from } \boldsymbol{y}_{1:k} \triangleq \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}.$$

Equivalently, one seeks the posterior distribution $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k})$, and typically a point estimate such as the posterior mean.

3.3 Kalman Filter: Linear–Gaussian Assumptions

The Kalman filter is optimal under the **linear–Gaussian** special case of (3.1)–(3.2). The state evolves linearly:

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1}, \tag{3.3}$$

and measurements are linear functions of the current state:

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k. \tag{3.4}$$

The noise terms are assumed to be zero-mean, white, and Gaussian:

$$\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k), \qquad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k), \tag{3.5}$$

and mutually independent across time and between processes, i.e.

$$\mathbb{E}[\boldsymbol{w}_k \boldsymbol{v}_j^T] = \boldsymbol{0}, \quad \forall k, j, \qquad \text{(and similarly across time indices).} \tag{3.6}$$

Finally, the initial condition is Gaussian:

$$\boldsymbol{x}_0 \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \boldsymbol{P}_0). \tag{3.7}$$

Key consequence (closure of Gaussians under linear transformations). Under (3.3)–(3.7), the predictive distribution $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1})$ is Gaussian if $p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1})$ is Gaussian. Then, incorporating the linear Gaussian measurement model (3.4) preserves Gaussianity of the posterior. Hence the filtering distribution remains Gaussian for all k:

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) = \mathcal{N}\big(\hat{\boldsymbol{x}}_{k|k},\, \boldsymbol{P}_{k|k}\big)\,,$$

and the Kalman filter provides recursive update equations for the posterior mean $\hat{\boldsymbol{x}}_{k|k}$ and covariance $\boldsymbol{P}_{k|k}$.

3.4 Kalman Filtering as Recursive Bayesian Estimation

The goal of Kalman filtering is to compute the posterior distribution of the hidden state at time step k given all measurements up to that time:

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}).$$

Rather than recomputing this posterior from scratch at each time step, we exploit the temporal structure of the problem to derive a *recursive Bayesian update.*

3.4.1 Bayesian Filtering Recursion

Bayesian filtering decomposes inference into two conceptually distinct steps: a **prediction** (time update) and a **correction** (measurement update).

Prediction (time update). Given the posterior at time $k-1$, we propagate uncertainty through the dynamics:

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})\, p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1})\, d\boldsymbol{x}_{k-1}. \tag{3.8}$$

Update (measurement update). Once a new measurement $\boldsymbol{y}_k$ is available, we incorporate it using Bayes' rule:

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) \propto p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)\, p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}). \tag{3.9}$$

The prediction step propagates the previous posterior forward in time, while the update step corrects this prediction using new information. For linear dynamics and Gaussian noise, both steps preserve Gaussianity, allowing the posterior to be summarized entirely by a mean and a covariance.

3.4.2 Kalman Filter: Prediction Step

Assume that at time $k-1$ the posterior is Gaussian:

$$p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}) = \mathcal{N}(\hat{\boldsymbol{x}}_{k-1}, \, \boldsymbol{\Sigma}_{k-1}) \, .$$

The linear state-space model is

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1}, \qquad \boldsymbol{w}_{k-1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_{k-1}), \tag{3.10}$$

where the process noise covariance $\boldsymbol{Q}_{k-1}$ may vary with time.

Applying (3.8) yields the predicted (prior) distribution:

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}) = \mathcal{N}\big(\hat{\boldsymbol{x}}_k^-, \, \boldsymbol{\Sigma}_k^-\big) \, .$$

The predicted mean is

$$\boxed{\hat{\boldsymbol{x}}_k^- = \mathbb{E}[\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}] = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1},} \tag{3.11}$$

and the predicted covariance is

$$\boxed{\boldsymbol{\Sigma}_k^- = \mathrm{Cov}[\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}] = \boldsymbol{A}\boldsymbol{\Sigma}_{k-1}\boldsymbol{A}^\top + \boldsymbol{Q}_{k-1}.} \tag{3.12}$$

The covariance increases due to process noise, reflecting growing uncertainty when the system evolves without measurement updates.

3.4.3 Kalman Filter: Update Step

After prediction, we incorporate the new measurement using the linear observation model

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k, \qquad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k), \tag{3.13}$$

where $\boldsymbol{R}_k$ is the (possibly time-varying) measurement noise covariance.

Define the **innovation** (measurement residual)

$$\boxed{\boldsymbol{r}_k = \boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_k^-,} \tag{3.14}$$

and its covariance

$$\boxed{\boldsymbol{S}_k = \boldsymbol{C}\boldsymbol{\Sigma}_k^-\boldsymbol{C}^\top + \boldsymbol{R}_k.} \tag{3.15}$$

The **Kalman gain** is then

$$\boxed{\boldsymbol{K}_k = \boldsymbol{\Sigma}_k^-\boldsymbol{C}^\top\boldsymbol{S}_k^{-1}.} \tag{3.16}$$

The posterior mean and covariance become

$$\boxed{\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\boldsymbol{r}_k,} \tag{3.17}$$

$$\boxed{\boldsymbol{\Sigma}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\,\boldsymbol{\Sigma}_k^-.} \tag{3.18}$$

The Kalman gain balances trust between the model prediction (encoded in $\boldsymbol{\Sigma}_k^-$) and the measurement (encoded in $\boldsymbol{R}_k$).

3.5 Bayes Filtering: The General Framework

Bayes filtering is the general mathematical framework for *recursive* state estimation in dynamical systems. The key idea is to maintain a probability distribution over the current state (the *belief*) and update it sequentially as new controls and measurements arrive.

3.5.1 State, Controls, Measurements, and Belief

Let

$$\boldsymbol{x}_k \in \mathbb{R}^n \quad \text{(state)}, \qquad \boldsymbol{u}_k \in \mathbb{R}^r \quad \text{(control/input)}, \qquad \boldsymbol{y}_k \in \mathbb{R}^m \quad \text{(measurement)}.$$

We denote the measurement history by $\boldsymbol{y}_{1:k} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}$ and the control history by $\boldsymbol{u}_{1:k} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.

The **belief** at time k is defined as the filtering distribution

$$\mathrm{bel}(\boldsymbol{x}_k) \triangleq p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}, \boldsymbol{u}_{1:k}), \tag{3.19}$$

i.e., the posterior distribution over the current state given all information available up to time k.

3.5.2 Probabilistic Assumptions (State-Space Model)

Bayes filtering relies on two key conditional-independence assumptions.

1) First-order Markov dynamics (motion model). The state at time k depends on the past only through the previous state and the current control:

$$p(\boldsymbol{x}_k \mid \boldsymbol{x}_{0:k-1}, \boldsymbol{u}_{1:k}) = p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k). \tag{3.20}$$

The conditional density $p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$ is called the **motion model** (or transition model).

2) Conditional independence of measurements (observation model). The measurement at time k depends on the past only through the current state:

$$p(\boldsymbol{y}_k \mid \boldsymbol{x}_{0:k}, \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}) = p(\boldsymbol{y}_k \mid \boldsymbol{x}_k). \tag{3.21}$$

The conditional density $p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)$ is called the **observation model** (or measurement likelihood).

These assumptions define a hidden Markov model (HMM) with inputs.

3.5.3 Deriving the Bayes Filter Recursion

Starting from the belief definition (3.19), we apply Bayes' rule:

$$\begin{aligned} \mathrm{bel}(\boldsymbol{x}_k) &= p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}, \boldsymbol{u}_{1:k}) \\ &= \frac{p(\boldsymbol{y}_k \mid \boldsymbol{x}_k, \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})\, p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})}{p(\boldsymbol{y}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})}. \end{aligned} \tag{3.22}$$

Introduce the normalization constant

$$\eta \triangleq \frac{1}{p(\boldsymbol{y}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})}, \tag{3.23}$$

so that

$$\mathrm{bel}(\boldsymbol{x}_k) = \eta\, p(\boldsymbol{y}_k \mid \boldsymbol{x}_k, \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})\, p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}). \tag{3.24}$$

Using the observation Markov assumption (3.21),

$$p(\boldsymbol{y}_k \mid \boldsymbol{x}_k, \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}) = p(\boldsymbol{y}_k \mid \boldsymbol{x}_k),$$

we obtain

$$\mathrm{bel}(\boldsymbol{x}_k) = \eta\, p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)\, p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}). \tag{3.25}$$

3.5.4 Prediction via the Law of Total Probability

The predictive distribution $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})$ is obtained by marginalizing over the previous state:

$$\begin{aligned} p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}) &= \int p(\boldsymbol{x}_k, \boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})\, d\boldsymbol{x}_{k-1} \\ &= \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})\, p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k})\, d\boldsymbol{x}_{k-1}. \end{aligned} \tag{3.26}$$

Using the motion-model Markov assumption (3.20) and recognizing

$$p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}) = p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k-1}) = \mathrm{bel}(\boldsymbol{x}_{k-1}),$$

we arrive at

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{u}_{1:k}) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\, \mathrm{bel}(\boldsymbol{x}_{k-1})\, d\boldsymbol{x}_{k-1}. \tag{3.27}$$

Substituting (3.27) into (3.25) yields the Bayes filter recursion:

$$\boxed{\mathrm{bel}(\boldsymbol{x}_k) = \eta\, p(\boldsymbol{y}_k \mid \boldsymbol{x}_k) \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\, \mathrm{bel}(\boldsymbol{x}_{k-1})\, d\boldsymbol{x}_{k-1}.} \tag{3.28}$$

3.5.5 Bayes Filtering as Two Steps

The recursion (3.28) naturally decomposes into two steps.

Prediction (prior / time update).

$$\boxed{\overline{\mathrm{bel}}(\boldsymbol{x}_k) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\, \mathrm{bel}(\boldsymbol{x}_{k-1})\, d\boldsymbol{x}_{k-1}.} \tag{3.29}$$

Correction (posterior / measurement update).

$$\boxed{\mathrm{bel}(\boldsymbol{x}_k) = \eta\, p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)\, \overline{\mathrm{bel}}(\boldsymbol{x}_k),} \tag{3.30}$$

with η chosen such that $\int \mathrm{bel}(\boldsymbol{x}_k)\, d\boldsymbol{x}_k = 1$.

3.5.6 Kalman Filtering as a Special Case

Bayes filtering is exact but generally intractable because the belief $\mathrm{bel}(\boldsymbol{x}_k)$ is an arbitrary distribution over $\mathbb{R}^n$. Kalman filtering arises when:

- the motion model is linear and Gaussian:

$$\boldsymbol{x}_k = \boldsymbol{A}_k\boldsymbol{x}_{k-1} + \boldsymbol{B}_k\boldsymbol{u}_k + \boldsymbol{w}_k, \qquad \boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k),$$

- the observation model is linear and Gaussian:

$$\boldsymbol{y}_k = \boldsymbol{C}_k\boldsymbol{x}_k + \boldsymbol{v}_k, \qquad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k).$$

Under these assumptions, the belief remains Gaussian for all k, and the Bayes filter recursion (3.29)–(3.30) reduces to the Kalman filter update equations for the mean and covariance.

3.6 Kalman Filter as Recursive Wiener Filtering

A useful way to interpret the Kalman filter is as a *time-varying* (recursive) version of Wiener/LMMSE estimation. In Wiener filtering we form an optimal linear estimator from second-order statistics, typically in a batch setting. In Kalman filtering we solve a closely related LMMSE problem at *each time step* using the *current* predicted covariance, leading to a gain that changes over time.

3.6.1 At Time k: A One-Step LMMSE Problem

Consider the linear–Gaussian measurement model at time k:

$$\boldsymbol{y}_k = \boldsymbol{C}_k\boldsymbol{x}_k + \boldsymbol{v}_k, \qquad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k). \tag{3.31}$$

Assume we already have a *prior* (predicted) distribution from the time update:

$$\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1} \sim \mathcal{N}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{\Sigma}_k^-). \tag{3.32}$$

Define the estimation error and consider estimators of the affine form

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\big(\boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-\big), \tag{3.33}$$

where $\boldsymbol{K}_k$ is a matrix to be chosen. This is exactly the Kalman update form: start from the prior mean and add a correction proportional to the innovation.

The key point is: (3.33) can be obtained by solving an LMMSE problem. Let

$$\tilde{\boldsymbol{x}}_k \triangleq \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-, \qquad \tilde{\boldsymbol{y}}_k \triangleq \boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-.$$

From (3.31)–(3.32),

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{C}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{v}_k, \qquad \mathbb{E}[\tilde{\boldsymbol{x}}_k] = \boldsymbol{0}, \quad \mathrm{Cov}(\tilde{\boldsymbol{x}}_k) = \boldsymbol{\Sigma}_k^-. \tag{3.34}$$

Now restrict to linear estimators of $\tilde{\boldsymbol{x}}_k$ from $\tilde{\boldsymbol{y}}_k$:

$$\widehat{\tilde{\boldsymbol{x}}}_k = \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k.$$

Choose $\boldsymbol{K}_k$ to minimize the conditional MSE

$$\boldsymbol{K}_k^\star = \arg\min_{\boldsymbol{K}} \mathbb{E}\left[\|\tilde{\boldsymbol{x}}_k - \boldsymbol{K}\tilde{\boldsymbol{y}}_k\|_2^2 \mid \boldsymbol{y}_{1:k-1}\right]. \tag{3.35}$$

This is a standard Wiener/LMMSE problem with "desired signal" $\tilde{\boldsymbol{x}}_k$ and "observation" $\tilde{\boldsymbol{y}}_k$.

3.6.2 Wiener Solution ⇒ Kalman Gain

The Wiener/LMMSE solution has the form

$$\boxed{\boldsymbol{K}_k^\star = \boldsymbol{R}_{\tilde{x}\tilde{y}}\,\boldsymbol{R}_{\tilde{y}\tilde{y}}^{-1},} \tag{3.36}$$

where (conditioning on $\boldsymbol{y}_{1:k-1}$ is implicit)

$$\boldsymbol{R}_{\tilde{x}\tilde{y}} \triangleq \mathbb{E}\{\tilde{\boldsymbol{x}}_k\tilde{\boldsymbol{y}}_k^T\}, \qquad \boldsymbol{R}_{\tilde{y}\tilde{y}} \triangleq \mathbb{E}\{\tilde{\boldsymbol{y}}_k\tilde{\boldsymbol{y}}_k^T\}.$$

Using (3.34) and independence of $\tilde{\boldsymbol{x}}_k$ and $\boldsymbol{v}_k$:

$$\boldsymbol{R}_{\tilde{x}\tilde{y}} = \mathbb{E}\{\tilde{\boldsymbol{x}}_k(\boldsymbol{C}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{v}_k)^T\} = \mathbb{E}\{\tilde{\boldsymbol{x}}_k\tilde{\boldsymbol{x}}_k^T\}\boldsymbol{C}_k^T = \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T,$$

and

$$\begin{aligned}\boldsymbol{R}_{\tilde{y}\tilde{y}} &= \mathbb{E}\{(\boldsymbol{C}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{v}_k)(\boldsymbol{C}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{v}_k)^T\}\\ &= \boldsymbol{C}_k\mathbb{E}\{\tilde{\boldsymbol{x}}_k\tilde{\boldsymbol{x}}_k^T\}\boldsymbol{C}_k^T + \mathbb{E}\{\boldsymbol{v}_k\boldsymbol{v}_k^T\}\\ &= \boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T + \boldsymbol{R}_k.\end{aligned}$$

Substituting into (3.36) yields

$$\boxed{\boldsymbol{K}_k = \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T\left(\boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T + \boldsymbol{R}_k\right)^{-1},} \tag{3.37}$$

which is exactly the Kalman gain. Defining the innovation covariance

$$\boldsymbol{S}_k \triangleq \boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T + \boldsymbol{R}_k,$$

we can write $\boldsymbol{K}_k = \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T\boldsymbol{S}_k^{-1}$.

Finally, returning from centered variables to the original ones,

$$\widehat{\tilde{\boldsymbol{x}}}_k = \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-) \quad \Rightarrow \quad \hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-),$$

which matches the usual Kalman measurement update.

3.6.3 Interpretation: "Recursive" Wiener Filtering

Equation (3.37) shows that the Kalman gain is a *Wiener/LMMSE filter*, but computed *online* using time-varying statistics:

- In Wiener filtering, the gain depends on (typically constant) second-order statistics such as $\boldsymbol{R}_{xx}$ and $\boldsymbol{R}_{xd}$.
- In Kalman filtering, the corresponding statistics are *conditional* on past data and evolve with time through the prediction step, via $\boldsymbol{\Sigma}_k^-$.

What makes it recursive? The key recursion is that $\boldsymbol{\Sigma}_k^-$ (the prior covariance) is produced by propagating the previous posterior covariance through the dynamics:

$$\boldsymbol{\Sigma}_k^- = \boldsymbol{A}_k\boldsymbol{\Sigma}_{k-1}\boldsymbol{A}_k^T + \boldsymbol{Q}_{k-1},$$

so the "Wiener statistics" used to compute $\boldsymbol{K}_k$ are updated at every time step.

What makes it time-varying? Even if $\boldsymbol{A}_k, \boldsymbol{C}_k$ are constant, the conditional covariance $\boldsymbol{\Sigma}_k^-$ changes over time (especially during transients), and the noise covariances $\boldsymbol{Q}_{k-1}, \boldsymbol{R}_k$ may also vary with time. Therefore, the gain $\boldsymbol{K}_k$ is typically *time-varying*, unlike the classical stationary Wiener filter.

Takeaway. Kalman filtering can be viewed as applying a Wiener/LMMSE design at each time step:

$$\text{“Kalman gain”} \quad = \quad \text{“Wiener filter” computed from current conditional covariances.}$$

Under linear–Gaussian assumptions, this recursive Wiener viewpoint is exactly equivalent to the Bayesian posterior-mean estimator.

3.7 Worked Kalman Filter Examples (1D and 2D)

In this section we work through concrete Kalman-filter updates by hand. We emphasize the two steps at each time k:

$$\textbf{Predict: } (\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{\Sigma}_{k-1}) \mapsto (\hat{\boldsymbol{x}}_k^-, \boldsymbol{\Sigma}_k^-), \qquad \textbf{Update: } (\hat{\boldsymbol{x}}_k^-, \boldsymbol{\Sigma}_k^-, \boldsymbol{y}_k) \mapsto (\hat{\boldsymbol{x}}_k, \boldsymbol{\Sigma}_k).$$

We allow time-varying noise covariances $\boldsymbol{Q}_{k-1}$ and $\boldsymbol{R}_k$.

3.7.1 Example 1 (1D): Random Walk with Noisy Measurements

Model. We estimate a scalar state $x_k \in \mathbb{R}$ from scalar measurements $y_k \in \mathbb{R}$ using

$$x_k = x_{k-1} + w_{k-1}, \qquad w_{k-1} \sim \mathcal{N}(0, Q_{k-1}), \tag{3.38}$$

$$y_k = x_k + v_k, \qquad v_k \sim \mathcal{N}(0, R_k). \tag{3.39}$$

Thus $A = 1$, $C = 1$, and there is no control input.

Kalman filter equations (scalar form). Prediction:

$$\hat{x}_k^- = \hat{x}_{k-1}, \qquad \Sigma_k^- = \Sigma_{k-1} + Q_{k-1}. \tag{3.40}$$

Update:

$$S_k = \Sigma_k^- + R_k, \qquad K_k = \frac{\Sigma_k^-}{S_k}, \tag{3.41}$$

$$\hat{x}_k = \hat{x}_k^- + K_k\,(y_k - \hat{x}_k^-), \qquad \Sigma_k = (1 - K_k)\Sigma_k^-. \tag{3.42}$$

Numerical values. Assume prior

$$\hat{x}_0 = 0, \qquad \Sigma_0 = 1.$$

Let the noise variances vary with time:

$$Q_0 = 0.10,\ R_1 = 0.40, \qquad Q_1 = 0.20,\ R_2 = 0.10.$$

Suppose we receive measurements

$$y_1 = 1.20, \qquad y_2 = 0.90.$$

Step $k = 1$. Prediction:

$$\hat{x}_1^- = \hat{x}_0 = 0, \qquad \Sigma_1^- = \Sigma_0 + Q_0 = 1 + 0.10 = 1.10.$$

Innovation and gain:

$$S_1 = \Sigma_1^- + R_1 = 1.10 + 0.40 = 1.50, \qquad K_1 = \frac{\Sigma_1^-}{S_1} = \frac{1.10}{1.50} \approx 0.7333.$$

Update:

$$\hat{x}_1 = \hat{x}_1^- + K_1(y_1 - \hat{x}_1^-) = 0 + 0.7333(1.20 - 0) \approx 0.8800,$$
$$\Sigma_1 = (1 - K_1)\Sigma_1^- = (1 - 0.7333) \cdot 1.10 \approx 0.2933.$$

Step $k = 2$. Prediction:

$$\hat{x}_2^- = \hat{x}_1 \approx 0.8800, \qquad \Sigma_2^- = \Sigma_1 + Q_1 \approx 0.2933 + 0.20 = 0.4933.$$

Innovation and gain:

$$S_2 = \Sigma_2^- + R_2 \approx 0.4933 + 0.10 = 0.5933, \qquad K_2 = \frac{\Sigma_2^-}{S_2} \approx \frac{0.4933}{0.5933} \approx 0.8316.$$

Update:

$$\hat{x}_2 = \hat{x}_2^- + K_2(y_2 - \hat{x}_2^-) \approx 0.8800 + 0.8316(0.90 - 0.8800) \approx 0.8966,$$
$$\Sigma_2 = (1 - K_2)\Sigma_2^- \approx (1 - 0.8316) \cdot 0.4933 \approx 0.0831.$$

Interpretation. At $k = 2$, the sensor is more reliable (R_2 is smaller), so K_2 increases and the filter trusts y_2 more strongly. The posterior uncertainty Σ_2 decreases accordingly.

3.7.2 Example 2 (2D): Constant-Velocity Model with Position Measurements

Model. Now the state is 2D:

$$\boldsymbol{x}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix},$$

where p_k is position and v_k is velocity. For a sampling period $\Delta t = 1$, the constant-velocity dynamics are

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}, \qquad \boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{w}_{k-1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_{k-1}). \tag{3.43}$$

Assume we measure position only:

$$y_k = \boldsymbol{C}\boldsymbol{x}_k + v_k^{(m)}, \qquad \boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad v_k^{(m)} \sim \mathcal{N}(0, R_k). \tag{3.44}$$

Kalman equations. Prediction:

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1}, \qquad \boldsymbol{\Sigma}_k^- = \boldsymbol{A}\boldsymbol{\Sigma}_{k-1}\boldsymbol{A}^T + \boldsymbol{Q}_{k-1}.$$

Update:

$$r_k = y_k - \boldsymbol{C}\hat{\boldsymbol{x}}_k^-, \qquad S_k = \boldsymbol{C}\boldsymbol{\Sigma}_k^-\boldsymbol{C}^T + R_k, \qquad \boldsymbol{K}_k = \boldsymbol{\Sigma}_k^-\boldsymbol{C}^T S_k^{-1},$$
$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k r_k, \qquad \boldsymbol{\Sigma}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{\Sigma}_k^-.$$

Numerical values (one full update at $k = 1$). Assume initial posterior:

$$\hat{\boldsymbol{x}}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \boldsymbol{\Sigma}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Choose a time-varying process noise (here, at $k = 0$):

$$\boldsymbol{Q}_0 = \begin{bmatrix} 0.10 & 0 \\ 0 & 0.20 \end{bmatrix}, \qquad R_1 = 0.50.$$

Suppose the first measurement is $y_1 = 0.70$.

Prediction to $k = 1$.

$$\hat{\boldsymbol{x}}_1^- = \boldsymbol{A}\hat{\boldsymbol{x}}_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

For the covariance, first compute

$$\boldsymbol{A}\boldsymbol{\Sigma}_0\boldsymbol{A}^T = \boldsymbol{A}\boldsymbol{A}^T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

Then add $\boldsymbol{Q}_0$:

$$\boldsymbol{\Sigma}_1^- = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 0.10 & 0 \\ 0 & 0.20 \end{bmatrix} = \begin{bmatrix} 2.10 & 1 \\ 1 & 1.20 \end{bmatrix}.$$

Update at $k = 1$. Innovation:

$$r_1 = y_1 - \boldsymbol{C}\hat{\boldsymbol{x}}_1^- = 0.70 - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.70 - 1 = -0.30.$$

Innovation covariance (scalar):

$$S_1 = \boldsymbol{C}\boldsymbol{\Sigma}_1^-\boldsymbol{C}^T + R_1 = \Sigma_{1,11}^- + R_1 = 2.10 + 0.50 = 2.60.$$

Kalman gain:

$$\boldsymbol{K}_1 = \boldsymbol{\Sigma}_1^-\boldsymbol{C}^T S_1^{-1} = \begin{bmatrix} 2.10 & 1 \\ 1 & 1.20 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \frac{1}{2.60} = \begin{bmatrix} 2.10 \\ 1 \end{bmatrix} \frac{1}{2.60} = \begin{bmatrix} 0.8077 \\ 0.3846 \end{bmatrix}.$$

Posterior mean:

$$\hat{\boldsymbol{x}}_1 = \hat{\boldsymbol{x}}_1^- + \boldsymbol{K}_1 r_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.8077 \\ 0.3846 \end{bmatrix} (-0.30) \approx \begin{bmatrix} 0.7577 \\ 0.8846 \end{bmatrix}.$$

Posterior covariance:

$$\boldsymbol{\Sigma}_1 = (\boldsymbol{I} - \boldsymbol{K}_1\boldsymbol{C})\boldsymbol{\Sigma}_1^-.$$

Here

$$\boldsymbol{K}_1\boldsymbol{C} = \begin{bmatrix} 0.8077 \\ 0.3846 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.8077 & 0 \\ 0.3846 & 0 \end{bmatrix}, \quad \boldsymbol{I} - \boldsymbol{K}_1\boldsymbol{C} = \begin{bmatrix} 0.1923 & 0 \\ -0.3846 & 1 \end{bmatrix}.$$

Thus

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.1923 & 0 \\ -0.3846 & 1 \end{bmatrix} \begin{bmatrix} 2.10 & 1 \\ 1 & 1.20 \end{bmatrix} \approx \begin{bmatrix} 0.4038 & 0.1923 \\ 0.1923 & 0.8154 \end{bmatrix}.$$

Interpretation. Even though we only measure position, the update changes both position and velocity: the position residual r_1 is partially attributed to velocity through the state covariance (and the coupling in the dynamics). The gain vector $\boldsymbol{K}_1$ shows how a scalar measurement updates each state component.

3.7.3 Python Implementation

We consider a 1D tracking problem with a 2D state:

$$\boldsymbol{x}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix},$$

where p_k is position and v_k is velocity at discrete time k.

Constant-velocity (CV) motion model. With sampling period $\Delta t > 0$, a standard linear CV model is

$$\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}, \qquad \boldsymbol{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{w}_{k-1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_{k-1}). \tag{3.45}$$

The process noise $\boldsymbol{w}_{k-1}$ captures unmodeled effects (e.g., small accelerations). In general we allow time-varying uncertainty:

$$\boldsymbol{Q}_{k-1} \succeq \boldsymbol{0}.$$

Position-only measurement model. We assume that we measure position only:

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k, \qquad \boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k), \tag{3.46}$$

where $\boldsymbol{y}_k \in \mathbb{R}$ is a scalar observation and $\boldsymbol{R}_k \in \mathbb{R}^{1\times 1}$ is a (time-varying) measurement noise covariance (i.e., variance).

Independence assumptions. We assume the standard Kalman conditions:

$$\boldsymbol{w}_k \perp \boldsymbol{v}_j \ \ \forall k, j, \qquad \boldsymbol{w}_k \text{ and } \boldsymbol{v}_k \text{ are white and zero-mean.}$$

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

# -----------------------------
# 2D Kalman Filter (CV model)
# State: x_k = [p_k, v_k]^T
# Measurement: y_k = p_k + noise
# Time-varying Q_k, R_k
# -----------------------------

def simulate_cv_1d(T=80, dt=1.0, q_base=0.05, r_base=0.4, seed=0):
    """
    Simulate constant-velocity motion with random acceleration-like
        process noise.
    Returns: true states X (T+1,2), measurements y (T+1,), Q_k list,
        R_k list
    """
    rng = np.random.default_rng(seed)

    A = np.array([[1.0, dt],
                  [0.0, 1.0]])
    C = np.array([[1.0, 0.0]])  # measure position only

    # True initial state
    x = np.array([0.0, 1.0])  # p0, v0

    X = np.zeros((T + 1, 2), dtype=float)
    y = np.zeros(T + 1, dtype=float)

    # Time-varying noise levels
    Q_list = []
    R_list = []

    # Make time-varying R_k (sensor gets worse in the middle)
    for k in range(T + 1):
        bump = 1.0 + 2.5 * np.exp(-0.5 * ((k - 0.6 * T) / (0.12 * T))
            ** 2)
        Rk = np.array([[r_base * bump]], dtype=float)  # scalar
            measurement variance
        R_list.append(Rk)

    # Make time-varying Q_k (process more uncertain near the end)
    for k in range(T):
        ramp = 1.0 + 1.5 * (k / max(T - 1, 1))
        # Simple diagonal Q_k on [p, v]; you can also use a physically-
            motivated accel model.
        Qk = np.array([[q_base * ramp, 0.0],
                       [0.0,           2.0 * q_base * ramp]], dtype=
                           float)
        Q_list.append(Qk)

    # Generate trajectory + measurements
    X[0] = x
    y[0] = (C @ x.reshape(-1, 1)).item() + rng.normal(0.0, np.sqrt(
        R_list[0].item()))

    for k in range(1, T + 1):
        # Process noise
        wk = rng.multivariate_normal(mean=np.zeros(2), cov=Q_list[k -
            1])
        x = A @ x + wk
        X[k] = x

        # Measurement noise
        vk = rng.normal(0.0, np.sqrt(R_list[k].item()))
        y[k] = (C @ x.reshape(-1, 1)).item() + vk

    return A, C, X, y, Q_list, R_list


def kf_filter(A, C, y, Q_list, R_list, x0_hat, Sigma0, u=None, B=None):
    """
    Kalman filter for:
      x_k = A x_{k-1} + B u_{k-1} + w_{k-1}, w ~ N(0, Q_{k-1})
      y_k = C x_k + v_k, v ~ N(0, R_k)
    with time-varying Q_{k-1}, R_k.

    Returns:
      x_hat (T+1,2), Sigma (T+1,2,2),
      x_hat_minus (T+1,2), Sigma_minus (T+1,2,2),
      K (T+1,2,1), innovations r (T+1,)
    """
    T = len(y) - 1
    n = A.shape[0]
    m = C.shape[0]

    if u is None:
        u = np.zeros((T, 1))
    if B is None:
        B = np.zeros((n, 1))

    x_hat = np.zeros((T + 1, n), dtype=float)
    x_hat_minus = np.zeros((T + 1, n), dtype=float)
    Sigma = np.zeros((T + 1, n, n), dtype=float)
    Sigma_minus = np.zeros((T + 1, n, n), dtype=float)
    K = np.zeros((T + 1, n, m), dtype=float)
    r = np.zeros(T + 1, dtype=float)

    # Initialize
    x_hat[0] = x0_hat
    Sigma[0] = Sigma0

    # (Optional) incorporate y0 as an update; here we do a standard
       update at k=0
    # Prediction at k=0 is trivial (use initial prior as predicted),
       then update with y0
    x_hat_minus[0] = x_hat[0]
    Sigma_minus[0] = Sigma[0]
    S0 = (C @ Sigma_minus[0] @ C.T + R_list[0])
    K[0] = (Sigma_minus[0] @ C.T) @ np.linalg.inv(S0)
    r[0] = y[0] - (C @ x_hat_minus[0].reshape(-1, 1)).item()
    x_hat[0] = x_hat_minus[0] + (K[0] * r[0]).reshape(-1)
    Sigma[0] = (np.eye(n) - K[0] @ C) @ Sigma_minus[0]

    # Main loop
    for k in range(1, T + 1):
        # Predict
        x_hat_minus[k] = (A @ x_hat[k - 1].reshape(-1, 1) + B @ u[k -
           1].reshape(-1, 1)).reshape(-1)
        Sigma_minus[k] = A @ Sigma[k - 1] @ A.T + Q_list[k - 1]

        # Update
        Sk = C @ Sigma_minus[k] @ C.T + R_list[k]
        K[k] = (Sigma_minus[k] @ C.T) @ np.linalg.inv(Sk)
        r[k] = y[k] - (C @ x_hat_minus[k].reshape(-1, 1)).item()
        x_hat[k] = x_hat_minus[k] + (K[k] * r[k]).reshape(-1)

        # Covariance update (simple form; Joseph form is more
           numerically robust)
        Sigma[k] = (np.eye(n) - K[k] @ C) @ Sigma_minus[k]

    return x_hat, Sigma, x_hat_minus, Sigma_minus, K, r

```

```python

# ------------------------------
# Run example
# ------------------------------
A, C, X_true, y, Q_list, R_list = simulate_cv_1d(
    T=120, dt=1.0,
    q_base=0.04, r_base=0.25,
    seed=3
)

# Prior (intentionally imperfect)
x0_hat = np.array([0.0, 0.0])
Sigma0 = np.array([[2.0, 0.0],
                   [0.0, 2.0]])

x_hat, Sigma, x_hat_minus, Sigma_minus, K, r = kf_filter(
    A=A, C=C, y=y,
    Q_list=Q_list, R_list=R_list,
    x0_hat=x0_hat, Sigma0=Sigma0
)

# ------------------------------
# Plot
# ------------------------------
t = np.arange(len(y))

plt.figure()
plt.plot(t, X_true[:, 0], label="true position $p_k$")
plt.plot(t, y, label="measured position $y_k$")
plt.plot(t, x_hat[:, 0], label="KF estimate $\hat{p}_k$")
plt.xlabel("k")
plt.ylabel("position")
plt.title("2D Kalman filter (state=[position, velocity], measurement=
    position)")
plt.legend()
plt.grid(True)

plt.figure()
plt.plot(t, X_true[:, 1], label="true velocity $v_k$")
plt.plot(t, x_hat[:, 1], label="KF estimate $\hat{v}_k$")
plt.xlabel("k")
plt.ylabel("velocity")
plt.title("Velocity is inferred from position-only measurements")
plt.legend()
plt.grid(True)

plt.figure()
plt.plot(t, Sigma[:, 0, 0], label="posterior var(p): $Σ_k[0,0]$")
plt.plot(t, Sigma[:, 1, 1], label="posterior var(v): $Σ_k[1,1]$")
plt.xlabel("k")
plt.ylabel("variance")
plt.title("Posterior uncertainties shrink/grow with Q_k and R_k")
plt.legend()
plt.grid(True)

plt.show()
```

Listing 2: Kalman Filter implementation

3.8 Kalman Filter Algorithm and Intuition

3.8.1 Kalman Filter Algorithm (Predict–Update)

We now summarize the Kalman filter as a simple predict–update recursion. We allow time-varying system matrices and noise covariances.

```
  Input: ${\boldsymbol{A}_k, \boldsymbol{B}_k, \boldsymbol{C}_k}$; ${\boldsymbol{Q}_k, \boldsymbol{R}_k}$; initial $(\hat{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_0)$; measurements $\{\boldsymbol{y}_k\}_{k\geq 1}$; inputs
         $\{\boldsymbol{u}_k\}_{k\geq 0}$
  Output: Filtered estimates $\{(\hat{\boldsymbol{x}}_k, \boldsymbol{\Sigma}_k)\}_{k\geq 1}$
1 for $k = 1, 2, \dots$ do
      // Prediction (Time Update)
2     $\hat{\boldsymbol{x}}_k^- \leftarrow \boldsymbol{A}_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}_k\boldsymbol{u}_{k-1}$
3     $\boldsymbol{\Sigma}_k^- \leftarrow \boldsymbol{A}_k\boldsymbol{\Sigma}_{k-1}\boldsymbol{A}_k^\top + \boldsymbol{Q}_{k-1}$
      // Update (Measurement Update)
4     $\boldsymbol{r}_k \leftarrow \boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-$                          // innovation
5     $\boldsymbol{S}_k \leftarrow \boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top + \boldsymbol{R}_k$                  // innovation cov.
6     $\boldsymbol{K}_k \leftarrow \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top\boldsymbol{S}_k^{-1}$                         // Kalman gain
7     $\hat{\boldsymbol{x}}_k \leftarrow \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\boldsymbol{r}_k$
8     $\boldsymbol{\Sigma}_k \leftarrow (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)\boldsymbol{\Sigma}_k^-$
                                                   // or Joseph form (3.51)
```

Conceptual summary. The Kalman filter alternates between:

- **Predict:** use the model to propagate mean and covariance forward.
- **Correct:** use the measurement to reduce uncertainty and refine the estimate.

This recursion is the closed-form Gaussian solution of the Bayes filter for linear systems.

3.8.2 Intuition I: The Role of the Kalman Gain

At time k, the Kalman gain is

$$\boxed{\boldsymbol{K}_k = \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top\big(\boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top + \boldsymbol{R}_k\big)^{-1} = \boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top\boldsymbol{S}_k^{-1},} \tag{3.47}$$

where

$$\boxed{\boldsymbol{S}_k = \boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top + \boldsymbol{R}_k} \tag{3.48}$$

is the innovation covariance.

Equation (3.47) makes explicit that $\boldsymbol{K}_k$ is an *automatic weighting* between the model prediction and the measurement:

- **Large measurement noise $\boldsymbol{R}_k$** implies $\boldsymbol{S}_k$ is large, hence $\boldsymbol{K}_k$ becomes small. The update term $\boldsymbol{K}_k\boldsymbol{r}_k$ is damped, and the filter *trusts the model* (prediction) more.
- **Large prior uncertainty $\boldsymbol{\Sigma}_k^-$** makes the term $\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^T$ large, and therefore increases $\boldsymbol{K}_k$. The filter *trusts the measurement* more because the prediction is uncertain.

Thus, $\boldsymbol{K}_k$ plays the role of a *data-driven mixing coefficient* that balances model reliability versus sensor reliability at each time step.

3.8.3 Intuition II: Innovation = New Information

The innovation (measurement residual) is defined as

$$\boxed{\boldsymbol{r}_k = \boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-.} \tag{3.49}$$

The term $\boldsymbol{C}_k\hat{\boldsymbol{x}}_k^-$ is what the model *predicts* the measurement should be. The residual $\boldsymbol{r}_k$ captures the component of the measurement that is *not explained* by the prediction.

Two limiting cases are instructive:

- If $\boldsymbol{r}_k \approx \boldsymbol{0}$, the measurement agrees with the prediction, so the update $\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\boldsymbol{r}_k$ makes only a small correction.
- If $\|\boldsymbol{r}_k\|$ is large, the measurement disagrees with the prediction. The correction is strong, but it is still modulated by the gain $\boldsymbol{K}_k$ (i.e., by uncertainty).

In this sense, $\boldsymbol{r}_k$ is the *new information* contained in $\boldsymbol{y}_k$ relative to what was already implied by past measurements and the system model.

3.8.4 Intuition III: Why Uncertainty Shrinks After the Update

The posterior covariance update (basic form) is

$$\boxed{\boldsymbol{\Sigma}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)\boldsymbol{\Sigma}_k^-.} \tag{3.50}$$

This shows that the measurement reduces uncertainty only along directions that are observed by the sensor. Indeed, the measurement model maps the state into measurement space through $\boldsymbol{C}_k$; therefore, only components of $\boldsymbol{x}_k$ that influence $\boldsymbol{y}_k$ can be corrected.

Geometrically:

- The **prediction step** propagates uncertainty through dynamics and injects process noise (via $\boldsymbol{Q}_{k-1}$), typically *expanding* the covariance ellipsoid.
- The **update step** uses the measurement to *contract* uncertainty in the observed subspace, by multiplying by $(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)$.

Unobserved directions remain uncertain (or may continue to grow during prediction).

Remark (numerically robust covariance update). In implementations one often uses the *Joseph form*, which preserves symmetry and positive semidefiniteness better in finite precision:

$$\boxed{\boldsymbol{\Sigma}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)\boldsymbol{\Sigma}_k^-(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)^\top + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^\top.} \tag{3.51}$$

3.9 Practical Considerations

3.9.1 When Assumptions Are Violated

The classical Kalman filter is exact (and optimal in the MMSE sense) for *linear* dynamics, *linear* observations, and *Gaussian* noise. When these assumptions do not hold, the algorithm can still be used as an approximation, but performance may degrade, and in extreme cases the filter may become inconsistent (covariance too small) or even diverge.

Nonlinear dynamics and/or nonlinear observations. In many systems the state and measurement models take the nonlinear form

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) + \boldsymbol{w}_{k-1}, \qquad \boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k, \tag{3.52}$$

with $\boldsymbol{w}_{k-1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_{k-1})$ and $\boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k)$. In this case, the Bayes filter recursion remains valid, but the posterior is generally *not Gaussian*, so it cannot be represented exactly by a mean and covariance.

Common approximations include:

- **Extended Kalman Filter (EKF):** linearize f and h around the current estimate using Jacobians, then apply a Kalman-like update in the locally linearized model.
- **Unscented Kalman Filter (UKF):** propagate a deterministic set of sigma points through f and h to approximate mean/covariance without explicit Jacobians.
- **Particle filters:** approximate the belief by a weighted set of samples, enabling nonlinear and non-Gaussian inference at higher computational cost.

Non-Gaussian noise, heavy tails, and outliers. If the noise is heavy-tailed (e.g., contains outliers), the Gaussian likelihood implicit in the Kalman update may assign too much influence to corrupted measurements. In practice this can lead to large residuals that cause erroneous corrections. A large literature studies **robust** variants, for example replacing the quadratic loss underlying MMSE with robust losses (Huber/Tukey), or modeling the noise with heavy-tailed distributions (e.g., Student-t), which effectively downweights outliers.

Model mismatch and incorrect uncertainty specification. Even when the model is linear, performance depends crucially on how well $\boldsymbol{Q}_{k-1}$ (process uncertainty) and $\boldsymbol{R}_k$ (measurement uncertainty) capture reality. If they are poorly tuned, the filter may become:

- **Sluggish** (too much trust in the model),
- **Noisy** (too much trust in measurements),
- **Inconsistent** (covariance underestimates true error),
- **Divergent** (errors grow and the estimate becomes unstable).

Takeaway. The Kalman filter is only as good as the assumed model and its uncertainty description: $\boldsymbol{Q}_{k-1}$, $\boldsymbol{R}_k$ encode *what we do not know.*

3.9.2 Effect of Tuning $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$

The tuning intuition can be read directly from the gain expression

$$\boldsymbol{K}_k = \boldsymbol{\Sigma}_k^- \boldsymbol{C}_k^\top \left(\boldsymbol{C}_k \boldsymbol{\Sigma}_k^- \boldsymbol{C}_k^\top + \boldsymbol{R}_k \right)^{-1}. \tag{3.53}$$

Recall also that the prior covariance is produced by the prediction step:

$$\boldsymbol{\Sigma}_k^- = \boldsymbol{A}_k \boldsymbol{\Sigma}_{k-1} \boldsymbol{A}_k^\top + \boldsymbol{Q}_{k-1}. \tag{3.54}$$

Increasing $\boldsymbol{Q}_{k-1}$ (more process noise). From (3.54), increasing $\boldsymbol{Q}_{k-1}$ increases $\boldsymbol{\Sigma}_k^-$, which typically increases $\boldsymbol{K}_k$ in (3.53). Therefore the update term $\boldsymbol{K}_k\boldsymbol{r}_k$ becomes larger.

Behavior: the filter trusts measurements more. This makes the estimate more responsive to changes, but typically noisier.

Increasing $\boldsymbol{R}_k$ (more measurement noise). From (3.53), increasing $\boldsymbol{R}_k$ increases the denominator $\boldsymbol{C}_k\boldsymbol{\Sigma}_k^-\boldsymbol{C}_k^\top + \boldsymbol{R}_k$, thereby decreasing $\boldsymbol{K}_k$.

Behavior: the filter trusts the model prediction more. This makes the estimate smoother, but typically slower to react to real changes.

A useful scalar intuition (1D case). If $C_k = 1$ and everything is scalar,

$$K_k = \frac{\Sigma_k^-}{\Sigma_k^- + R_k}, \qquad \Sigma_k^- = \Sigma_{k-1} + Q_{k-1}.$$

Hence:

$$Q_{k-1} \uparrow \Rightarrow\ \Sigma_k^- \uparrow \Rightarrow\ K_k \uparrow, \qquad R_k \uparrow \Rightarrow\ K_k \downarrow.$$

This explicitly shows how Q and R control the model-versus-measurement weighting.

Practical implication. Tuning $\boldsymbol{Q}_{k-1}$ and $\boldsymbol{R}_k$ is not merely numerical housekeeping: it encodes the designer's belief about uncertainty and determines the filter's qualitative behavior.

3.10 Filtering vs. Smoothing and the Batch Perspective

3.10.1 Filtering vs. Smoothing

Consider a linear–Gaussian state-space model and a finite sequence of measurements $\boldsymbol{y}_{1:T}$.

Filtering (online / causal inference). The Kalman filter computes the *filtering distribution*

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}), \tag{3.55}$$

which depends only on measurements available up to time k. The output of the filter is the posterior mean and covariance

$$(\hat{\boldsymbol{x}}_k, \boldsymbol{\Sigma}_k),$$

and the recursion is causal and suitable for real-time operation.

Smoothing (offline / non-causal inference). If all measurements $\boldsymbol{y}_{1:T}$ are available, we may instead compute the *smoothing distribution*

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:T}), \tag{3.56}$$

which uses both past *and future* measurements relative to time k. This yields the smoothed estimates

$$(\hat{\boldsymbol{x}}_k^s, \boldsymbol{\Sigma}_k^s).$$

Key property. Smoothing can never be worse than filtering in terms of uncertainty:

$$\boxed{\boldsymbol{\Sigma}_k^s \preceq \boldsymbol{\Sigma}_k \qquad \forall k,} \tag{3.57}$$

where $\preceq$ denotes the Loewner (positive semidefinite) ordering. Intuitively, future measurements contain information about past states and can be used to retrospectively correct earlier estimates.

Conceptual summary.

- Filtering: real-time, causal, uses $\boldsymbol{y}_{1:k}$.
- Smoothing: offline, non-causal, uses $\boldsymbol{y}_{1:T}$.
- Future data always improves (or leaves unchanged) past state estimates.

3.10.2 Kalman Smoothing: Rauch–Tung–Striebel (RTS) Algorithm

For linear–Gaussian systems, smoothing can be performed efficiently by a backward pass after Kalman filtering. The most common smoother is the **Rauch–Tung–Striebel (RTS) smoother**.

Forward pass (Kalman filter). Run the standard Kalman filter for $k = 0{:}T$ and store:

$$\hat{\boldsymbol{x}}_k, \quad \hat{\boldsymbol{x}}_{k+1}^-, \quad \boldsymbol{\Sigma}_k, \quad \boldsymbol{\Sigma}_{k+1}^-.$$

Backward pass (smoothing recursion). For $k = T-1, \dots, 0$, define the **smoother gain**

$$\boxed{\boldsymbol{G}_k = \boldsymbol{\Sigma}_k \boldsymbol{A}_k^\top (\boldsymbol{\Sigma}_{k+1}^-)^{-1}.} \tag{3.58}$$

The smoothed mean is updated as

$$\boxed{\hat{\boldsymbol{x}}_k^s = \hat{\boldsymbol{x}}_k + \boldsymbol{G}_k\Big(\hat{\boldsymbol{x}}_{k+1}^s - \hat{\boldsymbol{x}}_{k+1}^-\Big),} \tag{3.59}$$

and the smoothed covariance as

$$\boxed{\boldsymbol{\Sigma}_k^s = \boldsymbol{\Sigma}_k + \boldsymbol{G}_k\Big(\boldsymbol{\Sigma}_{k+1}^s - \boldsymbol{\Sigma}_{k+1}^-\Big)\boldsymbol{G}_k^\top.} \tag{3.60}$$

Interpretation. The backward correction term $\hat{\boldsymbol{x}}_{k+1}^s - \hat{\boldsymbol{x}}_{k+1}^-$ represents information revealed by future measurements. The smoother gain $\boldsymbol{G}_k$ propagates this information backward through the dynamics.

3.10.3 Batch View: MAP Estimation of the Entire Trajectory

Filtering and smoothing can also be understood from a **batch optimization** viewpoint. Under the linear–Gaussian model, estimating the entire state trajectory $\boldsymbol{x}_{0:T}$ is equivalent to a maximum a posteriori (MAP) problem.

Batch MAP formulation. The posterior over the full trajectory is Gaussian, and the MAP estimate solves

$$\boxed{\boldsymbol{x}_{0:T}^{\star} = \arg\min_{\boldsymbol{x}_{0:T}} \left[\sum_{k=1}^{T} \|\boldsymbol{y}_k - \boldsymbol{C}_k \boldsymbol{x}_k\|_{\boldsymbol{R}_k^{-1}}^2 + \sum_{k=1}^{T} \|\boldsymbol{x}_k - \boldsymbol{A}_k \boldsymbol{x}_{k-1} - \boldsymbol{B}_k \boldsymbol{u}_{k-1}\|_{\boldsymbol{Q}_{k-1}^{-1}}^2 + \|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0\|_{\boldsymbol{\Sigma}_0^{-1}}^2 \right].} \tag{3.61}$$

This is a quadratic least-squares problem with a block-tridiagonal structure in time.

Relationship to Kalman filtering and smoothing.

- Kalman filtering computes the MAP estimate *incrementally* using only past data.
- Kalman smoothing computes the same solution as (3.61), but efficiently, without explicitly forming or solving the full batch problem.
- Both are algorithmic realizations of Gaussian Bayesian inference.

Big picture.

- **Filtering:** online solution of a growing inference problem.
- **Smoothing:** offline solution using all data.
- **Batch MAP:** global optimization perspective.

Kalman filtering and smoothing are not ad hoc algorithms; they are efficient recursive solvers for structured Gaussian least-squares problems.