Topic 2B: State-Space Models for Time Series Forecasting

Victor M. Preciado

Contents

1 State-Space Models for Time Series

State-Space Models (SSMs) provide a cohesive framework for modeling time series data by capturing both the underlying dynamics of the system and the uncertainty present in the observations. This framework builds on the idea of **hidden latent states**, which evolve over time and represent the internal—often unobservable—conditions of the system. The evolution of these states follows a recursive process, where each new state is determined by the previous state, any external inputs, and a noise term that accounts for uncertainty in the system. Meanwhile, the observations are modeled as noisy functions of the hidden states, effectively linking the unobserved dynamics to the data we can observe directly. By incorporating both state evolution and observation mechanisms, SSMs offer a flexible way to model complex time series behaviors.

The state-space framework serves as a unified theoretical foundation for a broad spectrum of time series models. Classical linear models, such as AR, MA, and their combinations, can be viewed as special cases of state-space models where the state equations are linear and the hidden states directly relate to lagged values of the observed data. Moreover, the state-space formulation extends seamlessly to modern deep learning approaches, such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and neural state-space models such as Mamba. These advanced models can be viewed as extensions of the classical state-space approach, where neural networks are used to parameterize the state evolution and observation equations, allowing for the modeling of non-linear, non-stationary, and highly complex time series data. By positioning SSMs as a general and adaptable framework, we can clarify how these various models, though often discussed separately, all adhere to the same underlying principles.

1.1 State-Space Models

Mathematically, an SSM consists of two main components: the *state equation* (also known as the process model) and the *observation equation* (also known as the measurement model).

- **State Equation**: This equation governs the time evolution of the hidden state vector $\mathbf{X}_k$, which represents the unobserved internal dynamics of the system at time k. The most general version of the state equation is expressed as a nonlinear and time-variant recursive relation:

$$\mathbf{X}_{k+1} = f_{\boldsymbol{\theta}_x}(\mathbf{X}_k, \mathbf{u}_k, \boldsymbol{\eta}_k), \quad \text{with initial condition}\, \mathbf{X}_0 = \mathbf{x}_0, \tag{1}$$

where $\mathbf{X}_k$ is a random vector[1] called the **hidden state** at time k and $\mathbf{u}_k$

[1]Note that this vector is random due to the inclusion of the process noise $\boldsymbol{\eta}_k$. In what follows, we denote random vectors by bold capital letters.

denotes the vector of *deterministic* **external inputs** (such as exogenous variables); $f_{\boldsymbol{\theta}_x}(\cdot)$ is the **state transition function** that describes how the hidden state vector evolves based on the current state, inputs, and parameters with $\boldsymbol{\theta}_x$ being the vector of **model parameters** for the state equation, and $\boldsymbol{\eta}_k$ is the **process noise**, which accounts for uncertainties and unmodeled dynamics in the system's evolution.

- **Observation Equation**: The observation equation maps the random hidden state vector $\mathbf{X}_k$ to the observable output vector $\mathbf{Y}_k$, which represents the measured data at time k. This relationship is formalized as:

$$\mathbf{Y}_k = g_{\boldsymbol{\theta}_y}(\mathbf{X}_k, \mathbf{u}_k, \boldsymbol{\varepsilon}_k), \tag{2}$$

  where $\mathbf{Y}_k$ denotes the **observable data** at time k, $\mathbf{u}_k$ represents the external inputs, $g_{\boldsymbol{\theta}_y}(\cdot)$ is the observation function with parameters $\boldsymbol{\theta}_y$, and $\boldsymbol{\varepsilon}_k$ is the **observation noise**, accounting for measurement errors or uncertainties in the data collection process. This equation encapsulates how the latent dynamics of the system are reflected in the observable data, while also acknowledging the inherent noise and errors in the measurement process.

Together, these two equations recursively describe the stochastic behavior of the system, providing a clear distinction between the latent system dynamics (through the state equation) and the process of observation (through the observation equation). Importantly, the state-space framework induces a **Markov process**, where all relevant information about the system's evolution is encapsulated in the hidden state vector $\mathbf{X}_k$. This Markov property implies that the future behavior of the system depends solely on the current hidden state and not on the full history of past states.

Example 1: Hodgkin-Huxley Model

The Hodgkin-Huxley model describes the dynamical behavior of a biological neuron using a four-dimensional state vector. Let us represent the states as follows: $\mathbf{x}_k = [x_{k1}, x_{k2}, x_{k3}, x_{k4}]^\intercal$, where x_{k1} is the membrane potential and x_{k2}, x_{k3}, and x_{k4} are hidden gating variables. The evolution of these states over time is governed by a set of ordinary differential equations that can be discretized as follows:

- **Membrane potential** x_{k1} (discretized form):

$$\begin{aligned} x_{k+1,1} = x_{k1} + \frac{\Delta t}{C_m}\Big(& u_k - \gamma_1 x_{k3}^3 x_{k4}(x_{k1} - \delta_1) \\ & - \gamma_2 x_{k2}^4 (x_{k1} - \delta_2) - \gamma_3 (x_{k1} - \delta_3)\Big) + \eta_k, \end{aligned}$$

  where the γ's and δ's are model parameters, u_k is an external input, and η_k is the process noise accounting for random perturbations.

- **Hidden gating variables:** The gating variables x_{k2}, x_{k3}, and x_{k4} evolve according to the following discretized update equations, where $\sigma_2(x_{k1})$, $\sigma_3(x_{k1})$, and $\sigma_4(x_{k1})$ represent three different sigmoidal functions of the membrane potential x_{k1}:

$$
\begin{aligned}
x_{k+1,2} &= x_{k2} + \Delta t \cdot \left(\sigma_2(x_{k1})(1 - x_{k2}) - (1 - \sigma_2(x_{k1})) \cdot x_{k2}\right),\\
x_{k+1,3} &= x_{k3} + \Delta t \cdot \left(\sigma_3(x_{k1})(1 - x_{k3}) - (1 - \sigma_3(x_{k1})) \cdot x_{k3}\right),\\
x_{k+1,4} &= x_{k4} + \Delta t \cdot \left(\sigma_4(x_{k1})(1 - x_{k4}) - (1 - \sigma_4(x_{k1})) \cdot x_{k4}\right).
\end{aligned}
$$

  The sigmoidal functions describe the probability of the gating variables opening or closing based on the membrane potential x_{k1}.

- **Observation Equation:** The observed output y_k at each time step is a noisy measurement of the membrane potential x_{k1}: $y_k = x_{k1} + \varepsilon_k$, where ε_k is the observation noise.

Note that the set of discretized equations described above can be written as a standard state-space model in (1) and (2). The hidden latent state variables are both the membrane potential and the gating variables, while the output is a noisy version of the membrane potential. In the following figure, we plot the joint evolution of the three hidden gating variables in a 3D plot (left) and the membrane potential (right) for a particular choice of parameters (code available in GitHub).

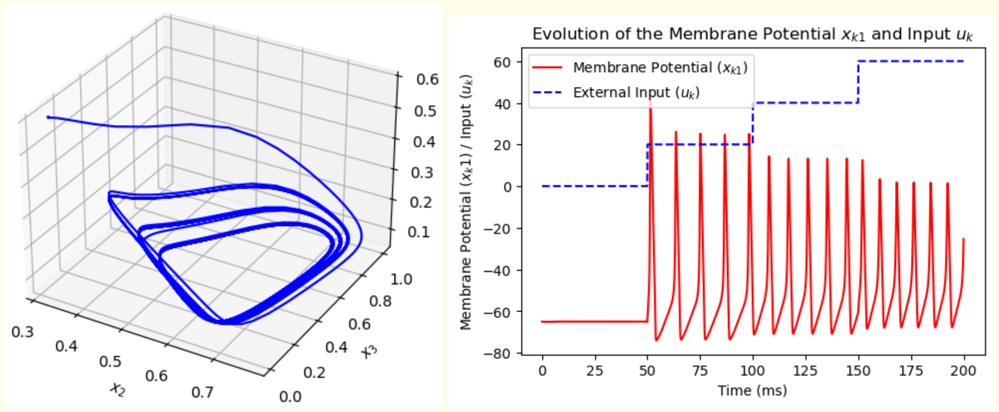

Figure 1: (Left) 3D plot for the joint evolution of gating variables, x_{k2}, x_{k3}, and x_{k4}. (Right) Evolution of the membrane potential x_{k1} and the input signal u_k.

1.2 Hidden Markov Models (HMMs)

Building on the state-space framework, **Hidden Markov Models (HMMs)** offer a specialized yet powerful instance of state-space models, where the hidden states follow a Markov process over a *discrete* set of H possible hidden states, $\mathcal{S} = \{s_1, \ldots, s_H\}$. Like general state-space models, HMMs rely on hidden states that evolve over time according to certain probabilistic rules, while the observed data are linked to the hidden states via a probabilistic observation process. The main distinction between HMMs and SSMs is that the hidden states in an HMM take values from a discrete set, making them particularly suited for modeling systems where the underlying process can be represented by a finite number of states.

Although HMMs can produce either discrete or continuous outputs, we will focus on the case of discrete observations. In particular, at each time step k, the current hidden state X_k generates an observable output Y_k, drawn from a discrete set of M possible outcomes, i.e., $Y_k \in \mathcal{O} = \{o_1, o_2, \ldots, o_M\}$, according to a predefined **emission probability distribution**. This distribution specifies the conditional probability of observing a given output Y_k based on the current hidden state X_k.

Similar to a state-space model, we can characterize an HMM by two main components:

1. **State Transition Process:** The hidden state X_k evolves according to a *time-homogeneous Markov chain* $\mathcal{X} = (X_0, X_1, \ldots, X_L)$, where $X_k \in \mathcal{S} = \{s_1, s_2, \ldots, s_H\}$ for all k. The probabilities of transitioning between states are described by a row-stochastic **state transition matrix** $P \in [0,1]^{H \times H}$, where each element p_{ij} represents the probability of moving from state s_i to state s_j. Specifically,

$$[P]_{ij} = p_{ij} = \mathbb{P}(X_{k+1} = s_j \mid X_k = s_i), \quad \text{for all } i, j \in \{1, \ldots, H\}.$$

   The process starts with an **initial state distribution**, represented by the vector:

$$\boldsymbol{\pi}_0 = \left[\mathbb{P}(X_0 = s_1), \quad \mathbb{P}(X_0 = s_2), \quad \cdots, \quad \mathbb{P}(X_0 = s_H)\right]^\intercal \in [0,1]^H,$$

   where the i-th entry $\pi_{0,i}$ gives the probability that the process begins in state s_i. This initial distribution, while not directly observable, strongly influences the system?s future evolution. According to the propagation rules for homogeneous Markov chains, the state distribution at time k is given by:

$$\boxed{\boldsymbol{\pi}_k = P^\intercal \boldsymbol{\pi}_{k-1} = (P^\intercal)^k \boldsymbol{\pi}_0.}$$

2. **Observation Process:** At each time step k, the HMM generates an output $Y_k \in \mathcal{O} = \{o_1, o_2, \ldots, o_E\}$ stochastically, based on the hidden state X_k.

This relationship is governed by a set of **emission probabilities**, $\{c_{he}\colon h = [H];\, e = [E]\}$, where each element c_{he} represents the conditional probability of observing $Y_k = o_e$ given that $X_k = s_h$. These emission probabilities can be organized into an **emission matrix** $C \in [0,1]^{H\times E}$. Formally, the entries of this matrix are defined as:

$$[C]_{he} = c_{he} = \mathbb{P}(Y_k = o_e \mid X_k = s_h), \quad \text{for all } h \in \{1,\ldots,H\},\, e \in \{1,\ldots,E\}.$$

Since the HMM generates an output at every time step k, the emission matrix C is a row-stochastic matrix, meaning that the sum of the probabilities across each row is 1, i.e., $\sum_{e=1}^{E} C_{he} = 1$ for all $h \in \{1,\ldots,H\}$. Using the total probability theorem, we can express the probability of observing a specific output $Y_k = o_e$, given the initial information set $\mathcal{F}_0$, as:

$$\mathbb{P}(Y_k = o_e \mid \mathcal{F}_0) = \sum_{h=1}^{H} \underbrace{\mathbb{P}(Y_k = o_e \mid X_k = s_h)}_{c_{he}} \underbrace{\mathbb{P}(X_k = s_h \mid \mathcal{F}_0)}_{\pi_{k,h}}. \tag{3}$$

Now, define the vector $\boldsymbol{\chi}_k$, whose entries represent the probabilities of observing each possible output given the initial distribution, as follows:

$$\boldsymbol{\chi}_k = \begin{bmatrix}\mathbb{P}(Y_k = o_1 \mid \mathcal{F}_0) & \mathbb{P}(Y_k = o_2 \mid \mathcal{F}_0) & \cdots & \mathbb{P}(Y_k = o_E \mid \mathcal{F}_0)\end{bmatrix}^\intercal.$$

Thus, using matrix multiplication, we can rewrite (3) in vector form as:

$$\boxed{\boldsymbol{\chi}_k = C^\intercal \boldsymbol{\pi}_k = C^\intercal (P^\intercal)^k \boldsymbol{\pi}_0.}$$

This formulation encapsulates the probabilistic nature of both the state transitions and the observations, providing a powerful framework for modeling sequential data where the underlying structure is not directly observable. HMMs excel in scenarios where the system alternates between distinct regimes that are not directly observable but can be inferred from the data. They have been widely applied in areas such as speech recognition, biological sequence analysis, and financial market modeling.

Example 2: Disease Progression as a Hidden Markov Model

In epidemiology, we can model the progression of a disease in a particular individual using HMMs, as follows.

1. **Hidden States:** Define a set of hidden states such that each state represents the health status of an individual. For example, we could use the following set:

$$X_k \in \{s_1 = \texttt{Susceptible}, s_2 = \texttt{Infected}, s_3 = \texttt{Recovered}, s_4 = \texttt{Dead}\}.$$

These hidden states evolve over time according to a Markov process, capturing how an individual transitions from being susceptible to infected, to either recovery or death.

2. **State Transition Matrix:** The matrix P governs the transitions between the different health states. As an example:

$$P = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 \\ 0 & 0.79 & 0.2 & 0.01 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where each row represents the probabilities of transitioning from one health state to another. For instance, an individual who is currently `Susceptible` has a 90% chance of remaining healthy and a 10% chance of becoming `Infected`.

Insert diagram for example.

3. **Observations:** The observations represent the measurable behaviors or conditions of the individual, such as being masked, visiting a hospital, or being in the mortuary. The set of possible observations in this example is:

$$Y_k \in \{o_1 = \texttt{Unmasked}, o_2 = \texttt{Masked}, o_3 = \texttt{Hospitalized}, o_4 = \texttt{Mortuary}\},$$

These observations provide indirect evidence of the individual's underlying health state.

4. **Emission Probability Matrix:** The matrix C defines the probability of observing each behavior given the individual's hidden health state. For example:

$$C = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 \\ 0.2 & 0.5 & 0.3 & 0 \\ 0.9 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where each row corresponds to the observation probabilities conditioned on a hidden health state. For instance, if the individual is `Infected`, there is a 50% chance they will wear a mask, a 30% chance they will be hospitalized, and a 20% chance they show no symptoms.

5. **Initial State Distribution:** The vector of initial state probabilities $\boldsymbol{\pi}_0$ provides the probabilities of an individual starting in each health state. For example:

$$\boldsymbol{\pi}_0 = \begin{bmatrix} 0.95 & 0.05 & 0 & 0 \end{bmatrix}^\intercal,$$

meaning that at the beginning, there is a 95% chance the individual is `Healthy` and a 5% chance they are already `Infected`.

6. **Evolution of Emission Probabilities:** The vector $\boldsymbol{\chi}_k = C^\intercal (P^\intercal)^k \boldsymbol{\pi}_0$ represents the evolution of the emission probabilities over time, i.e., how likely we are to observe certain outputs at each time step k. For example, at time $k = 1$, we can compute $\boldsymbol{\chi}_1$ as follows:

$$\boldsymbol{\pi}_1 = (P^\intercal)\boldsymbol{\pi}_0 = \begin{bmatrix} 0.855 \\ 0.095 \\ 0.05 \\ 0 \end{bmatrix}, \quad \boldsymbol{\chi}_1 = C^\intercal \boldsymbol{\pi}_1 = \begin{bmatrix} 0.704 \\ 0.2095 \\ 0.086 \\ 0 \end{bmatrix}.$$

Repeating this process for subsequent k values gives the evolution of the emission probabilities over time, reflecting how the individual's health state influences observable behaviors. In Fig. 2, we show the evolution of the hidden states and the observable behaviors for 50 time steps.

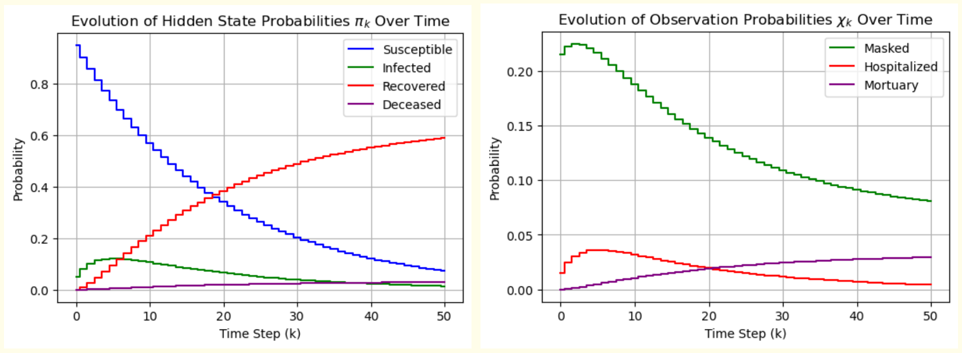

Figure 2: (Left) Evolution of the probabilities of being in each hidden state (e.g., Susceptible, Infected, Recovered, or Deceased). (Right) Evolution of the probabilities of observing a particular measurable behavior (e.g., Masked, Hospitalized, or Mortuary).

1.2.1 Parameter Estimation in Hidden Markov Models ⚠

EM ALGORITHM...

2 Linear State-Space Models for Time Series Analysis

Linear State-Space Models (LSSMs) provide a tractable framework for modeling and forecasting time-series data. By leveraging linear relationships between latent (hidden) states and observed outputs, LSSMs strike an effective balance between complexity and analytical tractability, making them a foundational tool in time series analysis. A major strength of LSSMs is their ability to reconstruct many classical time-series models, such as autoregressive, moving average, and other extensions, within a unified framework. This allows for a seamless transition between different modeling paradigms, while also enabling generalizations to more complex systems.

In an LSSM, both the state equation and the observation process are governed by linear transformations. Specifically, the hidden state at each time step is updated as a linear function of the previous state, external inputs, and process noise, while the observed output is modeled as a linear function of the current state, inputs, and measurement noise. Unlike Hidden Markov Models (HMMs), where the hidden state is a discrete random variable, the hidden state in LSSMs is a *random vector of continuous random variables*, reflecting the continuous nature of the underlying system. These linear transformations can always be conveniently expressed in terms of matrix operations[2]. This matrix formulation not only simplifies the mathematical treatment of the model but also allows for efficient computation, especially when dealing with high-dimensional data.

In the following sections, we will formalize the structure of LSSMs, discuss their properties, and explore how they can be applied to time series forecasting and estimation tasks. Let us now present the core equations of the LSSM in matrix form:

- The **state equation** in an LSSM is mathematically described by the following recursion:

$$\mathbf{X}_{k+1} = A_k\mathbf{X}_k + B_k\mathbf{u}_k + \boldsymbol{\eta}_k, \quad \text{with initial condition } \mathbf{X}_0 = \mathbf{x}_0,$$

  where $\mathbf{X}_k$ is the **hidden state vector** at time k, which is a vector containing continuous random variables. The matrix A_k, known as the **state transition matrix**, defines how the current state vector $\mathbf{X}_k$ influences the subsequent state $\mathbf{X}_{k+1}$, encapsulating the internal dynamics of the system. The matrix B_k, called the **input matrix**, models the effect of known, deterministic **external inputs $\mathbf{u}_k$** on the state evolution. The random vector $\boldsymbol{\eta}_k$ represents **process noise**, typically modeled as a multivariate white Gaussian noise, i.e., $\boldsymbol{\eta}_k \sim_{\text{iid}} \mathcal{N}(\mathbf{0}, Q)$, where Q is the process noise covariance matrix. This state equation governs the system's hidden state dynamics, describing how the state vector

[2]Assuming finite-dimensional state and output vectors.

evolves over time as a function of the previous state, external inputs, and random disturbances.

- The **measurement equation** maps the latent states to the observed data. It is mathematically expressed as:

$$\mathbf{Y}_k = C_k \mathbf{X}_k + D_k \mathbf{u}_k + \boldsymbol{\varepsilon}_k,$$

  where $\mathbf{Y}_k$ is the **observation vector** at time k. The matrix C_k, referred to as the **observation matrix**, maps the hidden state vector $\mathbf{X}_k$ to the observed data, translating the latent dynamics into real-world measurements. The matrix D_k, called the **direct transmission matrix**, captures any direct effects of the inputs on the observations, while the random vector $\boldsymbol{\varepsilon}_k$ represents **measurement noise**, typically modeled as a multivariate white Gaussian noise (independent of the process noise), i.e., $\boldsymbol{\varepsilon}_k \sim_{\text{iid}} \mathcal{N}(\mathbf{0}, R)$, where R is the measurement noise covariance matrix. This equation models how the measurement errors and other external factors may influence the observed data.

In the general formulation of Linear State-Space Models (LSSMs), the matrices A_k, B_k, C_k, and D_k are allowed to be time-varying, meaning their elements can change with each time step k. However, in many practical applications, these matrices are often assumed to be **time-invariant**, in which case the subscript k is dropped from the notation. Thus, the matrices A, B, C, and D remain constant over time. A diagrammatic representation of a time-invariance LSSM can be found in Fig. 3.

Insert diagram.

Figure 3: Block diagram representation of the linear state-space equations, where the D-block represents a one-time-step delay, i.e., $D\,\boldsymbol{X}_{k+1} = \boldsymbol{X}_k$.

Example 3: Autoregressive model as an LSSM

Consider the AR(p) process given by:

$$Y_{k+1} = \phi_1 Y_k + \phi_2 Y_{k-1} + \cdots + \phi_p Y_{k-p+1} + \epsilon_k, \quad \text{where } \epsilon_k \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

We can express this AR(p) process as a Linear State-Space Model (LSSM) by defining the state vector, state equation, and measurement equation as follows.

1. **State Equation**: The state vector $\mathbf{X}_k^{\text{AR}} \in \mathbb{R}^p$ is defined as:

$$\mathbf{X}_k^{\text{AR}} = \begin{bmatrix} Y_k & Y_{k-1} & \cdots & Y_{k-p+2} & Y_{k-p+1} \end{bmatrix}^\intercal.$$

The state equation describes how the state vector evolves over time. Specifically, the state transition equation is given by:

$$\underbrace{\begin{bmatrix} Y_{k+1} \\ Y_k \\ \vdots \\ Y_{k-p+3} \\ Y_{k-p+2} \end{bmatrix}}_{\mathbf{X}_{k+1}^{\mathrm{AR}} \in \mathbb{R}^p} = \underbrace{\begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_{p-1} & \phi_p \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}}_{A^{\mathrm{AR}} \in \mathbb{R}^{p \times p}} \underbrace{\begin{bmatrix} Y_k \\ Y_{k-1} \\ \vdots \\ Y_{k-p+2} \\ Y_{k-p+1} \end{bmatrix}}_{\mathbf{X}_k^{\mathrm{AR}}} + \underbrace{\begin{bmatrix} \epsilon_k \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\boldsymbol{\varepsilon}_k^{\mathrm{AR}} \in \mathbb{R}^p},$$

where A^{AR} is the state transition matrix and $\boldsymbol{\varepsilon}_k^{\mathrm{AR}}$ is the noise vector.

2. **Measurement Equation**: The measurement equation relates the state vector $\mathbf{X}_k^{\mathrm{AR}}$ to the observed data Y_k. The observation equation is:

$$Y_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}}_{C^{\mathrm{AR}} \in \mathbb{R}^{1 \times p}} \mathbf{X}_k^{\mathrm{AR}},$$

where C^{AR} is the observation matrix, which extracts the first element of the state vector (i.e., the current value Y_k).

Example 4: Moving Average as an LSSM

Consider the MA(q) process given by:

$$Y_k = \epsilon_k + \theta_1 \epsilon_{k-1} + \cdots + \theta_q \epsilon_{k-q}, \quad \text{where } \epsilon_k \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

We can express this MA(q) process as a Linear State-Space Model (LSSM) by defining the state vector, state equation, and measurement equation as follows.

1. **State Equation**: The state vector $\mathbf{X}_k^{\mathrm{MA}} \in \mathbb{R}^{q+1}$ is defined as:

$$\mathbf{X}_k^{\mathrm{MA}} = \begin{bmatrix} \epsilon_k & \epsilon_{k-1} & \cdots & \epsilon_{k-q+1} \end{bmatrix}^\intercal.$$

The state equation describes how the state vector evolves over time.

Specifically, the state transition equation is given by:

$$\underbrace{\begin{bmatrix} \epsilon_k \\ \epsilon_{k-1} \\ \vdots \\ \epsilon_{k-q+1} \\ \epsilon_{k-q} \end{bmatrix}}_{\mathbf{X}_k^{\text{MA}} \in \mathbb{R}^{q+1}} = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}}_{A^{\text{MA}} \in \mathbb{R}^{(q+1)\times(q+1)}} \underbrace{\begin{bmatrix} \epsilon_{k-1} \\ \epsilon_{k-2} \\ \vdots \\ \epsilon_{k-q} \\ \epsilon_{k-q-1} \end{bmatrix}}_{\mathbf{X}_{k-1}^{\text{MA}}} + \underbrace{\begin{bmatrix} \epsilon_k \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\boldsymbol{\varepsilon}_k^{\text{AR}} \in \mathbb{R}^{q+1}}$$

where A^{MA} is the state transition matrix and $\boldsymbol{\varepsilon}_k^{\text{MA}}$ is the noise vector.

2. **Measurement Equation**: The measurement equation relates the state vector $\mathbf{X}_k^{\text{MA}}$ to the observed data Y_k. The observation equation is:

$$Y_k = \underbrace{\begin{bmatrix} 1 & \theta_1 & \theta_2 & \cdots & \theta_q \end{bmatrix}}_{C^{\text{MA}} \in \mathbb{R}^{1\times(q+1)}} \mathbf{X}_k^{\text{MA}},$$

where C^{MA} is the observation matrix. This matrix applies the MA coefficients $\theta_1, \theta_2, \ldots, \theta_q$ to the lagged noise terms and adds them to the current noise ϵ_k.

Example 5: Autoregressive Moving-Average Model as an LSSM

Let us consider the Autoregressive Moving-Average model of order $(2, 2)$, denoted as ARMA(2,2), is defined by the following equation:

$$Y_k = \phi_1 Y_{k-1} + \phi_2 Y_{k-2} + \epsilon_k + \theta_1 \epsilon_{k-1} + \theta_2 \epsilon_{k-2}, \quad \text{where } \epsilon_k \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

where ϕ_1, ϕ_2 are the autoregressive coefficients, while θ_1, θ_2 are the moving average coefficients.
To represent this ARMA(2,2) process as a Linear State-Space Model (LSSM), we use an augmented state vector that includes both past observations and past error terms.

1. **State Equation**: We define the state vector $\mathbf{X}_k \in \mathbb{R}^4$ as:

$$\mathbf{X}_k = \begin{bmatrix} Y_{k-1} & Y_{k-2} & \epsilon_{k-1} & \epsilon_{k-2} \end{bmatrix}^\intercal .$$

The state equation describes the evolution of the state vector:

$$\underbrace{\begin{bmatrix} Y_k \\ Y_{k-1} \\ \epsilon_k \\ \epsilon_{k-1} \end{bmatrix}}_{\mathbf{X}_{k+1}} = \underbrace{\begin{bmatrix} \phi_1 & \phi_2 & \theta_1 & \theta_2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} Y_{k-1} \\ Y_{k-2} \\ \epsilon_{k-1} \\ \epsilon_{k-2} \end{bmatrix}}_{\mathbf{X}_k} + \underbrace{\begin{bmatrix} \epsilon_k \\ 0 \\ \epsilon_k \\ 0 \end{bmatrix}}_{\boldsymbol{\varepsilon}_k},$$

2. **Measurement Equation**: The measurement equation relates the state vector to the observed output:

$$Y_k = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{X}_k.$$

Thus, the observed output is directly taken from the first component of the state vector:

$$Y_k = Y_k.$$

This representation captures both the autoregressive and moving average components within a unified state-space framework, enabling the application of state-space analysis techniques to the ARMA(2,2) model.

2.1 Temporal Evolution of the LSSM

In this subsection, we derive the solution to the linear state-space model. For simplicity, we consider the case of the noiseless state-space model, in which both the process and measurement noises are zero. Notice that, in the noiseless case, the state and output vectors are deterministic, assuming that the initial condition is also deterministic. The evolution of the noiseless LSSM is governed by the following equations:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \tag{4}$$

$$\mathbf{y}_k = C\mathbf{x}_k + D\mathbf{u}_k, \tag{5}$$

with given initial condition $\mathbf{x}_0$. The state equation (4) can be recursively expanded to express the state vector $\mathbf{x}_k$ as a function of the initial state $\mathbf{x}_0$ and the sequence of inputs and noise terms. Starting from the initial condition $\mathbf{x}_0$, we obtain:

$$\begin{aligned} \mathbf{x}_1 &= A\mathbf{x}_0 + B\mathbf{u}_0, \\ \mathbf{x}_2 &= A\mathbf{x}_1 + B\mathbf{u}_1 = A^2\mathbf{x}_0 + AB\mathbf{u}_0 + B\mathbf{u}_1, \\ \mathbf{x}_3 &= A\mathbf{x}_2 + B\mathbf{u}_2 = A^3\mathbf{x}_0 + A^2B\mathbf{u}_0 + AB\mathbf{u}_1 + B\mathbf{u}_2. \end{aligned}$$

In general, the state vector at time k is given by:

$$\mathbf{x}_k = A^k \mathbf{x}_0 + \sum_{i=1}^{k} A^{i-1} B \mathbf{u}_{k-i}. \tag{6}$$

This expression illustrates how the current state depends on the initial state $\mathbf{x}_0$ and the history of inputs $\mathbf{u}_i$, and the cumulative effect of the process noise $\boldsymbol{\eta}_i$.

The output equation (5) can now be used to compute the output $\mathbf{y}_k$ as a function of the initial state vector $\mathbf{x}_0$, and the sequence of inputs and noise terms. Substituting the expression (6) into the output equation, we have:

$$\boxed{\mathbf{y}_k = CA^k \mathbf{x}_0 + \sum_{i=1}^{k} CA^{i-1} B \mathbf{u}_{k-i} + D\mathbf{u}_k.} \tag{7}$$

This solution can be written in terms of the so-called **Markov parameters** of the LSSM. The k-th Markov parameter is defined as the matrix:

$$H_i = CA^{i-1}B, \text{ for } k \geq 1; \quad H_0 = D.$$

Using the Markov parameters, the summation terms in (7) can be written in terms of **discrete convolutions**.

A discrete convolution is an operation that combines two discrete temporal sequences and outputs another temporal sequence, such that each element of the output sequence is a weighted sum of properly shifted versions of the input sequences (see Fig. ?). Mathematically, the convolution of two sequences $\mathbf{a} = (a_i)_{i\geq 0}$ and $\mathbf{b} = (b_i)_{i\geq 0}$ is defined as:

$$(\mathbf{a} * \mathbf{b})_k = \sum_{i=0}^{k} a_i b_{k-i}.$$

In the context of state-space models, the summation term in (7) can be written in terms of the matrix-valued sequence of Markov parameters $\mathbf{H} = (H_i)_{i\geq 0}$ and the vector-valued sequence of inputs $\mathbf{u} = (\mathbf{u}_i)_{i\geq 0}$ as follows:

$$\boxed{\mathbf{y}_k = CA^k \mathbf{x}_0 + (\mathbf{H} * \mathbf{u})_k.}$$

Thus, the output $\mathbf{y}_k$ at time k depends on:

- The term $CA^k\mathbf{x}_0$ represents the contribution of the initial state $\mathbf{x}_0$, propagated over time.
- The convolution term $(H * \mathbf{u})_k$ captures the cumulative effect of input sequence over time.

This expression elegantly captures the contributions of the initial state and the inputs, process noise, highlighting the key role of the system matrices A, B, C, and D in shaping the output evolution.

Insert diagram.

Figure 4: Pictorial representation of the convolution of two temporal sequences.

2.2 Similarity Transformations of LSSMs

In a Linear State-Space Model (LSSM), the system's input-output behavior—how input sequence $(\mathbf{u}_k)_{k\geq 0}$ is mapped to an output sequence $(\mathbb{Y}_k)_{k\geq 0}$—encapsulates its fundamental observable dynamics. However, the internal state representation, defined by the state vector $\mathbf{x}_k$, serves as an abstract mathematical construct and is not uniquely determined. The specific realization of the internal state may vary, as multiple distinct internal representations can describe the same input-output behavior. In fact, for any given input-output relationship, there exists an infinite family of state-space realizations that yield the same external dynamics but differ in their internal state descriptions. To demonstrate this, we will introduce a formal mechanism that modifies the internal state representation while preserving the input-output mapping, thereby illustrating that the observable behavior remains invariant despite changes in the internal realization.

For simplicity, let us consider the noiseless LSSM, described by:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \quad \mathbf{y}_k = C\mathbf{x}_k + D\mathbf{u}_k,$$

where we assume that there are n hidden states, i.e., $\mathbf{x}_k \in \mathbb{R}^n$, the inputs are r-dimensional vectors $\mathbf{u}_k \in \mathbb{R}^r$, and the outputs are m-dimensional vectors $\mathbf{y}_k \in \mathbb{R}^m$. Here, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times r}$, $C \in \mathbb{R}^{m\times n}$, and $D \in \mathbb{R}^{m\times r}$ are the system matrices. We will demonstrate that there exists an infinite family of LSSMs that can realize the same input-output mapping as the system above. Specifically, for any given sequence of inputs, the same sequence of outputs can be generated using a different collection of system matrices.

To construct an LSSM that realizes the same input-output mapping as the system defined by matrices (A, B, C, D), we apply a similarity transformation to the state vector. Let $T \in \mathbb{R}^{n\times n}$ be an invertible matrix and define a new state vector $\boldsymbol{\xi}_k$ as follows:

$$\boldsymbol{\xi}_k = T^{-1}\mathbf{x}_k.$$

Substituting this into the state-space equations, we can rewrite the LSSM in terms of the new state vector $\boldsymbol{\xi}_k$ as follows:

$$\begin{aligned}\boldsymbol{\xi}_{k+1} &= T^{-1}\mathbf{x}_{k+1} = T^{-1}A\mathbf{x}_k + T^{-1}B\mathbf{u}_k = \overbrace{T^{-1}AT}^{\widetilde{A}}\boldsymbol{\xi}_k + \overbrace{T^{-1}B}^{\widetilde{B}}\mathbf{u}_k,\\ \mathbf{y}_k &= C\mathbf{x}_k + D\mathbf{u}_k = \underbrace{CT}_{\widetilde{C}}\boldsymbol{\xi}_k + D\mathbf{u}_k.\end{aligned}$$

Thus, the transformed LSSM is defined by the new system matrix $(\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{D})$, i.e., the state and output equations of the transformed LSSM are given by:

$$\boldsymbol{\xi}_{k+1} = \widetilde{A}\boldsymbol{\xi}_k + \widetilde{B}\mathbf{u}_k, \quad \mathbf{y}_k = \widetilde{C}\boldsymbol{\xi}_k + D\mathbf{u}_k.$$

Note that the initial state of the transformed LSSM is also transformed as $\mathbf{x}_0 = T\boldsymbol{\xi}_0$.

To demonstrate that the input-output mapping is invariant under this transformation, we compare the input-output behavior of the original system with that of the transformed system. Using (7) to compute the output to the new LSSM with system matrices $(\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{D})$, we compute the output $\widetilde{\mathbf{y}}_k$ as follows:

$$\begin{aligned}\widetilde{\mathbf{y}}_k &= \widetilde{C}\widetilde{A}^k\boldsymbol{\xi}_0 + \sum_{i=1}^{k} \widetilde{C}\widetilde{A}^{i-1}\widetilde{B}\mathbf{u}_{k-i} + D\mathbf{u}_k \\ &= CT(T^{-1}AT)^k\boldsymbol{\xi}_0 + \sum_{i=1}^{k} CT(T^{-1}AT)^{i-1}T^{-1}B\mathbf{u}_{k-i} + D\mathbf{u}_k. \end{aligned} \tag{8}$$

Using the following three identities: $(T^{-1}AT)^p = T^{-1}A^pT$ for all $p \geq 0$, $\boldsymbol{\xi}_0 = T^{-1}\mathbf{x}_0$ and $TT^{-1} = T^{-1}T = \mathbb{I}_n$, we can simplify (8) as follows:

$$\widetilde{\mathbf{y}}_k = CA^k\mathbf{x}_0 + \sum_{i=1}^{k} CA^{i-1}B\mathbf{u}_{k-i} + D\mathbf{u}_k = \mathbf{y}_k, \tag{9}$$

where the last equality comes from (7). Therefore, the output $\widetilde{\mathbf{y}}_k$ of the transformed LSSM with system matrices $(\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{D})$ is identical to the output $\mathbf{y}_k$ of the original LSSM with system matrices (A, B, C, D), for all invertible transformation matrices $T \in \mathbb{R}^{n\times n}$. In other words, the input-output relationship of any LSSM is preserved under this type of transformations, called **similarity transformations**, indicating that the input-output behavior of the system is invariant to changes in the internal state representation.

This similarity transformation reveals the existence of an infinite family of equivalent LSSMs, all of which produce the same input-output behavior. Therefore, when identifying the system matrices of an LSSM from input/output data, we are effectively searching for one point on a manifold[3] of solutions, where each point corresponds to a distinct internal representation of the system.

[3]A manifold is a mathematical structure that generalizes the concept of curves and surfaces to higher dimensions, providing a framework for continuous transformations between different coordinate systems or representations.

Example 6: Canonical Forms of an LSSM

Consider the following (noiseless) Linear State-Space Model (LSSM):

$$\mathbf{x}_{k+1} = \underbrace{\begin{bmatrix} a_1 & a_2 & a_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{A_c} \mathbf{x}_k + \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{B_c} u_k,$$

$$y_k = \underbrace{\begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix}}_{C_c} \mathbf{x}_k,$$

where $\mathbf{x}_k \in \mathbb{R}^3$ is the hidden state vector, $u_k \in \mathbb{R}$ represents the scalar input, and $y_k \in \mathbb{R}$ is the scalar output. The system is fully characterized by the matrices (A_c, B_c, C_c) with $D_c = 0$. This particular configuration of the system matrices is known as the **controllable canonical form** and is particularly useful when designing input signals to design input signals to drive the hidden states towards desired values.
An alternative internal representation that preserves the input-output mapping is given by:

$$\boldsymbol{\xi}_{k+1} = \underbrace{\begin{bmatrix} a_1 & 1 & 0 \\ a_2 & 0 & 1 \\ a_3 & 0 & 0 \end{bmatrix}}_{A_o = A_c^\intercal} \boldsymbol{\xi}_k + \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}}_{B_o = C_c^\intercal} u_k,$$

$$y_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}}_{C_o = B_c^\intercal} \boldsymbol{\xi}_k.$$

In this form, $\boldsymbol{\xi}_k \in \mathbb{R}^3$ represents the new state vector and the new state matrices are A_o, B_o, C_o. This new representation is called the **observable canonical form** and is particularly useful when reconstructing the hidden states from the output y_k.

2.3 Parameter Identification

Detour to tensors in Appendix. Needed for this section.

Linear State-Space Models (LSSMs) can be understood as a special case of Recurrent Neural Networks (RNNs), where the structure is restricted to linear transformations and Gaussian noise assumptions. These constraints bestow LSSMs with a simplicity and interpretability that are often elusive in the more general, nonlinear RNN architectures. Our interest in LSSM identification stems from this relationship to RNNs: while RNNs often rely on non-linear activation functions, which can complicate and obscure the learning process, LSSMs offer a transparent and mathematically tractable alternative. In this section, we frame the identification of LSSMs within the broader context of RNN learning, utilizing similar optimization techniques, yet benefiting from the clarity afforded by the linear structure. By leveraging LSSMs as a simplified framework, we aim to introduce identification methods that are commonly applied to nonlinear RNNs, providing a more accessible entry point before transitioning to more complex scenarios.

Although classical techniques for identifying the parameters of a Linear State-Space Model (LSSM) are well established and efficient [1, 2, 4], we introduce a modern optimization framework that parallels methods commonly used for Recurrent Neural Networks (RNNs). By focusing on a noiseless LSSM, we simplify the identification process by assuming the absence of process and measurement noise. This assumption leads to a more tractable estimation problem, yet the framework outlined here provides a foundation for more advanced techniques that incorporate noise.

The goal is to estimate the LSSM's system matrices (A, B, C, D) along with the initial condition $\mathbf{x}_0$, collectively forming the parameter set $\boldsymbol{\theta} = (A, B, C, D, \mathbf{x}_0)$. These parameters will be optimized using methodologies commonly applied in the training of Recurrent Neural Networks (RNNs). The available data consist of an input sequence $\mathbf{u}_{0:L} = (\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_L)$ and its corresponding output sequence $\mathbf{y}_{0:L} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_L)$. The identification process can be outlined in the following steps:

1. **Initialization:** The first step in the identification process is the initialization of the system matrices A, B, C, D and the initial state $\mathbf{x}_0$. In practice, initialization is often done using small random values, or by applying domain-specific heuristics. A judicious choice of initial values can significantly improve the convergence speed of the optimization process. For instance, initializing the matrix A with eigenvalues inside the unit circle ensures the stability of the state evolution, which is particularly important when propagating the hidden states over long time horizons.

2. **Loss Function:** In the noiseless case, the identification problem is formulated as minimizing the prediction error between the observed output $\mathbf{y}_k$ and the predicted output $\widehat{\mathbf{y}}_k$, which is generated by the LSSM. The predicted output at time step k, $\widehat{\mathbf{y}}_k$, is obtained from the recursive state-space equations:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \quad \widehat{\mathbf{y}}_k = C\mathbf{x}_k + D\mathbf{u}_k.$$

   The prediction error at each time step is evaluated using a loss function ℓ, which, in this case, is chosen to be the squared ℓ_2-norm[4]. The error at time

[4]The squared ℓ_2-norm of a vector $\mathbf{z}$ is defined as $\|\mathbf{z}\|_2^2 = \mathbf{z}^\intercal \mathbf{z} = \sum_i z_i^2$.

step k is given by:

$$\ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k) = \|\mathbf{y}_k - \widehat{\mathbf{y}}_k\|_2^2.$$

The total loss over a time horizon of length L is defined as the average loss across all time steps:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{L+1} \sum_{k=0}^{L} \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k),$$

where $\boldsymbol{\theta} = (A, B, C, D, \mathbf{x}_0)$ denotes the parameters of the system. This loss function serves as a measure of the alignment between the model's predicted outputs and the observed data. Minimizing $\mathcal{L}(\boldsymbol{\theta})$ thus provides an estimate of the model parameters that best explain the observed system behavior.

3. **Optimization Using Backpropagation Through Time (BPTT):** The identification of the LSSM parameters is framed as an optimization problem where the goal is to find the set of parameters that minimizes the loss function $\mathcal{L}$. The method of **Backpropagation Through Time (BPTT)**, commonly used in RNN training, can be adapted for LSSMs. The optimization process involves the following steps:

   (a) **State Propagation:** The challenge in BPTT is computing the gradients of the loss function with respect to the system matrices over multiple time steps. Given the state equation:

   $$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k,$$

   the state at time $k+1$ depends linearly on the state at time k and the input $\mathbf{u}_k$. Therefore, the error gradients must be propagated both through the model's layers and across time. This allows the system to capture the temporal dependencies inherent in time-series data.

   (b) **Gradient Computation:** The gradients of the loss function with respect to the system parameters can be computed using the chain rule. For example, the gradient with respect to the system matrix A is given by:

   $$\frac{\partial \mathcal{L}}{\partial A} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{y}}_k} \frac{\partial \widehat{\mathbf{y}}_k}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial A}.$$

   Similar expressions hold for the gradient with respect to B, C, D, and the initial condition $\mathbf{x}_0$. Since the system dynamics are linear, the gradient computation is more straightforward compared to non-linear RNNs, in particular,

   $$\frac{\partial \mathcal{L}}{\partial A} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{y}}_k} \frac{\partial \widehat{\mathbf{y}}_k}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial A}.$$

(c) **Parameter Updates:** Once the gradients of the loss function with respect to the parameters $A, B, C, D, \mathbf{x}_0$ have been computed, the parameters are updated using a gradient-based optimization algorithm. Commonly used methods include Stochastic Gradient Descent (SGD) and more advanced algorithms like Adam, which adaptively adjust the learning rate based on the history of gradients.

The parameters are updated iteratively as follows:

$$A \leftarrow A - \eta \frac{\partial \mathcal{L}}{\partial A}, \quad B \leftarrow B - \eta \frac{\partial \mathcal{L}}{\partial B},$$

$$C \leftarrow C - \eta \frac{\partial \mathcal{L}}{\partial C}, \quad D \leftarrow D - \eta \frac{\partial \mathcal{L}}{\partial D}, \quad \mathbf{x}_0 \leftarrow \mathbf{x}_0 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{x}_0},$$

where η is the learning rate, which controls the step size in the optimization process. The updates are performed iteratively until the loss $\mathcal{L}$ converges to a satisfactory value, indicating that the model parameters have been successfully identified.

By assuming a noiseless system, the identification process becomes significantly simpler, as the dynamics are purely deterministic. This means that the evolution of states and outputs is governed entirely by the system matrices and input signals. Without the complexity of probabilistic uncertainties, we can apply more direct and efficient optimization techniques. However, it is important to note that the noiseless assumption may limit the model's robustness when applied to real-world data, where noise is inherently present. Nonetheless, the noiseless LSSM provides a valuable foundation for understanding the core principles of system identification before transitioning to more advanced methods that incorporate noise and uncertainty.

Gradient of the total error function:

We have the following ingredients:

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_{k-1}, \quad \widehat{\mathbf{y}}_k = C\mathbf{x}_k + D\mathbf{u}_k.$$

Assume that $\mathbf{x}_k \in \mathbb{R}^n$, $\widehat{\mathbf{y}}_k \in \mathbb{R}^n$ and $\mathbf{u}_k \in \mathbb{R}^r$ for all k.

$$\ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k) = \|\mathbf{y}_k - \widehat{\mathbf{y}}_k\|_2^2.$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{L+1} \sum_{k=0}^{L} \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k).$$

We consider $\boldsymbol{\theta}$ to be a 3-tensor vector formed by **stacking** the matrices $A, B, C, D, \mathbf{x}_0$ and **padding** the smaller matrices with zeros, so that the dimensions of all the matrices are the same. For example, consider an LSSM with a scalar input, a scalar output, and 3 hidden states, whose parameters have the following dimensions: $A \in \mathbb{R}^{3\times 3}$, $B \in \mathbb{R}^{3\times 1}$, $C \in \mathbb{R}^{1\times 3}$, $D \in \mathbb{R}^{1\times 1}$, and $\mathbf{x}_0 \in \mathbb{R}^{3\times 1}$. In what follows, we pad the matrices to match the dimension 3×3 and stack them together into a single 3-dimensional tensor of parameters. The padding and stacking operations are implemented in \texttt{PyTorch}, as follows:

```
import torch

# Define the matrices and scalar

A = torch.tensor([[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]])  # Shape (3, 3)
B = torch.tensor([[10.], [11.], [12.]])  # Shape (3, 1)
C = torch.tensor([[13., 14., 15.]])  # Shape (1, 3)
D = torch.tensor(0.)  # Scalar
x_0 = torch.tensor([[16.], [17.], [18.]])  # Shape (3, 1)

# Pad B to match shape (3, 3)
B_padded = torch.cat((B, torch.zeros(3, 2)), dim=1)

# Pad C to match shape (3, 3)
C_padded = torch.cat((C, torch.zeros(2, 3)), dim=0)

# Place the scalar D in the (1,1) position and pad to 3x3
D_padded = torch.zeros(3, 3)
D_padded[0, 0] = D

# Pad x_0 to match shape (3, 3)
x_0_padded = torch.cat((x_0, torch.zeros(3, 2)), dim=1)

# Now construct the 3x3x3 tensor using A, B_padded, and C_padded
theta = torch.stack([A, B_padded, C_padded])

print("Theta tensor:")
print(theta)
print("Theta shape:", theta.shape)
```

For example **torch.cat((x_0, torch.zeros(3, 2)), dim=1)** concatenate the 3×1 column vector $\mathbf{x}_0$ with a 3×2 matrix full of zeros along the column dimension (**dim=1**). The result of stacking all the padded matrices is a tensor $\boldsymbol{\theta} \in \mathbb{R}^{3\times 3\times 3}$.

Computing the gradient:

We want to compute

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{L+1}\sum_{k=0}^{L} \frac{\partial \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k)}{\partial \boldsymbol{\theta}}$$

Consider the dependency structure of the output $\mathcal{L}$ as a function of the model parameters $\boldsymbol{\theta} = (A, B, C, D, \mathbf{x}_0)$ and the input sequence $(\mathbf{u}_k)_{k\geq 0}$:

$$\mathcal{L}(\boldsymbol{\theta}) \leftarrow \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k) \leftarrow \widehat{\mathbf{y}}_k = C\mathbf{x}_k + D\mathbf{u}_k \leftarrow \mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_{k-1} \leftarrow \cdots$$
$$\cdots \leftarrow \mathbf{x}_{k-1} = A\mathbf{x}_{k-2} + B\mathbf{u}_{k-2} \leftarrow \cdots \leftarrow \mathbf{x}_2 = A\mathbf{x}_1 + B\mathbf{u}_1 \leftarrow \mathbf{x}_1 = A\mathbf{x}_0 + B\mathbf{u}_0$$

where the arrows indicate the directions of the dependencies and we have colored those elements in $\boldsymbol{\theta}$. These dependencies induces a **computational graph** with the form of a directed chain, which is one of the signatures of recurrent models, such as RNNs. Hence, we apply the chain rule, as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{1}{L+1}\sum_{k=0}^{L} \frac{\partial \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k)}{\partial \boldsymbol{\theta}} \\ &= \frac{1}{L+1}\sum_{k=0}^{L} \frac{\partial \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k)}{\partial \widehat{\mathbf{y}}_k}\frac{\partial \widehat{\mathbf{y}}_k}{\partial \mathbf{x}_k}\frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}}\frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{x}_{k-2}}\cdots\frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}\frac{\partial \mathbf{x}_1}{\partial \boldsymbol{\theta}}.\end{aligned}$$

Element-by-element, we obtain the following **gradient** for the squared error loss:

$$\frac{\partial \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k)}{\partial \widehat{\mathbf{y}}_k} = \frac{\partial \|\mathbf{y}_k - \widehat{\mathbf{y}}_k\|_2^2}{\partial \widehat{\mathbf{y}}_k} = 2(\widehat{\mathbf{y}}_k - \mathbf{y}_k)^\intercal,$$

and the following **Jacobian matrices**:

$$\frac{\partial \widehat{\mathbf{y}}_k}{\partial \mathbf{x}_k} = C, \quad \frac{\partial \mathbf{x}_{j+1}}{\partial \mathbf{x}_j} = A, \text{ for all } j \in [k-1].$$

The last term $\frac{\partial \mathbf{x}_1}{\partial \boldsymbol{\theta}}$ is a **Jacobian tensor** that we can break down into several tensorial components, as follows:

$$\left[\frac{\partial \mathbf{x}_1}{\partial A}\right]_{i,j_1j_2} = \frac{\partial x_{1i}}{\partial a_{j_1j_2}}, \quad \left[\frac{\partial \mathbf{x}_1}{\partial B}\right]_{i,j_1j_2} = \frac{\partial x_{1i}}{\partial b_{j_1j_2}}, \quad \left[\frac{\partial \mathbf{x}_1}{\partial C}\right]_{i,j_1j_2} = \frac{\partial x_{1i}}{\partial c_{j_1j_2}},$$
$$\left[\frac{\partial \mathbf{x}_1}{\partial D}\right]_{i,j_1j_2} = \frac{\partial x_{1i}}{\partial d_{j_1j_2}}, \quad \left[\frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}\right]_{i,j} = \frac{\partial x_{1i}}{\partial x_{0j}}.$$

Hence, we can compute the elements of $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ for each one of the tensorial components $(A, B, C, D, \mathbf{x}_0)$. For example, for the first tensorial component A in $\boldsymbol{\theta}$, we obtain:

$$\begin{aligned}
\left[\frac{\partial \mathbf{x}_1}{\partial A}\right]_{i,j_1 j_2} &= \frac{\partial\left[A\mathbf{x}_0 + B\mathbf{u}_0\right]_i}{\partial a_{j_1 j_2}} \\
&= \frac{\partial\left[A\mathbf{x}_0\right]_i}{\partial a_{j_1 j_2}} + \frac{\partial\left[B\mathbf{u}_0\right]_i}{\partial a_{j_1 j_2}} \\
&= \frac{\partial}{\partial a_{j_1 j_2}} \sum_i a_{ij} x_{0j} \\
&= \sum_i \frac{\partial a_{ij}}{\partial a_{j_1 j_2}} x_{0j} \\
&= \sum_i \delta_{i,j_1} \delta_{j,j_2} x_{0j} \\
&= \delta_{i,j_1} x_{0j_2}
\end{aligned}$$

Using the Kronecker product, we have that:

$$\begin{aligned}
\frac{\partial \mathbf{x}_1}{\partial A} &= \sum_{i,j_1 j_2} \mathbf{e}_i \otimes \mathbf{e}_{j_1} \otimes \mathbf{e}_{j_2} x_{0j_2} \\
&= \sum_{i,j_1} \mathbf{e}_i \otimes \mathbf{e}_{j_1} \otimes \left(\sum_{j_2} \mathbf{e}_{j_2} x_{0j_2}\right) \\
&= \sum_{i,j_1} \mathbf{e}_i \otimes \mathbf{e}_{j_1} \otimes \mathbf{x}_0.
\end{aligned}$$

Therefore, we have that:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial A} = \frac{2}{L+1} \sum_{k=0}^{L} (\widehat{\mathbf{y}}_k - \mathbf{y}_k)^\intercal C A^{k-1} \left(\sum_{i,j} \mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{x}_0\right)$$

$$\frac{1}{L+1} \sum_{k=0}^{L} \frac{\partial \ell(\mathbf{y}_k, \widehat{\mathbf{y}}_k)}{\partial \widehat{\mathbf{y}}_k} \frac{\partial \widehat{\mathbf{y}}_k}{\partial \mathbf{x}_k} \left(\frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}} \frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{x}_{k-2}} \cdots \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}\right) \frac{\partial \mathbf{x}_1}{\partial \boldsymbol{\theta}}$$

Similar derivations for the rest of tensorial entries of $\boldsymbol{\theta}$.

Gradient descent!!!

Appendix: Tensors

In this section, we cover computational aspects of tensors, leaving more abstract algebraic concepts such as subspaces to other references (see [3] for details).

A **d-dimensional tensor** $T \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is an array of values with d indices, where the set of integers $(n_1, \ldots, n_d)$ defines the **shape** of the tensor. The uppercase letter T denotes the tensor itself, while its entries are denoted by lowercase letters $t_{i_1 i_2 \cdots i_d}$, i.e., $[T]_{i_1 i_2 \cdots i_d} = t_{i_1 i_2 \cdots i_d}$. For instance, a vector is a 1-dimensional tensor, while a matrix is a 2-dimensional tensor.

An important example is the **delta (or identity) tensor**, defined component-wise as:

$$\delta_{i_1 i_2 \cdots i_d} = \begin{cases} 1 & \text{if } i_1 = i_2 = \cdots = i_d, \\ 0 & \text{otherwise.} \end{cases}$$

For example, the 2-dimensional identity tensor, also called the **Kronecker delta**, satisfies $\delta_{i,j} = 1$ when $i = j$; and 0 otherwise. This tensor defines the identity matrix as follows: $[\mathbb{I}_n]_{ij} = \delta_{i,j}$, for all $i, j \in [n]$.

In what follows, we use **multiindexes** (i.e., ordered sequences of indexes), such as $\mathcal{I}_p = (i_1, i_2, \ldots, i_p)$, $\mathcal{J}_q = (j_1, j_2, \ldots, j_q)$, and $\mathcal{K}_r = (k_1, k_2, \ldots, k_r)$, to index tensor elements. The concatenation of two multiindexes is another multiindex, denoted by:

$$\mathcal{I}_p, \mathcal{J}_q = (i_1, i_2, \ldots, i_p, j_1, j_2, \ldots, j_q),$$

where a *comma* is used to concatenate two index sequences. Using this notation, we index tensor elements as $[T]_{\mathcal{I}_d} = [T]_{i_1 i_2 \cdots i_d} = t_{i_1 i_2 \cdots i_d} = t_{\mathcal{I}_d}$. Furthermore, we can extend the definition of the delta tensor to sequence indexes, as follows:

$$\delta_{\mathcal{I}_d \mathcal{J}_d} = \begin{cases} 1, & \text{if } \mathcal{I}_d = \mathcal{J}_d, \\ 0, & \text{otherwise.} \end{cases}$$

List of Tensor Operations

- **Sum and Subtraction**: The sum (or subtraction) of two tensors A and B of the same shape is another tensor of the same shape, defined entry-wise:

$$(A + B)_{\mathcal{I}_d} = (A + B)_{i_1 \cdots i_d} = a_{i_1 \cdots i_d} + b_{i_1 \cdots i_d} = a_{\mathcal{I}_d} + b_{\mathcal{I}_d}.$$

- **Scalar Product**: The product of a scalar α and a tensor $T \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is another tensor, defined as the element-wise as:

$$(\alpha\, T)_{\mathcal{I}_d} = \alpha\, t_{\mathcal{I}_d}.$$

- **Hadamard Product**: The Hadamard product of two tensors A and B of the same shape is another tensor of the same shape, defined entry-wise:

$$[A \odot B]_{\mathcal{I}_d} = [A \odot B]_{i_1 \cdots i_d} = a_{i_1 \cdots i_d} \cdot b_{i_1 \cdots i_d} = a_{\mathcal{I}_d} \cdot b_{\mathcal{I}_d}.$$

- **Inner Product**: The inner product of two tensors A and B of the same shape is a scalar defined as:

$$\langle A, B \rangle = \sum_{i_1, i_2, \ldots, i_d} a_{i_1 i_2 \ldots i_d} b_{i_1 i_2 \ldots i_d} = \sum_{\mathcal{I}_d \in \mathcal{N}^d} a_{\mathcal{I}_d} b_{\mathcal{I}_d},$$

where $\mathcal{N}^d = [n_1] \times [n_2] \times \cdots \times [n_d]$ is the set of all possible multiindexes.

- **Contraction Product**: Consider two tensors:

$$A \in \mathbb{R}^{n_1 \times \cdots \times n_p \times c_1 \times \cdots \times c_r} \text{ (dimension } p + r),$$

$$B \in \mathbb{R}^{c_1 \times \cdots \times c_r \times m_1 \times \cdots \times m_q} \text{ (dimension } q + r).$$

The contraction product of these two tensors is another tensor of dimension $p + q$, defined entry-wise as:

$$\begin{aligned} (A \cdot B)_{\mathcal{I}_p, \mathcal{J}_q} &= (A \cdot B)_{i_1 \ldots i_p, j_1 \ldots j_q} \\ &= \sum_{k_1 \ldots k_r} a_{i_1 \ldots i_p, k_1 \ldots k_r} b_{k_1 \ldots k_r, j_1 \ldots j_q} \\ &= \sum_{\mathcal{K}_r \in \mathcal{C}^r} a_{\mathcal{I}_p, \mathcal{K}_r} b_{\mathcal{K}_r, \mathcal{J}_q}. \end{aligned}$$

where the summation is performed over all r indices, i.e., $\mathcal{C}^r = [c_1] \times [c_2] \times \cdots \times [c_r]$. For example, the contraction product of two 2-dimensional tensors M and N (i.e., two matrices) can be written as:

$$(M \cdot N)_{i,j} = \sum_k m_{i,k} n_{k,j},$$

which is the standard matrix product.

- **Kronecker Product**: The Kronecker product of two tensors A and B of dimensions p and q, respectively, results in a tensor of dimension $p+q$, defined entry-wise as:

$$[A \otimes B]_{\mathcal{I}_p, \mathcal{J}_q} = [A \otimes B]_{i_1 \cdots i_p, j_1 \cdots j_q} = a_{i_1 \cdots i_p} b_{j_1 \cdots j_q} = a_{\mathcal{I}_p} b_{\mathcal{J}_q}.$$

For example, the Kronecker product of two matrices M and N is a 4-dimensional tensor defined entry-wise as $[M \otimes N]_{i_1 i_2, j_1 j_2} = m_{i_1 i_2} n_{j_1 j_2}$. Furthermore, the Kronecker product can be used to express a tensor $T \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ in terms of its individual components $t_{i_1\cdots i_d}$ as follows. Let us denote by $\mathbf{e}_i^n$ the unit vector representing the i-th element in the canonical basis of $\mathbb{R}^n$, i.e., the one-hot encoded vector for the i-th element in an n-dimensional space. Hence, we can write:

$$T = \sum_{i_1\cdots i_d} t_{i_1\cdots i_d} \cdot \mathbf{e}_{i_1}^{n_1} \otimes \cdots \otimes \mathbf{e}_{i_d}^{n_d} = \sum_{\mathcal{I}_d} t_{\mathcal{I}_d} \mathbf{E}_{\mathcal{I}_d},$$

where $\mathcal{I}_d = (i_1, \cdots, i_d)$ and $\mathbf{E}_{\mathcal{I}_d} = \mathbf{e}_{i_1}^{n_1} \otimes \cdots \otimes \mathbf{e}_{i_d}^{n_d}$, which is a d-dimensional tensor full of zeros except for a 1 in the element indexed by $\mathcal{I}_d$.

Tensor Gradient

In many machine learning problems, it is necessary to compute the derivative of a tensor-valued function with respect to tensor arguments. For example, consider a function that maps a p-dimensional input tensor X to a q-dimensional output tensor, i.e., $F\colon \mathbb{R}^{n_1\times n_2\times\cdots\times n_p} \to \mathbb{R}^{m_1\times m_2\times\cdots\times m_q}$. The derivatives of a tensor-valued function $F(X)$ with respect to its tensor argument X can be arranged into a new tensor of dimensions $p+q$, denoted as the **Jacobian tensor**, defined entry-wise as:

$$[\nabla_X F]_{\mathcal{I}_p,\mathcal{J}_q} = [\nabla_X F]_{i_1\cdots i_p, j_1\cdots j_q} = \frac{\partial f_{i_1\cdots i_p}(X)}{\partial x_{j_1\cdots j_q}} = \frac{\partial f_{\mathcal{I}_p}(X)}{\partial x_{\mathcal{J}_q}},$$

where $f_{\mathcal{I}_p}(X) = f_{i_1\cdots i_p}(X) = [F(X)]_{i_1\cdots i_p} = [F(X)]_{\mathcal{I}_p}$ represents the indexed entry of the output tensor. For instance, consider the identity function of a 2-dimensional tensor X, i.e., $F(X) = X$ for $X \in \mathbb{R}^{n_1\times n_2}$. The Jacobian is then the following 4-dimensional tensor:

$$[\nabla_X X]_{\mathcal{I}_2,\mathcal{J}_2} = [\nabla_X X]_{i_1 i_2, j_1 j_2} = \frac{\partial x_{i_1 i_2}}{\partial x_{j_1 j_2}} = \delta_{\mathcal{I}_2,\mathcal{J}_2}.$$

Properties of the Tensor Jacobian

Given two tensor-valued functions with the same tensor argument, $F(X)$ and $G(X)$, of appropriate shapes, the following properties hold:

- **Linearity**: The Jacobian of a linear combination of tensor functions satisfies:

  $$\nabla_X (\alpha F + \beta G) = \alpha \nabla_X (F) + \beta \nabla_X (G),$$

  where α and β are scalars.

- **Product Rule** (also known as *Leibniz rule*): For all tensor products defined above, the Jacobian of the product satisfies:

  $$\begin{aligned} \nabla_X(F \cdot G) &= (\nabla_X F) \cdot G + F \cdot (\nabla_X G), \\ \nabla_X(F \otimes G) &= (\nabla_X F) \otimes G + F \otimes (\nabla_X G), \\ \nabla_X(F \odot G) &= (\nabla_X F) \odot G + F \odot (\nabla_X G), \end{aligned}$$

where the order of the products is important for the *contraction* and *Kronecker* products, since they are not commutative.

- **Tensor Chain Rule**: Consider two tensor functions $F(Y)$ and $Y(X)$. Then, the chain rule for the tensor Jacobian is:

$$\nabla_X F = (\nabla_Y F) \cdot (\nabla_X Y),$$

where the *contraction* product is taken over the indices of the tensor Y.

Example 8: Jacobian of Power Matrix

Consider the power function of a matrix $F(X) = X^p$. Then, both the input tensor X and the output tensor X^p have dimension 2. Since the matrix product is equivalent to the contraction product of two 2-dimensional tensors (i.e., matrices), we have that:

$$\nabla_X(X^p) = \nabla_X \left(X \cdot X^{p-1}\right) = \nabla_X (X) \cdot X^{p-1} + X \cdot \nabla_X \left(X^{p-1}\right).$$

Since $[\nabla_X X]_{\mathcal{I}_2,\mathcal{J}_2} = \delta_{\mathcal{I}_2,\mathcal{J}_2}$ is a 4-dimensional tensor and X^{p-1} is a 2-dimensional tensor, we have that:

$$\begin{aligned}\left[\nabla_X (X) \cdot X^{p-1}\right]_{\mathcal{I}_2} &= \sum_{\mathcal{K}_2} [\nabla_X X]_{\mathcal{I}_2,\mathcal{K}_2} \left[X^{p-1}\right]_{\mathcal{K}_2} \\ &= \sum_{\mathcal{K}_2} \delta_{\mathcal{I}_2,\mathcal{K}_2} \left[X^{p-1}\right]_{\mathcal{K}_2} \\ &= \left[X^{p-1}\right]_{\mathcal{I}_2}.\end{aligned}$$

Hence, we obtain the following recursion for the Jacobian of the matrix power:

$$\nabla_X(X^p) = X^{p-1} + X \cdot \nabla_X \left(X^{p-1}\right), \quad \text{with } [\nabla_X X]_{\mathcal{I}_2,\mathcal{J}_2} = \delta_{\mathcal{I}_2,\mathcal{J}_2}.$$

The solution to this recursion gives the result:

$$\nabla_X(X^p) = pX^{p-1}.$$

...

References

[1] L. Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

[2] R. Pintelon and J. Schoukens. *System identification: a frequency domain approach.* John Wiley & Sons, 2012.

[3] J. G. Simmonds. *A Brief on Tensor Analysis.* Springer Science & Business Media, 2012.

[4] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory—Implementation—Applications.* Springer Science & Business Media, 2012.