Linear Dynamical Systems

Aqib A. Syed

Course Notes for EE263

September 25, 2024

Contents

How to Cite These Notes

If you use these notes in your work, please cite them using the following BibTeX entry:

```
@misc{aasyed2024ee263notes,
  author       = {Syed, Aqib A.},
  title        = {EE 263 - Linear Dynamical Systems Course Notes},
  year         = {2024},
  institution  = {Stanford University},
  url          = {https://github.com/aasyed36/EE-263-Linear-Dynamical-Systems},
  note         = {Accessed: Month, Year}
}
```

Example usage in a LaTeX document:

```
\bibliographystyle{plain}
\bibliography{your_bib_file}
```

1 Engineering Examples

We consider a scenario where a unit mass is subject to a sequence of varying forces over discrete time intervals. Our goal is to find its final position and velocity based on the applied forces.

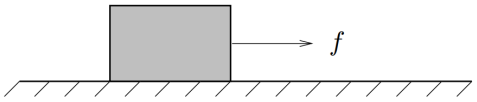


Figure 1: Unit Mass subject to Varying Forces

1.1 Mass Properties and Forces Applied

The properties of the mass are:

- Unit mass $m = 1$ kg

The initial conditions are:

- $x(0) = 0$ (initial position)
- $v(0) = 0$ (initial velocity)

The forces applied are denoted by $f(t)$, where:

$$f(t) = x_j.$$

The force applied to the mass is constant over each time interval $j - 1$ to j, where j ranges from 1 to n.

A subtle point is that the effect of each force changes over time. Each incremental force not only affects the immediate next position but also all subsequent positions. The position is accumulated by considering the velocity at each time step, which is updated by each applied force.

1.2 Final Position and Velocity

We seek to find the final position y_1 and the velocity y_2 at time $t = n$.

1.3 Matrix Representation

We represent the final position and velocity using a transformation matrix A, which maps the sequence of forces x over time to the final state of position and velocity. This matrix is used in the linear transformation of the vector x to produce the vector y.

The sequence of forces applied during each unit time interval is described by the input vector:

$$x = [x_1, x_2, \ldots, x_n]^T.$$

The matrix A transforms these forces into the final position and velocity as follows:

$$y = Ax,$$

where $y = [y_1, y_2]^T$ represents the final position y_1 and the final velocity y_2 at time $t = n$.

1.4 Force Input Model

The sequence of forces applied during each unit time interval is described by the vector $x = [x_1, x_2, \ldots, x_n]^T$, where each force x_j influences both the final position and velocity of the mass.

The matrix A captures these effects:

- The first row of A, denoted A_{1j}, represents the influence of the force x_j (applied during the interval $j-1$ to j) on the final position y_1. This considers both the direct and cumulative effects of the forces on the position.
- The second row, A_{2j}, describes the direct influence of the force x_j on the final velocity y_2.

1.5 Position and Velocity Calculations

From Newton's second law, $m\ddot{x} = f$, with $m = 1$ kg, we simplify to:

$$f = \ddot{x} = a,$$

where a is the acceleration. To calculate velocity and position, we leverage kinematic principles:

Velocity Calculation

The velocity at time τ, denoted $v(\tau)$, is the integral of the acceleration (which is equivalent to the integral of the applied force):

$$v(\tau) = \int_0^\tau f(\tau)\, d\tau.$$

At time t, the velocity of the mass is given by:

$$v(t) = v(0) + \int_0^t f(\tau)\, d\tau.$$

Since $v(0) = 0$, the final velocity is simply the sum of all forces applied over time:

$$y_2 = \sum_{j=1}^{n} x_j \implies A_{2j} = 1, \quad j = 1, \ldots, n.$$

Position Calculation

To calculate the final position, we integrate the velocity. The position at time τ, denoted $q(\tau)$, is:

$$q(\tau) = \int_0^\tau v(\tau)\ d\tau.$$

At time t, the position of the mass is given by:

$$q(t) = q(0) + \int_0^t v(\tau)\ d\tau.$$

In our scenario, the final position y_1 can be calculated as:

$$y_1 = \sum_{j=1}^{n} \left(n - j + \frac{1}{2}\right) x_j \implies A_{1j} = n - j + \frac{1}{2}, \quad j = 1, \ldots, n.$$

The factor $(n - j + \frac{1}{2})$ scales the force x_j by the time period over which its influence persists. This can be compared to the kinematic relation:

$$x = v\Delta t,$$

where, in this context, $t \sim \left(n - j + \frac{1}{2}\right)$. This accounts for the cumulative effect of each force, weighted by the duration of its impact on the position.

1.6 Matrix Construction

The matrix $A \in \mathbb{R}^{2\times n}$ is a linear transformation, which maps input forces $x_1, \cdots, x_n$ to a final velocity $v(t)$ and a final position $q(t)$. Each element $A_{ij} \in A$ gives us the coefficient of x_j in the expression for y_i To this end, A will have the following structure:

$$A = \begin{bmatrix} A_{11} & A_{12} & \ldots & A_{1j} \\ A_{21} & A_{22} & \ldots & A_{2j} \end{bmatrix}$$

The first row scales each force by its time-adjusted impact on position, while the second row reflects their cumulative impact on velocity.

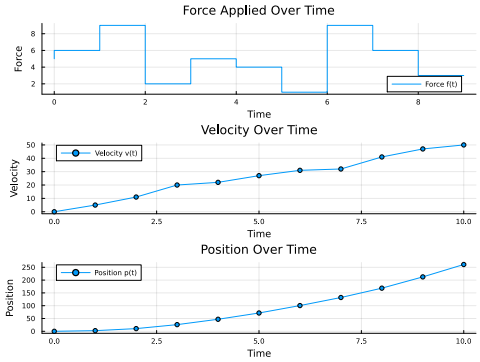


Figure 2: Force, Velocity and Position History for Unit Mass Subject to a Sequence of Arbitrary Varying Forces

2 Time Series Models and State Space Representations

2.1 Moving Average (MA) Model

The state-space representation is a way to capture the dynamics of a system over time, particularly when the system's output depends on past inputs. In the Moving Average (MA) model, the state vector captures the previous values of the input signal.

The state vector $x(k)$ consists of the previous r inputs, capturing the input history:

$$x(k) = \begin{bmatrix} u(k-1) \\ u(k-2) \\ \vdots \\ u(k-r) \end{bmatrix}$$

By storing the previous inputs in this way, the MA model uses the history of inputs to compute the current output $y(k)$. This representation allows for a compact formulation of the system's dynamics, even when the output depends on multiple past inputs.

To update the state at the next time step, we shift all elements of $x(k)$ by one time step to accommodate the new input $u(k)$, which will replace the oldest input $u(k-r)$. The new state update can be written as:

$$x(k+1) = Ax(k) + Bu(k)$$

Here:

- A is the shift matrix, which shifts each element of the state vector down by one position. This matrix has ones on the subdiagonal, enabling the shift.

- B is a vector that inserts the new input $u(k)$ at the top of the state vector during the update.

Now, we compute the output. The output at time k is a linear combination of the current and past inputs:

$$y(k) = a_0u(k) + a_1u(k-1) + \cdots + a_ru(k-r)$$

To express this in matrix form, we use:

- C as the coefficient matrix for the past inputs, which determines the influence of each input at previous time steps on the output.
- D as the coefficient matrix for the current input, which determines the influence of $u(k)$ on the output at time k.

In summary, the dynamics of input transitions over time are described by the shift matrix A and the vector B, while the conversion of inputs to outputs is governed by the matrices C and D.

Example

Say we want to understand trading volume behavior for a stock to predict the next day's trading volume.

Motivation: We can modify our trading strategies (we can figure out an estimate for how much of a stock we want to short based on this prediction).

Our input at $u(k)$ will correspond to the trading volume on day k. Suppose we want to use the past 3 days' trading volume to predict the volume on the next day. The state vector $x(k)$ consists of the past trading volumes:

$$x(t) = \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \end{bmatrix}$$

The shift matrix A will lag the trading volume by a day:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Note that the last row is zeros because the oldest value moves out of the state vector and is no longer tracked.

The vector B introduces the new trading volume at the top of the state vector:

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Suppose we assign weights to the various days; we posit that more recent trading days are more indicative of the next day's volume:

- 0.4 for $u(k-1) \Rightarrow$ most indicative
- 0.35 for $u(k-2) \Rightarrow$ slightly less indicative
- 0.25 for $u(k-3) \Rightarrow$ least impact

This idea of diminishing impact of past values as time progresses is also seen in moving average (MA) models. We have:

$$C = [0.4 \quad 0.35 \quad 0.25]$$

For simplicity, say we only care about past volumes, i.e., $D = 0$.

Finally, suppose we have the following trading volumes for the past few days (in 10^7 shares):

- Day 0 (3 days back): 200 shares $\Rightarrow x(t) = \begin{bmatrix} u(k-1) \\ u(k-2) \\ u(k-3) \end{bmatrix}$
- Day 1 (2 days back): 270 shares $\Rightarrow u(k-1) = 270$
- Day 2 (1 day back): 220 shares $\Rightarrow u(k-2) = 220$
- Day 3 (today): 230 shares $\Rightarrow u(k-3) = 230$

From this, our initial state $x(3)$ contains information about trading volumes from days 0, 1, and 2:

$$x(3) = \begin{bmatrix} 230 \\ 220 \\ 270 \end{bmatrix}$$

Our state update on day 3 is given by:

$$x(4) = Ax(3) + Bu(3)$$

$$x(4) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 230 \\ 220 \\ 270 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} 230 = \begin{bmatrix} 220 \\ 270 \\ 0 \end{bmatrix} + \begin{bmatrix} 230 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 450 \\ 270 \\ 0 \end{bmatrix} \ (\times 10^7 \text{ shares})$$

$$y(4) = Cx(4) + Du(4)$$

$$y(4) = [0.4\, 0.35\, 0.25] \begin{bmatrix} 450 \\ 270 \\ 0 \end{bmatrix} + 0 \cdot 230$$

$$y(4) = 231.5 \times 10^7 \text{ shares}$$

So our model predicts that the trading volume tomorrow (Day 4) will be $\approx 231.5 \times 10^7$ shares, based on the weighted average of the past 3 days' volumes.

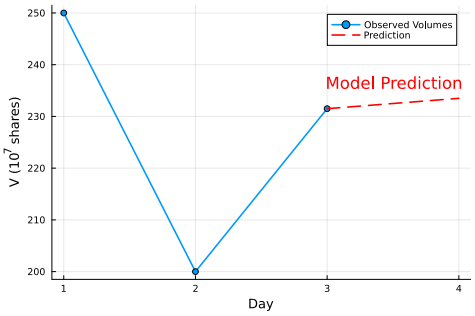


Figure 3: Volume Prediction using MA Model

2.2 Autoregressive Model (AR)

In the autoregressive (AR) model, the state vector consists of the previous outputs, making it useful for forecasting future values based on past observations. The system's dynamics can be expressed in the following state-space form:

$$x(k+1) = Ax(k) + Bu(k)$$

Where:

- A is the shift matrix, which shifts the previous outputs by one period. This matrix plays the role of "memory", ensuring that each output is shifted down by one index, while the oldest output is discarded.
- B is the input matrix, introducing new information from the input $u(k)$. In the pure AR model, the input matrix often influences the output directly through D, but may have an indirect role on the state vector depending on how the input is modeled.

The state vector $x(k)$ captures the history of the system by holding the previous p outputs, where p is the order of the AR model:

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ \vdots \\ y(k-p) \end{bmatrix}$$

The dimensionality of the state vector is $p \times 1$, corresponding to the p past output values. As new data arrives, the system updates its state by shifting these output values and incorporating new information through the input $u(k)$, if present.

The matrix A, also called the state transition or shift matrix, evolves the state by shifting each past output value down one index. The most recent output $y(k-1)$ moves to the top of the vector, while the oldest output $y(k-p)$ is forgotten. Mathematically, this shift is represented as:

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

The input matrix B introduces the input $u(k)$, allowing it to influence the current state. Depending on the specific model, B might be a vector that directly impacts the state, or it may influence the output more directly through the matrix D.

The output at time k, denoted by $y(k)$, is computed from the state vector using the equation:

$$y(k) = Cx(k+1) + Du(k)$$

Where:

- $C = [a_1, a_2, \ldots, a_p]$ contains the AR coefficients, which describe how each past output influences the current output. These coefficients weigh the previous output values stored in the state vector.
- D introduces a direct dependency on the input $u(k)$. If $D \neq 0$, the current input can impact the output immediately.

Additionally, a noise term $\epsilon(k)$ could be included in the output equation to model the effect of disturbances or measurement noise. This is common in practical AR models when dealing with real-world data.

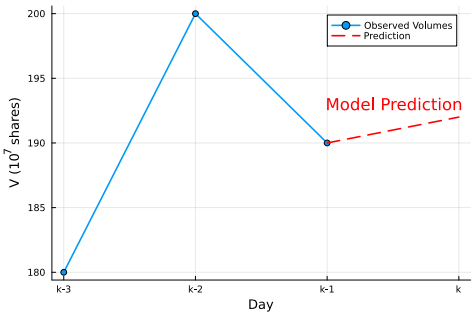


Figure 4: Volume Prediction using AR(3) Model

2.3 Autoregressive Moving Average (ARMA) Model

In the Autoregressive Moving Average (ARMA) model, the state vector combines both past outputs (for the AR component) and past inputs (for the MA component), enabling us to forecast future outputs by considering both the memory of past observations and the effect of previous inputs. The system dynamics can be expressed in the following state-space form:

$$x(k+1) = Ax(k) + Bu(k)$$

Where:

- A is the state transition matrix, which captures the memory of both the AR and MA components. It shifts the previous outputs and inputs by one period.
- B is the input matrix, introducing new information from the input $u(k)$ into the state vector. This matrix ensures that the new input enters the appropriate position in the state, affecting both current and future outputs.

The state vector $x(k)$ consists of both past output values from the AR part and past input values from the MA part:

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ \vdots \\ y(k-p) \\ u(k-1) \\ u(k-2) \\ \vdots \\ u(k-q) \end{bmatrix}$$

Here, the first p components correspond to the past outputs, while the next q components capture the past inputs. The dimensionality of the state vector is $(p+q) \times 1$, reflecting the combined memory of the system.

The state transition matrix A has a block structure that shifts both the AR and MA components:

$$A = \begin{bmatrix} \text{AR Block} & 0 \\ 0 & \text{MA Block} \end{bmatrix}$$

The AR block shifts past output values, and the MA block shifts past input values, preserving the relevant history of the system.

The input matrix B introduces the current input $u(k)$, which directly influences the output at the next time step and is then shifted down in future steps:

$$B = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The output at time k, denoted by $y(k)$, is computed from the state vector as follows:

$$y(k) = Cx(k) + \epsilon(k)$$

Where:

- $C = [a_1, a_2, \ldots, a_p, b_0, \ldots, b_q]$ contains both the AR coefficients $a_1, \ldots, a_p$ and the MA coefficients $b_0, \ldots, b_q$, determining how past outputs and inputs influence the current output.
- $\epsilon(k)$ is the noise term at time k.

In summary, the ARMA model in this form captures the system's dynamics by leveraging both the autoregressive (AR) and moving average (MA) components, evolving the state vector based on past outputs, past inputs, and new inputs through the state-space framework. This allows for a concise and structured way to model and forecast the output of the system.

Regarding the input shift:

Suppose our MA model consists of 3 past input variables to obtain the current output, i.e., our model is of the form:

$$y(k) = b_1 u(k-1) + b_2 u(k-2) + b_3 u(k-3) + \epsilon(k)$$
$$i = 1, 2, 3, \quad q = 3$$

Our current state is given by:

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ u(k-1) \\ u(k-2) \\ u(k-3) \end{bmatrix}$$

And our shift matrix will only shift input terms:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Explanation:

- The AR part: $y(k-1)$ moves to where $y(k-2)$ was previously.
- The MA part, $u(k-3)$, resets to zero, and $u(k-1)$ and $u(k-2)$ are shifted by one position.

And we will apply $Bu(k)$:

$$Bu(k) = \begin{bmatrix} u(k) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Consequently, $x(k+1)$ becomes:

$$x(k+1) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ u(k) \\ u(k-1) \\ u(k-2) \end{bmatrix}$$

Comparing $x(k)$ with $x(k+1)$, we observe that in our state update, the input $u(k)$ shifts into the position formerly held by $u(k-1)$. This shift allows us to utilize the new input $u(k)$ in computing the next output $y(k+1)$.

3 Linear Algebra Review

3.1 Bases vs. Coordinate Systems

Definition (Basis): A basis of a vector space is a set of linearly independent vectors spanning the entire space.
It is just a collection of vectors such that any vector in the space can be expressed as a unique linear combination of these basis vectors.

- **Linear independence**: No basis vector can be written as a linear combination of the others.
- **Spanning**: Any vector in the space can be expressed using only the vectors of the basis.

Example

In $\mathbb{R}^3$, the set

$$\{(1,0,0),(0,1,0),(0,0,1)\}$$

forms the standard basis. Another basis for $\mathbb{R}^3$ could be:

$$\{(1,0,0),(1,1,0),(1,1,1)\}.$$

Example

Consider the standard basis $\{e_1, e_2\}$ in $\mathbb{R}^2$, where:

$$e_1 = (1,0), \quad e_2 = (0,1).$$

Suppose we consider any vector $v = (a,b) \in \mathbb{R}^2$ and we want to express said vector using the basis $\{e_1, e_2\}$.

To begin, we represent v as a linear combination:

$$v = ae_1 + be_2.$$

This becomes:

$$v = a(1,0) + b(0,1)$$

which simplifies to:

$$v = (a,0) + (0,b) = (a,b).$$

In the above, a and b are the coordinates of v formed by e_1 and e_2. This gives us an idea of how much of each basis vector is needed to construct v. Moreover, the coefficients a and b are scalars that scale the basis vectors e_1 and e_2, respectively.

Interpretation

Using the standard basis $\{e_1, e_2\}$, we are able to "reach" any vector $(a, b) \in \mathbb{R}^2$. That is, any vector in the 2D plane can be expressed as a combination of these basis vectors.

Example

In $\mathbb{R}^3$, the vectors $\mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and $\mathbf{a}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ are linearly independent and they form a basis for the span, which is the same as the x_1-x_3 plane (and is a 2D subspace of $\mathbb{R}^3$).

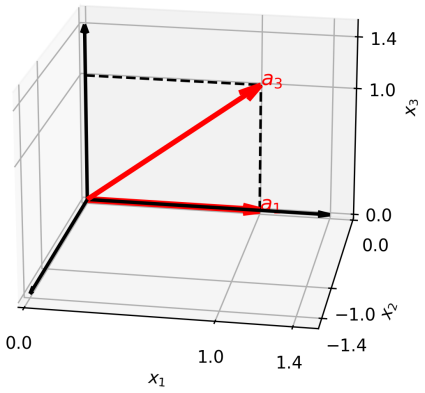


Figure 5: Linearly Independent Vectors in $\mathbb{R}^3$

This subspace is $\mathbf{span}\{\mathbf{a}_1, \mathbf{a}_3\}$

The vectors span the x_1-x_3 plane and $\mathbf{span}\{\mathbf{a}_1, \mathbf{a}_3\}$ forms the plane where $x_2 = 0$.

Definition (Coordinate System)

In a vector space, a coordinate system is one that uses coordinates, i.e., a tuple, to uniquely determine the position of a point, or other geometric element within the space.
It is just a way to define a unique position in space using a tuple of scalars that are functions of some basis.

The Relationship between these two Concepts

The basis defines a framework for a coordinate system. Given a basis, any vector in the space can be uniquely represented as a linear combination of the basis vectors. The coefficients of this linear combination are the coordinates of the vector with respect to the chosen basis. Thus, starting from a basis, we construct vectors by specifying coordinates, which serve as the weights in the

linear combination of the basis vectors. This process provides a systematic way to represent vectors in terms of the underlying basis.

In summary: **Basis** $\rightarrow$ **Coordinates** $\rightarrow$ **Vector**.

Example

Consider the basis vectors, which represent the standard basis in $\mathbb{R}^2$:

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We construct a vector from these basis vectors using the corresponding scalar coefficients:

$$\mathbf{v} = 2e_1 + 3e_2$$

This results in:

$$= 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Thus, the vector $\mathbf{v}$ has coordinates $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$ in the standard basis.

3.2 Dimension of a Subspace

Given a subspace S, the dimension of S is the number of vectors in a basis for S.

Fact: Any two bases for a subspace have the same number of elements.

Supppose we have that

$$d = \dim S$$

This implies any set of vectors that spans S has at least d elements.

3.3 What is meant by "Form a Basis For"

When we say a set of vectors form a basis for a vector space, we mean:

1. They span the space.
2. They are the minimal set necessary to have linear independence.
3. We can construct the entirety of the vector space with these vectors and their linear combinations.

In short, these vectors are enough to construct every aspect of the space.

3.4 Change of Coordinates

We now explore how a vector can be expressed in different coordinate systems with respect to different bases.

The standard basis vectors in $\mathbb{R}^n$ are given by:

$$\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$$

Each vector $\mathbf{e}_i$ in this standard basis has a 1 in its i-th component and 0's in all other components. This structure allows each vector $\mathbf{e}_i$ to isolate a single coordinate axis, meaning that $\mathbf{e}_i$ represents a unit vector pointing along the i-th axis of the coordinate system. Any vector in $\mathbb{R}^n$ can be expressed as a linear combination of these basis vectors, with the coefficients representing how much of each axis is involved in constructing the vector.

We have that any vector $x \in \mathbb{R}^n$ may be represented as a linear combination of these standard basis vectors:

$$x = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \ldots + x_n\mathbf{e}_n$$

Here, x_i (coordinate of x in the standard basis) is how much of each basis vector is needed to construct x.

We now consider an alternative basis $\{t_1, t_2, \ldots, t_n\}$, which is any set of n linearly independent vectors spanning $\mathbb{R}^n$.

In this new basis, the vector x can be expressed as a linear combination of the basis vectors:

$$x = \tilde{x}_1 t_1 + \tilde{x}_2 t_2 + \cdots + \tilde{x}_n t_n$$

Here, $\tilde{x}_i$ are the coordinates of x in the new basis $\{t_1, t_2, \ldots, t_n\}$ and represent the components of x relative to the new basis. These coordinates $\tilde{x}$ describe the different proportions of the basis vectors t_i needed to reconstruct x. Each coordinate $\tilde{x}_i$ tells us how much of the corresponding vector t_i contributes to the overall vector x, similar to how the standard coordinates x_i work in the standard basis.

To convert between the standard basis and the new basis, we use a transformation matrix T. This matrix T has columns corresponding to the new basis vectors $t_1, t_2, \ldots, t_n$, expressed in terms of the standard basis. Using T, we can express the vector x in the standard basis as:

$$x = T\tilde{x}$$

In the above, $T\tilde{x}$ gives us the vector x in terms of the standard basis, based on the new coordinates $\tilde{x}$. Conversely, if we want to express x in the new basis, we compute the inverse of the transformation matrix T. By applying the inverse, we convert the coordinates of x from the standard basis back into the new basis:

$$\tilde{x} = T^{-1}x$$

In summary, the transformation matrix T allows us to translate between the standard basis and the new basis, converting coordinates from one representation to another. This is particularly useful when working in different coordinate systems or changing the basis in which a vector is expressed.

3.5 A Closer Look at the Transformation Matrix

When working in vector spaces, it is often useful to switch between different bases depending on the problem at hand. This is particularly helpful when solving problems in different coordinate systems, such as rotational motion in polar versus Cartesian coordinates.

The goal is to understand how to represent a vector in different coordinate systems—that is, how the same vector can be expressed in different bases.

Transformation Matrices

Let T be a transformation matrix that converts the coordinates of a vector from a new basis back to the standard basis. In other words, T takes coordinates of a vector expressed in the new basis and transforms them into coordinates with respect to the standard basis.

Conversely, T^{-1} converts coordinates from the standard basis to the new basis. Thus, T allows us to compute using the basis most appropriate for the problem, and then switch between different bases as needed.

Two Bases in $\mathbb{R}^n$

Consider two different bases for a vector space $\mathbb{R}^n$:

1. The standard basis $\{e_1, e_2, \ldots, e_n\}$, consisting of the unit vectors in each coordinate direction.
2. An arbitrary basis $\{t_1, t_2, \ldots, t_n\}$, which is any set of n linearly independent vectors that also spans $\mathbb{R}^n$.

Our goal is to express vectors in terms of either of these two bases and understand how they are related.

Expressing One Basis in Terms of Another

Each new basis vector t_i in $\{t_1, t_2, \ldots, t_n\}$ can be expressed as a linear combination of the standard basis vectors $\{e_1, e_2, \ldots, e_n\}$. Specifically, we can write:

$$t_i = a_{i1}e_1 + a_{i2}e_2 + \cdots + a_{in}e_n$$

where $a_{i1}, a_{i2}, \ldots, a_{in}$ are the coordinates of t_i in the standard basis. These coefficients a_{ij} indicate how much of each standard basis vector e_j contributes to t_i.

Constructing the Transformation Matrix

The transformation matrix T, which converts coordinates from the new basis to the standard basis, is constructed by placing the coordinates of each t_i as columns in T:

$$T = \begin{bmatrix} t_1 & t_2 & \dots & t_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Each column corresponds to one of the t_i vectors expressed in terms of the standard basis. This matrix T provides the complete relationship between the new basis and the standard basis.

Using the Transformation Matrix

Now, suppose we have a vector x expressed in the new basis $\{t_1, t_2, \dots, t_n\}$ as:

$$x = x_1 t_1 + x_2 t_2 + \cdots + x_n t_n$$

We want to express this vector in the standard basis. Let z be the vector of coordinates of x in the new basis:

$$z = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Then, the coordinates of x in the standard basis can be obtained by multiplying T by z:

$$x = Tz$$

Thus, the transformation matrix T allows us to convert vector coordinates between the new basis and the standard basis.

Summary

The transformation matrix T is a key tool for switching between bases:

- T converts vector coordinates from a new basis back to the standard basis.
- T^{-1} converts coordinates from the standard basis to the new basis.
- Each column of T corresponds to the coordinates of a new basis vector t_i in terms of the standard basis.
- To transform a vector from the new basis to the standard basis, we simply multiply the vector of its new basis coordinates by T.

This process provides a convenient way to shift between coordinate systems or bases, enabling us to solve problems in the most appropriate basis.

Expressing One Basis in Terms of Another

To express the new basis vectors $t_1, t_2, \ldots, t_n$ in terms of the standard basis $\{e_1, e_2, \ldots, e_n\}$, we can use a linear combination of the standard basis vectors. Each vector t_i in the new basis can be written as:

$$t_i = a_{i1}e_1 + a_{i2}e_2 + \cdots + a_{in}e_n$$

where $a_{i1}, a_{i2}, \ldots, a_{in}$ are the coordinates of t_i in the standard basis. The coefficients a_{ij} tell us how much of each standard basis vector e_j contributes to t_i.

Constructing the Transformation Matrix

The transformation matrix T that converts coordinates from the new basis to the standard basis is built by placing the coordinates of the t_i's as columns in T:

$$T = \begin{bmatrix} t_1 & t_2 & \ldots & t_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}$$

Each column corresponds to one of the t_i vectors expressed in the standard basis. The matrix T fully encodes how to move between the new basis and the standard basis.

Using the Transformation Matrix

Now, suppose we have a vector x expressed in the new basis $\{t_1, t_2, \ldots, t_n\}$ as:

$$x = x_1t_1 + x_2t_2 + \cdots + x_nt_n.$$

Our task is to express this vector in the standard basis. To do this, we use the transformation matrix T. Let z be the vector of coordinates of x in the new basis:

$$z = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Then, we can transform x to the standard basis by multiplying T by z:

$$x = Tz.$$

This gives us the coordinates of x in the standard basis.

Thus, T serves as the "middleman" that allows us to convert vector representations between different bases.

Summary

- T converts vector coordinates from a new basis back to the standard basis.
- T^{-1} converts coordinates from the standard basis to the new basis.
- Each column of T corresponds to the coordinates of a new basis vector t_i expressed in terms of the standard basis.
- To transform a vector from the new basis to the standard basis, we multiply the vector of its new basis coordinates by T.

This approach gives us a way shift between different coordinate systems and bases, allowing us to solve problems in the most convenient basis.

3.6 Similarity Transform

In many cases, we need to re-express a matrix A relative to a different set of basis vectors. The goal is to make A compatible with the language of the new basis—think of it like updating a framework for new software. This process is known as a *similarity transformation.*

Recall from earlier, using our new basis $t_1, t_2, \ldots, t_n$, we construct a matrix T, which reflects a basis change from the standard basis to the new basis. Specifically, each new basis vector t_i is expressed as a linear combination of the standard basis vectors:

$$t_i = a_{i1}e_1 + a_{i2}e_2 + \cdots + a_{in}e_n, \quad e_n = \mathbf{span}\{e_1, e_2, \ldots, e_n\} = \mathbf{span}(\mathbb{R}^n)$$

The transformation matrix T is constructed as follows:

$$T = \begin{bmatrix} t_1 & t_2 & \ldots & t_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}$$

Transformation of Vectors

Now, to express vectors x and y with respect to the new basis, we leverage the transformation matrix T. We obtain the coordinates of these vectors in the new basis as:

$$\tilde{x} = T^{-1}x, \quad \tilde{y} = T^{-1}y$$

This allows us to transform the vectors from the standard basis to the new basis.

Similarity Transform

Next, consider a situation where x and y were related in the standard basis by a linear transformation A, such that:

$$y = Ax, \quad A \in \mathbb{R}^{n \times n}, \quad x, y \in \mathbb{R}^n$$

When transforming A into the new basis, we need to ensure the relationship between x and y is preserved. To do this, we apply the similarity transformation to A, yielding:

$$\tilde{A} = T^{-1}AT$$

The matrix $\tilde{A}$ is the transformed version of A with respect to the new basis, and this is called the *similarity transformation*.

Key Properties of the Similarity Transformation

It is important to note that a similarity transformation does not change the fundamental properties of the matrix A. Specifically:

- *Eigenvalues are preserved*:

$$\text{eig}(A) = \text{eig}(\tilde{A})$$

- *Determinant is preserved*:

$$\det(A) = \det(\tilde{A})$$

- *Trace is preserved*:

$$\text{tr}(A) = \text{tr}(\tilde{A})$$

Thus, while the representation of the matrix A may change depending on the basis, its key properties such as eigenvalues, determinant, and trace remain invariant under similarity transformations.

3.7 Vector Projection onto Subspaces

The projection of a vector $x \in \mathbb{R}^n$ onto a subspace $\mathcal{V} \subseteq \mathbb{R}^n$ is essentially the process of finding the closest vector in $\mathcal{V}$ to the vector x.

Visualization

Consider a subspace $\mathcal{V}$ as a flat surface or a line within a higher-dimensional space, and let x be a point or vector in this space, not necessarily lying on $\mathcal{V}$. The projection $\text{proj}_{\mathcal{V}}(x)$ is analogous to the shadow that x casts onto $\mathcal{V}$ when a light is shone perpendicular to $\mathcal{V}$. This "shadow" or projection is the point on $\mathcal{V}$ closest to x.

Mathematically, $\text{proj}_{\mathcal{V}}(x)$ is the vector in $\mathcal{V}$ such that the difference $x - \text{proj}_{\mathcal{V}}(x)$ is orthogonal to every vector in $\mathcal{V}$. In other words, the vector $x - \text{proj}_{\mathcal{V}}(x)$ points directly away from $\mathcal{V}$, making a right angle with every vector in $\mathcal{V}$.

Example

Say we have $v \in \mathcal{V}$ and let $v = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ and $x = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$. The projection of x onto the subspace spanned by v is given by:

$$\text{proj}_v(x) = \frac{x^\top v}{v^\top v} v$$

In this case, the projection of the vector x onto the subspace $\mathcal{V}$ is:

$$\text{proj}_v(x) = \frac{(4)(2) + (1)(2)}{(2)^2 + (2)^2} v = \frac{10}{8} v = 1.25 \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix}$$

The projection of the vector x onto the subspace $\mathcal{V}$ looks like:

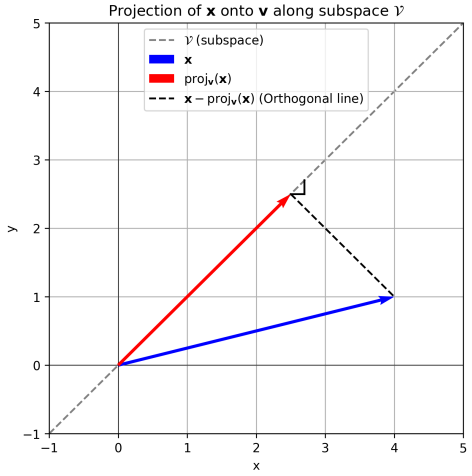


Figure 6: Projection of the vector x onto the subspace $\mathcal{V}$. Notice that the residual term $x - \text{proj}_{\mathcal{V}}(x)$ is orthogonal to every vector $v \in \mathcal{V}$

$$x - \text{proj}_v(x) = y \perp v \quad \text{for all } v \in \mathcal{V}$$

Thus, $x - \text{proj}_v(x) \perp v$, meaning that $(x - \text{proj}_v(x))^\top v = 0$.

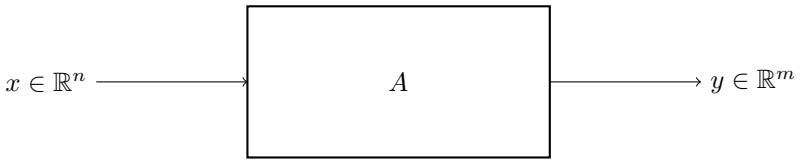
Why is this important?

The concept of projecting a vector onto a subspace is fundamental in many areas of applied mathematics, including signal processing, machine learning, and numerical optimization. In these fields, it is often necessary to approximate a point (or vector) by its closest counterpart in a lower-dimensional subspace. This allows us to reduce the dimensionality of data while preserving important characteristics.

In optimization, for instance, projections are used to enforce constraints by ensuring that a solution stays within a feasible set. Similarly, in least squares problems, the projection corresponds to finding the best approximation of a vector by a set of basis vectors, minimizing the error in the least-squares sense.

The projection operator is crucial in solving linear systems, particularly when the system of equations is overdetermined. In such cases, projecting the solution onto the subspace spanned by the columns of the matrix gives us the least-squares solution.

3.8 Range



Consider a matrix $A \in \mathbb{R}^{m \times n}$, which defines a linear transformation from $\mathbb{R}^n$ (the input space) to $\mathbb{R}^m$ (the output space).

The range of a matrix A, denoted $\mathcal{R}(A)$, is the set of all possible vectors $y \in \mathbb{R}^m$ that can be written as a linear combination of the columns of A. Formally, it is defined as:

$$\mathcal{R}(A) = \{y \in \mathbb{R}^m \mid y = Ax \text{ for some } x \in \mathbb{R}^n\}.$$

Each vector $x \in \mathbb{R}^n$ can be viewed as scaling the columns of A, resulting in the output y. Thus, the range of A is spanned by the columns of A, or equivalently, it is the column space of A.

Rank: The rank of the matrix A, denoted **rank**(A), is the dimension of the range of A, i.e., the number of linearly independent columns of A. It gives the maximum number of linearly independent vectors that A can output.

Example

Suppose $A \in \mathbb{R}^{m \times n}$ has the columns:

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

and let $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$. Then:

$$y = Ax = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n.$$

This expresses y as a linear combination of the columns of A. The set of all such possible linear combinations constitutes the range of A, or equivalently, the span of the columns of A.

Interpretation

The operation Ax can be viewed as scaling each column of A by the corresponding entry of x and summing the result. The range of A tells us the set of all vectors that can be reached through this process, which is central to understanding the behavior of the linear transformation defined by A.

Example

Consider the matrix $A \in \mathbb{R}^{2 \times 2}$ with columns a_1 and a_2, and a vector $x \in \mathbb{R}^2$. The matrix A and vector x are given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The output vector $y = Ax \in \mathbb{R}^2$ is given by the matrix-vector multiplication:

$$y = Ax = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

In this example:

$$y_1 = a_{11}x_1 + a_{12}x_2,$$

which tells us how much the first coordinates of a_1 and a_2 contribute to the first coordinate of the resulting vector y.

Similarly,

$$y_2 = a_{21}x_1 + a_{22}x_2,$$

which tells us how much the second coordinates of a_1 and a_2 contribute to the second coordinate of the resulting vector y.

Interpretation

Each entry of the output vector y is a linear combination of the corresponding coordinates of a_1 and a_2, weighted by the components of x. This shows how the matrix A transforms the input vector x through a combination of its columns.

3.9 Null Space

Consider the matrix $A \in \mathbb{R}^{m \times n}$. The nullspace of A, denoted as $\mathcal{N}(A)$, is the set of all vectors $x \in \mathbb{R}^n$ such that:

$$Ax = 0.$$

These are the input vectors that, when transformed by the matrix A, yield the zero vector in $\mathbb{R}^m$. nullspace vectors represent the *lack of action* of A.

The nullspace provides valuable insights into the solutions of the homogeneous equation $Ax = 0$. Formally:

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}.$$

If $\text{null}(A) = \{0\}$, this implies that the only solution to $Ax = 0$ is the trivial solution. This typically occurs when A has full rank, meaning $Ax = b$ has a unique solution for any non-zero b.

However, if $\text{null}(A)$ contains more than just the zero vector, then the equation $Ax = 0$ may have many solutions. To fully describe the nullspace, we must determine a set of basis vectors that span $\mathcal{N}(A)$.

The dimension of the nullspace, known as the nullity of A, is given by $n - \textbf{rank}(A)$.

Example

Suppose we have a sensor A that detects frequencies of input signals, represented by $x \in \mathbb{R}^n$. The output readings of these signals are denoted by $y = Ax$.

For example, we may want to measure frequencies in $\mathbb{R}^3$ (corresponding to, say, $\hat{i}, \hat{j}, \hat{k}$ axes). Let:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}.$$

Each column of A measures frequencies along each axis of the input space. In this case, $\mathcal{N}(A)$ includes all vectors x for which the sensor A yields no output, i.e., signals to which the system fails to respond (they are "inaudible" with respect to the sensor).

Example

Consider the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$. We can write this matrix as:

$$A = \begin{bmatrix} a_1 & a_2 \end{bmatrix}, \quad \text{where} \quad a_2 = 2a_1.$$

Clearly, a_2 is a multiple of a_1, so the columns are linearly dependent, and the matrix does not have full rank. We compute:

$$\mathbf{rank}(A) = 1.$$

To solve $Ax = 0$, consider:

$$\begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This gives us the following system of equations:

$$x_1 + 2x_2 = 0, \quad 3x_1 + 6x_2 = 0.$$

Since both equations are equivalent, we are left with just one equation: $x_1 + 2x_2 = 0$. Therefore, the nullspace is spanned by vectors that satisfy this equation.

Thus, the nullspace $\mathcal{N}(A)$ is non-trivial and can be expressed as:

$$\mathcal{N}(A) = \{x \in \mathbb{R}^2 \mid x_1 + 2x_2 = 0\} = \mathbf{span}\left(\begin{bmatrix} -2 \\ 1 \end{bmatrix} \right).$$

Example

Now consider the matrix $B = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$. The rank of this matrix is 1 because:

$$b_2 = 2b_1, \quad \text{so} \quad \mathbf{rank}(B) = 1.$$

This matrix also does not have full rank, and solving $Bx = 0$ gives a non-trivial solution.

Thus, the nullspace of B is spanned by non-trivial vectors satisfying the equation $x_1 + x_2 = 0$. So, the nullspace of B is:

$$\mathcal{N}(B) = \{x \in \mathbb{R}^2 \mid x_1 + x_2 = 0\}.$$

Example

Consider the matrix $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, which is the identity matrix. Clearly, the rank of this matrix is 2, and it is full rank:

$$\textbf{rank}(C) = 2.$$

Since C has full rank, the only solution to $Cx = 0$ is the trivial solution $x = 0$, meaning the nullspace is:

$$\mathcal{N}(C) = \{0\}.$$

3.10 Rank

The rank of a matrix $A \in \mathbb{R}^{m \times n}$ is the dimension of its column space, which is the space spanned by the linearly independent columns of A. Therefore, we can write:

$$\textbf{rank}(A) = \textbf{dim}(\mathcal{R}(A)),$$

where $\mathcal{R}(A)$ is the column space of A. This also equals the number of linearly independent columns of A.

Furthermore, the rank of A is also equal to the dimension of its row space, as the linear independence of rows equates to the linear independence of the corresponding columns of A^T (the transposed matrix). Thus:

$$\textbf{rank}(A) = \textbf{dim}(\text{row space of } A).$$

The maximum number of linearly independent columns (or rows) of A is given by the smaller dimension of the matrix, i.e., $\min(m, n)$, where $A \in \mathbb{R}^{m \times n}$.

3.11 Determining the Maximum Number of Linearly Independent Columns (or Rows)

Case 1: Row-Limited (More Columns than Rows, $n > m$)

When there are more columns than rows, i.e., $n > m$, each column is an m-dimensional vector, meaning the vector space is spanned by m dimensions. Therefore, the maximum number of linearly independent columns is m.

This implies that although we have n columns, only up to m of them can be linearly independent. Any additional columns must be linear combinations of the first m independent columns.

Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}.$$

Here, we observe that $a_3 = a_1 + 2a_2$ (verify this!). Hence, although we have 3 columns, only 2 can be linearly independent because the columns reside in a 2D space (spanned by the two rows).

Case 2: Column-Limited (More Rows than Columns, $m > n$)

When there are more rows than columns, i.e., $m > n$, each row is an n-dimensional vector. The row space is spanned by the n columns of the matrix. Therefore, the maximum number of linearly independent rows is bounded by n, the dimension of the space they inhabit.

Thus, for $m > n$, the matrix can only have up to n linearly independent rows.

Example

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Here, we observe that $b_3 = b_1 + 2b_2$. Hence, although we have 3 rows, only 2 can be linearly independent since they reside in a 2D space (spanned by the 2 columns).

Conclusion:

In both cases, whether we are row-limited or column-limited, the number of linearly independent vectors (columns or rows) is determined by the smaller dimension of the matrix. Therefore, for $A \in \mathbb{R}^{m \times n}$, we have that $\mathbf{rank}(A) \leq \mathbf{min}(m, n)$, which is the maximum number of linearly independent rows or columns that the matrix can have.

3.12 Conservation of Dimension

We now consider the rank-nullity theorem, which provides a clear way of understanding how the input dimensions are transformed by a matrix. Specifically, consider again the matrix $A \in \mathbb{R}^{m \times n}$. The rank-nullity theorem states that:

$$\mathbf{rank}(A) + \mathbf{dim}(\mathcal{N}(A)) = n$$

or equivalently,

$$\mathbf{dim}(\mathcal{R}(A)) + \mathbf{dim}(\mathcal{N}(A)) = n,$$

where:

- $\mathbf{rank}(A)$ (or $\mathbf{dim}(\mathcal{R}(A))$) tells us how many dimensions in the output space $\mathbb{R}^m$ are reached by the transformation A. This represents the number of independent directions in the output space that are influenced by the input.
- $\mathbf{dim}(\mathcal{N}(A))$ is the dimension of the nullspace of A, which corresponds to the number of input directions that are "crushed" to zero by the transformation A. These are the input directions that have no effect on the output.

The key insight from the rank-nullity theorem is that every dimension of the input space $\mathbb{R}^n$ is accounted for: either it contributes to the output, or it is nullified by mapping to zero. No input dimension is "lost" or unaccounted for. Each input vector is either mapped to a nonzero output or is part of the nullspace, which is mapped to zero.

This concept is closely related to the idea that a subspace and its orthogonal complement span the entire space. Specifically, if S is a subspace of $\mathbb{R}^n$, then:

$$\mathbf{dim}(S) + \mathbf{dim}(S^\perp) = n$$

where $S^\perp$ is the orthogonal complement of S. The dimensions of S and $S^\perp$ together account for all the dimensions of $\mathbb{R}^n$, just as the dimensions of the range and nullspace of A account for all input dimensions.

Indeed, there are some interesting properties between a subspace S and its orthogonal complement $S^\perp$. One such property is that their only common element is the zero vector; that is,

$$S \cap S^\perp = \{0\}.$$

Moreover, the union of the two sets spans the entire space $\mathbb{R}^n$, so

$$S \cup S^\perp = \mathbb{R}^n.$$

This is illustrated in the figure below.

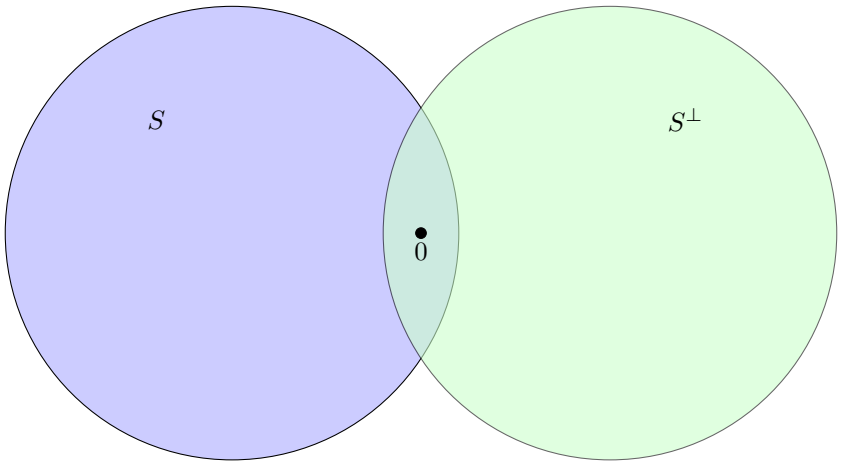


Figure 7: Illustration of S and $S^\perp$ in $\mathbb{R}^n$

4 QR Factorization

4.1 Gram-Schmidt Procedures

The Gram-Schmidt procedure converts a set of linearly independent vectors into an orthonormal set. Given independent vectors $a_1, a_2, \ldots, a_n \in \mathbb{R}^m$ as input, we want to generate a set of orthonormal vectors $q_1, q_2, \ldots, q_n$ that spans the same subspace as the original vectors $a_1, a_2, \ldots, a_n$.

That is to say, any vector which we can express as a linear combination of the original vectors can also be represented as a linear combination of the orthonormal vectors:

$$V = \alpha_1 a_1 + \alpha_2 a_2 + \cdots + \alpha_r a_r = \beta_1 q_1 + \beta_2 q_2 + \cdots + \beta_r q_r \quad \text{for } r \leq n.$$

Properties

i) The set $q_1, q_2, \ldots, q_n$ is orthonormal.

ii) Each pair of vectors is orthogonal: $q_i^T q_j = 0$ for $i \neq j$.

iii) Each vector is a unit vector: $\|q_i\| = 1$.

Span Preservation

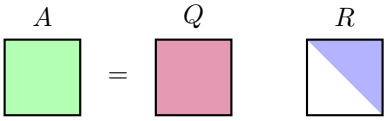
A linear combination of $a_1, a_2, \ldots, a_n$, producing any vector in the subspace, may equivalently be represented using $q_1, q_2, \ldots, q_n$:

$$\mathbf{span}(a_1, a_2, \ldots, a_n) = \mathbf{span}(q_1, q_2, \ldots, q_n).$$

To this end, suppose $A \in \mathbb{R}^{m\times n}, m \geq n$. There exists an orthogonal matrix Q and upper triangular matrix R such that:

$$A = QR.$$

Case 1: $A \in \mathbb{R}^{n\times n}$ (Square Matrix)

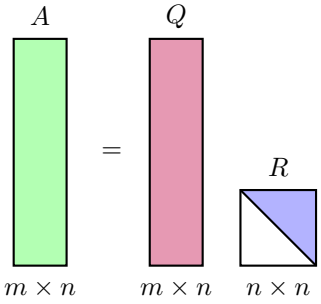


In this case, A is a square matrix. The QR factorization produces an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n\times n}$.

Case 2: $A \in \mathbb{R}^{m\times n}, m \gg n$ ("Tall and Skinny")

Choice 1:

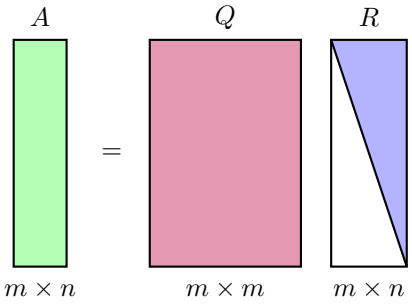
2-1)



In this case, A is tall and skinny. The QR factorization produces an orthogonal matrix $Q \in \mathbb{R}^{m\times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n\times n}$.

Choice 2:

2-2)



In this case, we again factor A, but now Q is square with dimensions $m \times m$, and R is upper triangular with dimensions $m \times n$.

4.2 Using QR to Solve Least-Squares Problems

We can solve least-squares problems with QR factorization, using either form of the factorization depending on whether the matrix is square or tall and skinny.

For $A \in \mathbb{R}^{m \times n}, m \gg n$, we have:

$$\text{minimize } \|Ax - b\|^2.$$

This minimization problem arises from trying to find the vector x that best approximates b in the range of A, and QR decomposition provides a way to simplify solving it.

QR decomposition factorizes a matrix A into a product of an orthogonal matrix Q and an upper triangular matrix R. That is:

$$A = QR,$$

where $Q \in \mathbb{R}^{m \times n}$ is comprised of orthonormal columns (if all columns are normalized), and $R \in \mathbb{R}^{n \times n}$ is an upper triangular matrix. This decomposition helps simplify the least-squares minimization problem into a form that can be solved efficiently.

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_{nn} \end{bmatrix}.$$

Note: If $\textbf{\textit{rank}}(A) = n$ *(i.e., full column rank), then* R *is invertible.*

The QR decomposition can be obtained via the Gram-Schmidt process, which transforms the columns of A into an orthogonal set. This involves subtracting the projection of each column onto the space spanned by the previous columns.

To form Q, we normalize these orthogonal vectors, ensuring that they form an orthonormal set. The coefficients from the orthogonalization process (i.e., lengths and projections) are then used to populate R. These coefficients correspond to the projections of the columns of A onto the orthogonal basis vectors formed in Q:

r_{ij} = component of the projection of column j of A onto the orthogonal basis vector q_i.

Columns of Q as a Basis for $\mathcal{R}(A)$

The columns of Q form an orthonormal basis for the column space $\mathcal{R}(A)$, which comprises all possible linear combinations of the column vectors of A. Since $\mathcal{R}(A) = \textbf{span}(a_1, a_2, \dots, a_n)$, it is a subspace of $\mathbb{R}^m$.

Now, because Q is formed using the columns of A, the columns of Q span the same subspace as the columns of A. Thus, they form a basis for $\mathcal{R}(A)$.

4.3 Gram-Schmidt Procedure

Normalizing the First Vector

We begin by setting $\tilde{q}_1 := a_1$ and then normalize it to form the first orthonormal vector:

$$q_1 := \frac{\tilde{q}_1}{\|\tilde{q}_1\|}$$

This ensures q_1 has unit length.

Orthogonalizing and Normalizing the Second Vector

Next, we process the second vector a_2. We remove the component of a_2 in the direction of q_1, making it orthogonal to q_1:

$$\tilde{q}_2 := a_2 - (q_1^T a_2) q_1$$

We then normalize $\tilde{q}_2$ to obtain the second orthonormal vector:

$$q_2 := \frac{\tilde{q}_2}{\|\tilde{q}_2\|}$$

Orthogonalizing and Normalizing Subsequent Vectors

For each subsequent vector a_i (where $i = 3, \ldots, n$), we subtract the projections onto all previously computed orthonormal vectors $q_1, \ldots, q_{i-1}$ to make the vector orthogonal to them. This is done step by step for each vector a_i:

$$\tilde{q}_i := a_i - \sum_{j=1}^{i-1} (q_j^T a_i) q_j$$

Finally, we normalize $\tilde{q}_i$ to form the orthonormal vector q_i:

$$q_i := \frac{\tilde{q}_i}{\|\tilde{q}_i\|}$$

Expressing a_i as a Linear Combination

Each vector a_i can now be written as a linear combination of the orthonormal vectors $q_1, q_2, \ldots, q_i$. Specifically, for each i, we have:

$$a_i = (q_1^T a_i) q_1 + (q_2^T a_i) q_2 + \cdots + (q_{i-1}^T a_i) q_{i-1} + \|\tilde{q}_i\| q_i$$

This shows that each a_i can be decomposed into its components along the directions of the orthonormal vectors.

Interpretation of $\|\tilde{q}_i\|$

The quantity $\|\tilde{q}_i\|$ represents the norm of the component of a_i that is orthogonal to the span of the previously computed vectors $q_1, \ldots, q_{i-1}$. The direction of this orthogonal component is captured by q_i, making it a unit vector in that direction.

4.4 Orthogonal Decomposition

Orthogonal decomposition of $\mathbb{R}^m$ entails splitting a space into two mutually orthogonal subspaces, $\mathcal{R}(A)$ and $\mathcal{N}(A^T)$.

We start with $A = QR$, which leads to the following relation:

$$A^T = R^T Q^T$$

Given that $Q^T Q = I$, we can now relate the two subspaces:

$$\mathcal{R}(Q_1) = \mathcal{N}(A^T)$$

Properties of Vectors in the Subspaces

All possible outputs of the vectors in the subspace spanned by Q_2 are equivalent to the vectors which are orthogonal to the row vectors of A. The vectors in the subspace spanned by Q_2 (which is part of $Q \perp \mathcal{R}(A)$) form an orthonormal basis for $\mathcal{N}(A^T)$.

Any vector $x \in \mathcal{N}(A^T)$ can be constructed using a linear combination of all vectors in Q_2.

The matrix Q contains columns where:

- Q_1: consists of orthonormal vectors spanning the column space of A. - Q_2: consists of vectors orthogonal to all vectors in Q_1.

4.5 Orthogonal Decomposition

Recall from the full QR factorization:

$$Q = [Q_1 \; Q_2]$$

- Q_1 contains orthonormal vectors spanning the column space of A.
- Q_2 contains vectors orthogonal to all vectors in Q_1.

In the scenario where A is tall and skinny, i.e., rank-deficient ($m > n$), the columns of Q_1 do not span $\mathbb{R}^m$ (they span the subspace of dimension $\mathbf{rank}(A)$). The vectors in Q_2 form an orthonormal basis for the remaining subspace of $\mathbb{R}^m$.

We then have the decomposition:

$$A = Q_1 R \quad \text{where} \quad Q_1^T Q_1 = I, \quad Q_1^T Q_2 = 0$$

Orthogonal Basis for $\mathbb{R}^m$

The combined matrix $[Q_1\ Q_2]$ forms an orthonormal basis for $\mathbb{R}^m$. If $A \in \mathbb{R}^{m\times n}$, then:

$$[Q_1\ Q_2] \in \mathbb{R}^{m\times m}$$

For any vector $z \in \mathcal{N}(A^T)$, we have $Q_1^T z = 0$, as the vectors are orthogonal to each other:

$$Q_2^T Q_1 = 0$$

If R is upper triangular and invertible (and if A is full rank), then $Q_2^T z = 0$ implies that $z \in \mathcal{N}(A^T)$, corresponding to the columns of Q_2.

Decomposition of a Vector in $\mathbb{R}^m$

Moreover, any vector $y \in \mathbb{R}^m$ can be uniquely decomposed as:

$$y = z + w$$

where $z \in \mathcal{R}(A)$ (spanned by Q_1) and $w \in \mathcal{N}(A^T)$ (spanned by Q_2).

4.6 Implication of Being Mutually Orthogonal

Mutually orthogonal vectors or subspaces are those which are orthogonal to one another.

I. Mutually Orthogonal Vectors

Two vectors are mutually orthogonal if their dot product is zero. Consider two vectors $u, v \in \mathbb{R}^n$:

$$u^T v = 0 \quad \Rightarrow \quad u \perp v$$

II. Mutually Orthogonal Subspaces

Two subspaces are mutually orthogonal if every vector in one subspace is orthogonal to every vector in the other subspace. Consider two subspaces U and W of a vector space $\mathbb{R}^n$.

U and W are mutually orthogonal if $u \in U$ and $w \in W$ implies:

$$u^T w = 0$$

We recall that the subspaces corresponding to $\mathcal{R}(A)$ and $\mathcal{N}(A^T)$ are said to be mutually orthogonal. This means that for all $v \in \mathcal{R}(A)$ and $u \in \mathcal{N}(A^T)$, we have:

$$u^T v = 0$$

4.7 Implication of $\mathbf{rank}(A) = \mathbf{rank}(A^T)$

Transposing a matrix does not affect linear independence among its vectors.

This implies that if we had m linearly independent columns in A, we would have m linearly independent rows in A^T.

Proof:

This follows directly from the Rank-Nullity Theorem.

5 Least-Squares

Suppose we have an overdetermined system, meaning we have more equations than unknowns. In such a system, $A \in \mathbb{R}^{m \times n}$ where $m > n$.

In these cases, we are not able to determine an exact solution $y = Ax$, since there are more constraints (equations) than unknowns.

We introduce the residual, which measures the disparity between the actual output and the model's prediction:

$$r = Ax - y$$

where:

- y represents the observed outputs
- Ax represents the predicted outputs

Least Squares Solution

To determine the least squares solution, i.e., the best possible solution given an overdetermined system, we want to find x that minimizes the norm of the residual:

$$\text{minimize} \quad \|Ax - y\|^2$$

Expanding this expression:

$$(Ax - y)^T (Ax - y)$$

This simplifies to:

$$= (x^T A^T Ax - 2(A^T y)^T x + y^T y)$$

Normal Equations

We find the minimizer by taking the gradient and setting it to zero:

$$\nabla_x \|Ax - y\|^2 = 0$$

$$2A^T Ax - 2A^T y = 0$$

From this, we derive the normal equations:

$$A^T Ax = A^T y$$

Thus, the least squares solution is:

$$x_{\text{ls}} = (A^T A)^{-1} A^T y$$

This solves $y = Ax_{\text{ls}}$ if $y \in \mathcal{R}(A)$.

5.1 Presudoinverse

If A is skinny and full rank, we introduce the pseudoinverse:

$$A^\dagger = (A^T A)^{-1} A^T$$

We know:

$$A^\dagger A = (A^T A)^{-1} A^T A = I$$

Thus, $A^\dagger$ is the left inverse of A when A is skinny and full rank.

Handling Ill-Conditioned Matrices and Rank Deficiency

The pseudoinverse works particularly well for ill-conditioned matrices, providing more stable results. Moreover, when A does not have full (column) rank, $A^\dagger = (A^T A)^\dagger$, and the pseudoinverse is well-equipped to handle the rank deficiency of A.

Applications of the Pseudoinverse

We leverage the pseudoinverse in scenarios where our matrix A is:

I. Not Square

Suppose we have an underdetermined system (more unknowns than equations), i.e., $m < n$. Since A lacks the number of rows to uniquely obtain a solution, we end up with an infinite number of solutions.

However, since the pseudoinverse yields the smallest Euclidean norm solution, we consequently end up with the minimum norm solution.

II. Overdetermined System ($m > n$)

Next, suppose we have an overdetermined system. Since A is not square, it would not have an inverse.

In this case, the pseudoinverse yields the solution to:

$$\text{minimize} \quad \|Ax - y\|^2$$

Thus, we get the optimal solution in the least squares sense.

III. Square but Singular

For scenarios where A is square but singular (i.e., $\det(A) = 0$ and A^{-1} does not exist), we would then leverage the pseudoinverse to obtain the least squares solution.

6 Least-Norm Solutions of Underdetermined Equations

Consider a linear system of equations:

$$y = Ax \quad \text{where} \quad y \in \mathbb{R}^m,\ A \in \mathbb{R}^{m \times n},\ x \in \mathbb{R}^n$$

This system is said to be underdetermined if $m < n$, meaning there are more unknowns than equations. In other words, the matrix A has more columns than rows, which implies that there are infinitely many solutions for x.

With more variables than constraints, the system does not have a unique solution, but rather an entire set of solutions that span an infinite space of possible values for x.

We assume that A is full rank, meaning $\mathbf{rank}(A) = m$, which ensures that the rows of A are linearly independent and that no redundancy exists among them. Under this assumption, there is at least one solution $x \in \mathbb{R}^n$ that satisfies:

$$Ax = y$$

However, since the system is underdetermined, this solution is not unique. The existence of more unknowns than equations guarantees that we can add any vector from the nullspace of A to a particular solution and still satisfy the equation $Ax = y$.

Solutions to Underdetermined Systems

In underdetermined systems, where $m < n$, there are infinitely many possible solutions for x. The set of all solutions can be expressed as:

$$x = x_p + z$$

Where:

- x_p is a particular solution to the equation $Ax = y$,
- z is any vector in the nullspace of A, i.e., $Az = 0$.

The nullspace of A, also called the kernel of A, consists of all vectors that map to the zero vector when multiplied by A. Since $m < n$, the dimension of the nullspace of A is $n - m$, which corresponds to the degrees of freedom in the system. Thus, while we can always find a particular solution x_p, the full solution set is given by the particular solution plus any vector from the nullspace.

In summary, underdetermined systems with full-rank matrices A have an infinite number of solutions, each of which can be written as a particular solution plus a component from the nullspace of A.

General Form of the Solution

The set of all solutions to the underdetermined system can be written as:

$$x = x_p + z \quad \text{where} \quad z \in \mathcal{N}(A)$$

Here, x_p is a particular solution that satisfies $Ax_p = y$, and z is any vector in the nullspace of A, satisfying $Az = 0$. By choosing different values of z, we obtain different solutions for x, forming a family of solutions.

The nullspace $\mathcal{N}(A)$ represents the set of vectors that lie in the "freedom" or "degrees of freedom" of the solution. These vectors do not change the outcome of $Ax = y$, but they affect the overall solution x.

The dimension of the nullspace is given by:

$$\textbf{dim}\, \mathcal{N}(A) = n - m$$

This dimension represents the difference between the number of columns and rows of A, which corresponds to the number of independent directions (degrees of freedom) we have in the solution space.

Choosing the Vector z

To select z, we typically impose additional constraints or optimize certain properties of the solution. One common choice is to select z such that the norm of x is minimized:

$$\text{minimize} \quad \|x\|_2$$

This approach yields the minimum norm solution, which is the solution with the smallest possible Euclidean norm among all solutions. This is particularly useful when we need to find a solution that balances competing objectives or when additional physical constraints are not explicitly imposed by the equation $Ax = y$.

6.1 Least Norm Solution

We aim to find the solution x with the smallest norm that still satisfies $Ax = y$. The formulation provided by $z \in \mathcal{N}(A)$ finds the point in the intersection of the solution space $\{x \mid Ax = y\}$ closest to the origin in $\mathbb{R}^n$, which minimizes $\|x\|_2$.

$$\begin{aligned} \text{minimize} \quad & \|x\|_2 \\ \text{subject to} \quad & Ax = y \end{aligned}$$

This gives the smallest magnitude solution, where $x_{\text{ln}} \perp \mathcal{N}(A)$, ensuring orthogonality between the nullspace and the particular solution x_{ln}.

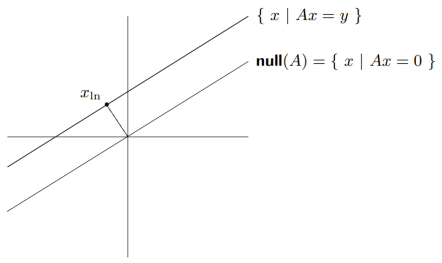


Figure 8: Illustration of the least norm solution x_{ln} for the system $Ax = y$. The set of all possible solutions lies on the affine subspace $\{x \mid Ax = y\}$, while the nullspace of A, $\mathbf{null}(A) = \{x \mid Ax = 0\}$, forms a subspace through the origin. The least norm solution x_{ln} is the point in $\{x \mid Ax = y\}$ closest to the origin, and is orthogonal to the nullspace of A.

Underdetermined Linear Systems

Given an underdetermined linear system $Ax = y$, where $A \in \mathbb{R}^{m \times n}$ with $m < n$ (i.e., more unknowns than equations), and assuming A has full row rank $\mathbf{rank}(A) = m$, we seek to find the particular solution x_{ln} that satisfies:

$$Ax = y \quad \text{and} \quad x_{\text{ln}} = \arg\min_x \|x\|_2$$

We solve this using the following steps.

Formulating the Lagrangian

We can solve this constrained optimization problem by formulating the Lagrangian:

$$L(x, \lambda) = \frac{1}{2}\|x\|_2^2 + \lambda^T(Ax - y)$$

We now apply the first-order necessary conditions (KKT conditions) to minimize the Lagrangian:

1. Gradient with respect to x:

$$\nabla_x L(x, \lambda) = x + A^T\lambda = 0 \quad \Rightarrow \quad x = -A^T\lambda$$

2. Gradient with respect to λ:

$$\nabla_\lambda L(x, \lambda) = Ax - y = 0 \quad \Rightarrow \quad A(-A^T\lambda) = y$$

Substitute $x = -A^T\lambda$ into this equation to obtain λ^*:

$$AA^T\lambda^* = y$$

Thus, solving for λ^*:

$$\lambda^* = (AA^T)^{-1}y$$

Solving for x^*

Now that we have λ^*, substitute it back into the expression for x:

$$x^* = -A^T\lambda^* = -A^T(AA^T)^{-1}y$$

Therefore, the minimum norm solution is given by:

$$x^* = A^T(AA^T)^{-1}y$$

This is the particular solution that minimizes $\|x\|_2$, which corresponds to the smallest norm solution.

Interpretation

The solution $x^* = A^T(AA^T)^{-1}y$ is a form of the left pseudoinverse of A. Since A has full rank, $(AA^T)^{-1}$ exists, and the pseudoinverse provides the solution that minimizes the residual $\|Ax - y\|_2$ in the least squares sense.

The pseudoinverse effectively maps $y \in \mathbb{R}^m$ onto an m-dimensional subspace of $\mathbb{R}^n$, satisfying the equation $Ax = y$ while also minimizing the Euclidean norm $\|x\|_2$. This is the solution of the smallest possible magnitude that satisfies the equation of the hyperplane.

7 On the Duality between $\mathcal{R}(A)$ and $\mathcal{N}(A)$ and between Least-Squares and Least-Norm

Duality between $\mathcal{R}(A^T)$ and $\mathcal{N}(A^T)$

The spaces $\mathcal{R}(A)$ and $\mathcal{N}(A)$ are not dual spaces in the functional analysis sense. However, they are intrinsically tied and demonstrate symmetrically opposite behaviors related to matrix operations involving A and A^T.

This relates to the fundamental subspaces corresponding to A and A^T. Let's look at $\mathcal{R}(A^T)$ and $\mathcal{N}(A^T)$:

- $\mathcal{R}(A^T)$ is the row space of A, comprising all linear combinations of the rows of A.
- $\mathcal{N}(A^T)$ is the nullspace of A^T, comprising all vectors x satisfying $A^T x = 0$.

We now note that:

$$\mathcal{R}(A) \perp \mathcal{N}(A^T)$$

Also:

$$\mathcal{R}(A^T) \perp \mathcal{N}(A)$$

This duality between these two spaces means that for all $v \in \mathcal{R}(A)$ and $w \in \mathcal{N}(A^T)$:

$$v^T w = 0$$

Similarly, for all $v \in \mathcal{R}(A^T)$ and $z \in \mathcal{N}(A)$:

$$v^T z = 0$$

Now, consider the Rank-Nullity Theorem:

$$\mathbf{dim}(\mathcal{R}(A)) + \mathbf{dim}(\mathcal{N}(A)) = n$$

or equivalently:

$$\mathbf{rank}(A) + \mathbf{dim}(\mathcal{N}(A)) = n$$

For A^T, the same applies:

$$\mathbf{rank}(A^T) + \mathbf{dim}(\mathcal{N}(A^T)) = m$$

Moreover, since $\mathbf{rank}(A) = \mathbf{rank}(A^T)$, we see a direct relationship between the dimensions of the range and nullspaces, which are complementary. Thus, we get a full picture of how the matrix transforms the vector space.

7.1 Least-Squares and Least-Norm Duality

Least-squares (LS) and least-norm (LN) solutions are considered primal and dual forms of an optimization problem.

The primal problem in LS seeks to minimize the residual between the predicted and actual outcomes under a linear transformation.

For a given matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, the least-squares problem is:

$$\text{minimize} \quad \|Ax - b\|^2$$

which leads to the normal equations:

$$\left(A^T A\right) x = A^T b$$

Thus, x_{ls} is the vector of unknowns that minimizes the squared residuals.

The dual problem, the least-norm (LN) solution, seeks the solution that minimizes the norm of x subject to the constraint $Ax = y$. This can be written as:

$$\begin{aligned} \text{minimize} \quad & \|x\| \\ \text{subject to} \quad & Ax = y \end{aligned}$$

By solving this using the Lagrangian formulation, we obtain:

$$x_{\text{ln}} = A^T (A A^T)^{-1} y$$

Thus, x_{ln} provides the solution of smallest magnitude that satisfies $Ax = y$, and this is orthogonal to the nullspace of A.

Here, the goal is to minimize the norm of x, subject to the constraint that $Ax = b$. Geometrically, this corresponds to finding the point on the hyperplane $\{x \mid Ax = b\}$ that is closest to the origin.

The duality lies in the fact that LS focuses on minimizing the residual, while LN minimizes the norm of the solution.

The dual of the Least-Squares (LS) problem is the Least-Norm (LN) problem, which seeks to find the solution to the linear system with the smallest norm. This arises naturally in underdetermined systems where $m < n$ and A has full column rank.

Key Differences between LS and LN

- LS minimizes the error in the solution space, by minimizing the residual $\|Ax - b\|_2^2$, finding the best fit when the system is overdetermined.
- LN minimizes the norm of the solution $\|x\|_2$ while satisfying the linear constraints exactly, which is crucial in underdetermined systems.

In summary, LS focuses on reducing the residual error, while LN focuses on finding the smallest solution that meets the linear constraint.

7.2 Matrix Crimes

For a detailed discussion on common mistakes in matrix algebra, often referred to as "Matrix Crimes", see the notes by Stephen Boyd in Matrix Crimes.

7.3 Lagrange Multipliers

Consider the following constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0 \end{aligned}$$

where:

- $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function.
- $g : \mathbb{R}^n \to \mathbb{R}^p$ is the constraint function.

We now extend the notion of optimality conditions to constrained optimization problems. The Lagrangian is given by:

$$L(x, \lambda) = f(x) + \lambda^T g(x)$$

where $\lambda \in \mathbb{R}^p$ is the Lagrange multiplier associated with the constraint $g(x) = 0$.

First-Order Necessary Conditions

The necessary (but not sufficient) optimality conditions for this constrained problem are the first-order necessary conditions (F.O.N.C.), which include:

1. The gradient of the Lagrangian with respect to x:

$$\nabla_x L(x, \lambda) = 0$$

2. The satisfaction of the constraint:

$$g(x) = 0$$

These conditions are collectively known as the Karush-Kuhn-Tucker (KKT) conditions. They are required at any optimal point.

Condition (2) is straightforward — any feasible x must satisfy the constraint $g(x) = 0$.

Condition (1) is more interesting. It involves finding the point where the gradient of the Lagrangian vanishes, which gives us a critical point for optimization.

Solving the KKT Conditions

When solving constrained optimization problems, we attempt to solve the KKT conditions. If x^* is an optimal point, it must satisfy both conditions:

$$\nabla_x L(x^*, \lambda^*) = 0 \quad \text{and} \quad g(x^*) = 0$$

By solving these equations, we can find candidates for optimal solutions in constrained optimization problems.

Example

Consider the following linearly constrained least-squares problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|Ax - b\|^2 \\ \text{subject to} \quad & Cx = d \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and $C \in \mathbb{R}^{p \times n}$.

Lagrangian

The Lagrangian for this problem is:

$$L(x, \lambda) = \frac{1}{2}\|Ax - b\|_2^2 + \lambda^T (Cx - d)$$

Expanding the Lagrangian:

$$L(x, \lambda) = \frac{1}{2}x^T A^T Ax - b^T Ax + \frac{1}{2}b^T b + \lambda^T Cx - \lambda^T d$$

KKT Conditions

The KKT conditions are derived by taking the gradients of the Lagrangian with respect to x and λ.

1. Gradient with respect to x:

$$\nabla_x L(x, \lambda) = A^T Ax - A^T b + C^T \lambda = 0$$

This simplifies to:

$$A^T (Ax - b) + C^T \lambda = 0$$

2. Gradient with respect to λ:

$$\nabla_\lambda L(x, \lambda) = Cx - d = 0$$

Hence, the KKT conditions are:

$$\begin{aligned} A^T Ax - A^T b + C^T \lambda &= 0 \\ Cx - d &= 0 \end{aligned}$$

Solving the KKT System

We now have a set of $n + p$ linear equations in $n + p$ variables, which can be written in matrix form as:

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

Thus, the solution (x^*, λ^*) can be found by solving this linear system, yielding the unique minimizer:

$$\begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

7.4 How we have $n + p$ linear equations in $n + p$ variables

In the previous example, we had $n + p$ linear equations in $n + p$ variables.The system of equations from the KKT conditions includes both the n variables $x \in \mathbb{R}^n$ (the decision variables) and the p variables $\lambda \in \mathbb{R}^p$ (the Lagrange multipliers associated with the constraints).

The KKT conditions are:

$$\nabla_x L(x, \lambda) = A^T Ax - A^T b + C^T \lambda = 0$$

This provides n equations from the gradient of the Lagrangian with respect to x:

$$A^T (Ax - b) + C^T \lambda = 0$$

Additionally, we have the p equations from the constraint:

$$Cx - d = 0$$

The KKT system in matrix form is:

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

Block Matrix Interpretation

- A^TA is $n \times n$ and represents the second derivative of $\frac{1}{2}\|Ax - b\|_2^2$ with respect to x.
- C^T is $n \times p$, the transposed Jacobian of the constraint $Cx = d$ with respect to x.
- The C^T block couples the Lagrange multipliers λ to the decision variables x.
- C is $p \times n$ and is the Jacobian of the constraints with respect to x.
- The 0 block is $p \times p$; the Lagrangian does not involve terms related to products of different Lagrange multipliers.

Note that there is no interaction between different Lagrange multipliers within the constraints.

7.5 KKT Conditions for Constrained Optimization

The use of Lagrange multipliers, originally applied for equality constraints, can be extended to handle problems with inequality constraints.

KKT conditions provide a systematic way to handle scenarios involving both:

- Equality constraints (conditions we must meet exactly)
- Inequality constraints (conditions we must not exceed).

The KKT conditions help identify optimal points (i.e., solutions) for these constrained optimization problems.

Consider an optimization problem of the form:

$$\begin{aligned} &\min_x f(x) \\ \text{subject to} \quad &g_i(x) \leq 0, \quad i = 1, \ldots, m \\ &h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned}$$

The KKT conditions are:

1. Stationarity: Ensures that the gradient of the Lagrangian with respect to x is zero at the optimum. The Lagrangian is defined as:

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{j=1}^{p} \lambda_j h_j(x)$$

The stationarity condition is:

$$\nabla_x L(x, \lambda, \mu) = 0$$

2. Primal Feasibility: Requires that the constraints are satisfied at the solution:

$$g_i(x) \leq 0, \quad h_j(x) = 0$$

3. Dual Feasibility (only for inequality constraints): Requires that the Lagrange multipliers for the inequality constraints are non-negative. This ensures that we cannot "pay" to violate these constraints:

$$\mu_i \geq 0$$

4. Complementary Slackness: For each inequality constraint, either the constraint is active (i.e., $g_i(x) = 0$) or the corresponding Lagrange multiplier must be zero. This means that if a constraint is not active (i.e., the inequality is strict), the corresponding multiplier must be zero:

$$\mu_i g_i(x) = 0, \quad \forall i$$

So, for a strict inequality constraint $g_i(x) < 0$, complementary slackness requires $\mu_i = 0$, and for $g_i(x) = 0$, complementary slackness allows $\mu_i \geq 0$.

8 Recursive Estimation

Suppose we have a growing set of measurements, and the objective is to solve a least squares (LS) problem, which is in a form we often encounter in statistics and data fitting problems.

Let $\tilde{a}_i^T x = y_i$ represent a system of m equations modeling measurements, where each row $\tilde{a}_i$ of matrix A and each corresponding scalar y_i represent one measurement equation.

Our objective function is:

$$\begin{aligned} & \text{minimize} \quad \|Ax - y\|^2 \\ = & \text{minimize} \quad \sum_{i=1}^{m} \left(\tilde{a}_i^T x - y_i\right)^2 \end{aligned}$$

We want to minimize the squared differences between our predicted measurements Ax and our actual measurements y.

The matrix A is constructed such that each row corresponds to a vector $\tilde{a}_i^T$:

$$A = \begin{bmatrix} \tilde{a}_1^T \\ \tilde{a}_2^T \\ \vdots \\ \tilde{a}_m^T \end{bmatrix}$$

We encounter such structures when we estimate parameters based on multiple observations or measurements.

Note that each measurement vector $\tilde{a}_i \in \mathbb{R}^n$, which means each row $\tilde{a}_i^T$ has n components, and $A \in \mathbb{R}^{m \times n}$, where:

- m: number of measurements,
- n: number of parameters being estimated.

Both m and n can increase as we collect more data. Each pair $(\tilde{a}_i^T, y_i)$ corresponds to one measurement.

The least squares solution is given by:

$$x_{\text{ls}} = \left(\sum_{i=1}^{m} \tilde{a}_i \tilde{a}_i^T \right)^{-1} \sum_{i=1}^{m} y_i \tilde{a}_i$$

This solution x_{ls} gets recomputed as we obtain new values of $\tilde{a}_i$ and y_i, i.e., when m increases over time. The sum of the outer products of each measurement with itself is given by:

$$\sum_{i=1}^{m} \tilde{a}_i \tilde{a}_i^T$$

This is a covariance matrix of the measurement directions.

The weighted sum of measurement vectors is:

$$\sum_{i=1}^{m} y_i \tilde{a}_i$$

Each time a new measurement is added, A grows by one row, and y grows by one element.

Hence, recursive estimation lets us update our least squares solution to reflect new data points, without needing to recompute from scratch.

8.1 Recursive Least Squares Algorithm

Initial Setup

Let $P(0)$ denote the initial inverse covariance matrix and $g(0)$ the initial weighted sum of the product of measurements and their corresponding vectors.

Recursive Updates

Given each new measurement pair $(\tilde{a}_m, \tilde{y}_m)$:

Update the P matrix

The update rule for the inverse covariance matrix is:

$$P(m+1) = P(m) + \tilde{a}_m \tilde{a}_m^T$$

This updates the inverse covariance matrix by adding the outer product of the new measurement vector $\tilde{a}_m$.

Update the q vector

The weighted sum vector q is updated as:

$$q(m+1) = q(m) + \tilde{y}_m \tilde{a}_m$$

This updates the weighted sum vector by adding the new measurement scaled by the corresponding measurement vector.

Solution Computation

Computing the Least-Squares Estimate

Assuming $P(m)$ is invertible, the least-squares estimate at step m is given by:

$$x_{\text{ls}}(m) = P(m)^{-1} q(m)$$

Conditions for Unique Solutions

It is essential that the rows $\tilde{a}_1, \ldots, \tilde{a}_m$ span $\mathbb{R}^n$, i.e., they provide enough information to uniquely determine the estimate.

Properties of $P(m)$

Once $P(m)$ becomes invertible, it remains so. This facilitates continuous updates of the solution without $P(m)$ becoming undefined.

Note: It is worth noting that $P(m)$ is at risk of becoming ill-conditioned or singular if the new data matrix does not provide additional information.

For example, suppose new vectors $\tilde{a}_m$ are linearly dependent on previous vectors. In this case, we end up with rank-degeneration, i.e., the new data does not add any new independent information, so the rank of $P(m)$ does not increase. Since the updates neither alter nor expand the capabilities or the content of P, the matrix P stagnates.

8.2 Fast-Update for RLS

We can efficiently update the inverse of the matrix directly when it is modified by a rank-one update of the form $\tilde{a}_{m+1}\tilde{a}_{m+1}^T$.

The rank-one update formula is:

$$(P(m) + \tilde{a}_{m+1}\tilde{a}_{m+1}^T)^{-1} = P(m)^{-1} - \frac{P(m)^{-1}\tilde{a}_{m+1}\tilde{a}_{m+1}^T P(m)^{-1}}{1 + \tilde{a}_{m+1}^T P(m)^{-1}\tilde{a}_{m+1}}$$

This formula holds if:

1. $P = P^T$ (i.e., P is symmetric),
2. P is invertible,
3. $(P + aa^T)^{-1}$ exists.

Efficiency of the Rank-One Update

The RLS update provides an efficient $O(n^2)$ method for computing $P(m+1)^{-1}$ from $P(m)^{-1}$. This is a significant improvement over the standard $O(n^3)$ complexity of directly computing $P(m+1)$.

9 Least-Squares Data Fitting

Consider the setting where we have a set of functions, $\{f_1, f_2, \ldots, f_n\}$, which we will refer to as regressors or basis functions, and a set of data points. We seek to find the coefficients that best fit these data points using a linear combination of the functions f_i.

We are given the functions $f_1, \ldots, f_n$ so that $f_j : S \to \mathbb{R}$. These functions map elements from a set S to real numbers. We then apply each function f_j to each data point $s_i \in S$.

We have m data or measurements $\{(s_i, g_i)\}$ for $i = 1, \ldots, m$, where typically $m \gg n$.

- s_i: Elements from the set S where the measurements are made. This is the domain from which the inputs s_i are taken.
- g_i: Corresponding measured values, or responses, for each s_i.

The measured values g_i can be expressed as a linear combination of the functions $f_1, \ldots, f_n$ evaluated at each point s_i.

Our objective is to find the coefficients $x_1, x_2, \ldots, x_n$ such that the sum of the squared differences between the measured values g_i and the estimated values (from the linear combination of the functions f_i) is minimized. To this end, our model to estimate g_i is given by:

$$g_i \approx x_1 f_1(s_i) + x_2 f_2(s_i) + \cdots + x_n f_n(s_i), \quad \text{for } i = 1, \ldots, m$$

The objective is then to minimize the sum of the squared errors between the measured values g_i and the estimated values $\hat{g}_i$ from the model. Formally, we minimize the following:

$$\sum_{i=1}^{m} \left(x_1 f_1(s_i) + \cdots + x_n f_n(s_i) - g_i\right)^2$$

This expression gives the total least-squares fitting error over the m data points. The values g_i are observed, and the goal is to approximate the model as accurately as possible by finding the best combination of the basis functions.

In this setting, we involve multiple basis functions for modeling, where each function contributes a component to the overall model. The data-fitting problem is then to find the best linear combination of these basis functions that best approximates the target variable g_i based on the given data points.

We construct the matrix A such that each entry A_{ij} is the value of our j-th basis function evaluated at s_i, i.e.,

$$A_{ij} = f_j(s_i)$$

Let g represent the vector of measurements g_i. Our optimization problem is then given by

$$\text{minimize} \quad \|Ax - g\|^2$$

Assuming that A is *skinny* ($m \gg n$) and full rank (i.e., $\mathbf{rank}(A) = n$), the least-squares fit is given by:

$$x = (A^T A)^{-1} A^T g$$

From this, we can construct an estimated function $f_{\text{lsfit}}(s)$:

$$f_{\text{lsfit}}(s) = x_1 f_1(s) + \cdots + x_n f_n(s)$$

This expression is a linear combination of the basis functions, weighted by the coefficients we obtained (contained in x).

Since we are assuming $m \gg n$, we posit that there are more measurements than unknown coefficients. This readily yields a well-posed problem, wherein the matrix $(A^T A)^{-1}$ exists.

9.1 Basis Functions

In the context of data fitting, basis functions are a set of predefined functions used to model a relationship between variables. These functions form the building blocks of the model, with each basis function contributing to the overall fit of the data. When we employ multiple basis functions for the modeling task, each one captures a different aspect or feature of the data. The challenge of the data fitting problem is then to find the optimal linear combination of these basis functions that best approximates the target variable across the set of given data points.

This involves finding the coefficients of the basis functions such that the predicted values (from the linear combination of the basis functions) are as close as possible to the observed values in a least-squares sense. By doing so, we aim to minimize the overall error between the model and the actual data, leading to an effective representation of the underlying relationship between the variables.

9.2 Least-Squares Polynomial Fitting

We now consider the problem of fitting a polynomial of degree n to data points $\{(t_i, y_i)\}$, where $i = 1, \ldots, m$.

Basis Functions

We define our basis functions as:

$$f_j(t) = t^{j-1} \quad \text{for } j = 1, \ldots, n$$

These correspond to the terms of the polynomial, from the constant term t^0 up to t^{n-1}.

Now, we want to represent evaluating our j-th basis function at the i-th data point. This is facilitated using a Vandermonde matrix, which is well-suited for structures where each column is a geometric progression of the prior column.

Polynomial Fitting

We seek to fit the polynomial

$$p(t) = a_0 + a_1 t + \cdots + a_{n-1} t^{n-1}$$

Consequently, matrix A is of the form

$$A_{ij} = t_i^{j-1}$$

which can be written as:

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^{n-1} \end{bmatrix}$$

Least Squares Formulation

Note that our corresponding linear model is then $Ax \approx y$, where:

- $x = [a_0, a_1, \ldots, a_{n-1}]^T$ are the polynomial coefficients we seek to model.
- $y = [y_1, y_2, \ldots, y_m]^T$ are our observed data points at times $t_1, t_2, \ldots, t_m$.

We seek to minimize the squared error between the observed data and the model predictions:

$$\text{minimize} \quad \|Ax - y\|^2$$

Properties of the Vandermonde Matrix

The Vandermonde matrix is well-suited to polynomial fitting problems and is full rank under certain conditions.

Simplifications

Consider the case where $Ax = 0$. Recall, x is the vector of polynomial coefficients:

$$x = [a_0, a_1, \ldots, a_{n-1}]$$

In this case, the polynomial $p(t) = a_0 + a_1 t + \cdots + a_{n-1}t^{n-1}$ is zero when evaluated at m distinct points $t_1, \ldots, t_m$. Therefore, $x \in \mathcal{N}(A) = 0$ as well.

The fundamental theorem of algebra tells us that if the polynomial $p(t)$ is of degree $n-1$, it can have at most $n-1$ zeros, unless it is the zero polynomial (i.e., $a = [a_0, a_1, \ldots, a_{n-1}] = 0$).

Interestingly enough, $p(t)$ vanishes at m points, where $m \geq n$. Moreover, we assume $t_i \neq t_j$ for all $i \neq j$.

The only possibility for this to occur would be if $p(t) = 0$, and:

$$p(t) = 0 \iff a_0 = a_1 = \cdots = a_{n-1} = 0$$

Thus, since the only solution to $Ax = 0$ is the trivial solution $x = 0$, A's columns must be linearly independent. Hence, we conclude that A has full rank, and that $\mathbf{rank}(A) = n$. The Vandermonde matrix is constrained by the number of coefficients in the polynomial we are trying to fit.

10 Continuous Time Linear Dynamical Systems (LDS)

Consider the continuous-time linear dynamical system (LDS):

$$\dot{x} = Ax + Bu$$

where $\dot{x}$ is the derivative of the state with respect to time, $x(t)$ is the state, $u(t)$ is the input, A is the system matrix, and B is the input matrix.

The output is given by:

$$y = Cx + Du$$

where y is the output, C is the output matrix, and D is the feedthrough (or direct) term.

The solution to the state-space equation is given by:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)\,d\tau$$

where the first term, $e^{At}x(0)$, represents the autonomous response, and the second term accounts for the forced response due to the input $u(t)$.

For the specific case of zero input, the solution simplifies to:

$$x(t) = e^{At}x(0)$$

where e^{At} is known as the state transition matrix or impulse matrix.

Impulse Response

The impulse response of the system is a crucial concept in understanding the system's behavior when an impulse is applied at time $t = 0$. The system's output in response to an impulse input is given by:

$$h(t) = Ce^{At}B + D\delta(t)$$

This gives us insight into how the system's output y evolves over time in response to an instantaneous impulse input applied at $t = 0$. The response is composed of the system dynamics via A and the direct input effect via B.

Discretization with Piecewise Constant Inputs

We start with the LDS in continuous time:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

To discretize this system, we apply a zero-order hold (ZOH) to keep the input $u(t)$ constant between sampling instants. Let $u_k = u(kh)$ be the input at the k-th sampling instant, where h is the sampling interval. We assume that:

$$u(t) = u_k \quad \text{for } t \in [kh, (k+1)h], \quad k = 0, 1, \ldots$$

The sampling interval h defines the time between consecutive samples in a sampled data system. The value of h governs how frequently the state and output are sampled in order to create a discrete-time version from the continuous-time model.

The variables $x_d(k)$ and $y_d(k)$ are sampled values of the states and outputs at the sampling instants kh, starting from $k = 0$.

To this end, in discretizing $x(t+1) = Ax(t)+Bu(t)$ using a ZOH (Zero-Order Hold) on the input, we get the discrete-time system:

$$\begin{aligned} x_d(k+1) &= A_d x_d(k) + B_d u_d(k) \\ y_d(k) &= C_d x_d(k) + D_d u_d(k) \end{aligned}$$

So, to summarize, we apply the ZOH to a continuous system, then measure how the state and input behave as a result:

$$x_d(k+1) = x((k+1)h) = e^{Ah} x_d(k) + \left(\int_0^h e^{A\tau} d\tau \right) B u_d(k)$$

Thus, x_d, u_d, and y_d satisfy discrete-time LTI system equations:

$$\begin{aligned} x_d(k+1) &= A_d x_d(k) + B_d u_d(k) \\ y_d(k) &= C_d x_d(k) + D_d u_d(k) \end{aligned}$$

Where:

$$\begin{aligned} A_d &= e^{Ah} \\ B_d &= \left(\int_0^h e^{A\tau} d\tau \right) B \\ C_d &= C \\ D_d &= D \end{aligned}$$

If A is invertible, then:

$$B_d = A^{-1} \left(e^{Ah} - I \right) B$$

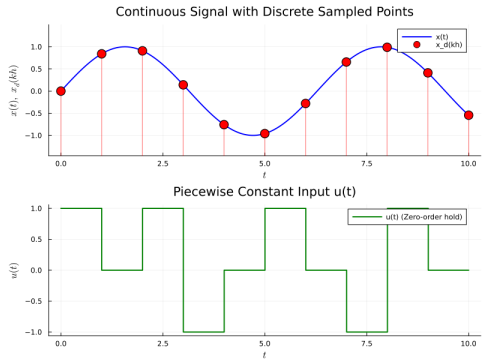


Figure 9: *Top.* Continuous-time signal $x(t) = \sin(t)$ (blue curve) with discrete samples $x_d(kh)$ (red dots) taken at intervals of $h = 1$. Vertical dashed lines beneath the sampled points indicate the sampling instants. *Bottom.* Zero-order hold input signal $u(t)$ (green step function), formed by holding each discrete sample $u_d(kh)$ constant over the interval $[kh, (k+1)h)$.

10.1 Discrete Time Systems

Consider the discrete-time state-space system:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

We can express the state at the first and second time steps as follows:

$$\begin{aligned} x(1) &= Ax(0) + Bu(0) \\ x(2) &= A^2x(0) + ABu(0) + Bu(1) \end{aligned}$$

Generalizing this for $t \in \mathbb{Z}_+$, the state at time t is given by:

$$x(t) = A^t x(0) + \sum_{i=0}^{t-1} A^{t-1-\tau} Bu(\tau)$$

Each term $A^{t-1-\tau}Bu(\tau)$ represents the effect of the input $u(\tau)$ at time τ on the state at time t, propagated through the system dynamics by the matrix A raised to the power $t-1-\tau$.

Interpretation

The matrix A governs how the effect of the input $Bu(\tau)$ evolves over $t-1-\tau$ time steps.

Each component of the sum

$$\sum_{\tau=0}^{t-1} A^{t-1-\tau} B u(\tau)$$

accounts for the influence of the control input at each previous time step, adjusted for how long ago the input was applied relative to the current time t. The term $A^{t-1-\tau}$ propagates the effect of the input $u(\tau)$ forward to the state at time t, while $Bu(\tau)$ translates the control input $u(\tau)$ into a change in the state space.

The solution to the discrete-time system is:

$$x(t) = A^t x(0) + \sum_{\tau=0}^{t-1} A^{t-1-i} B u(i)$$

And the output is:

$$y(t) = CA^t x(0) + \sum_{\tau=0}^{t-1} H(t-\tau) u(\tau)$$

Where $H(t)$ is the system's impulse response, defined as:

$$H(t) = \begin{cases} 0 & t < 0 \\ CA^{t-1}B & t \geq 0 \end{cases}$$

The system can be expressed in matrix form as:

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(t) \end{bmatrix} = \begin{bmatrix} D & & & & \\ CB & D & & & \\ CAB & CB & D & & \\ \vdots & & \ddots & \ddots & \\ CA^{t-1}B & CA^{t-2}B & \dots & CB & D \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(t) \end{bmatrix} + \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^t \end{bmatrix} x(0)$$

The resulting matrix has a block-Toeplitz structure, where each descending diagonal consists of repeated blocks. This is a key characteristic of time-invariant (TI) systems.

The matrices above describe systems where the output at a given time step is influenced by the input at preceding time steps, which is a characteristic of causal systems.

The block-Toeplitz structure captures the time-invariance (TI) property of the system, meaning the relationship between inputs and outputs remains the same over time. T