Mathematical Models and Methods For Image Processing

Rao
Politecnico di Milano

Originally written in: **2025-02-17**

Last updated at: **2025-07-20**

Contents

1 Introduction to Sparsity 4
1.1 Sparsity in Statistics 4
1.2 Sparsity in Signal Processing 4
2 Image Prior 5
2.1 Image Denoising Problem 5
2.2 Local Constancy Prior 6
2.3 Non-local Means Prior 7
2.4 Block-Matching 3D (BM3D) Algorithm 9
2.5 Sparsity-Based Image Prior 11
3 Discrete Cosine Transform (DCT) 12
3.1 1D DCT 12
3.2 2D DCT 12
3.3 DCT-Based Denoising Pipeline 13
3.4 Noise Standard Deviation Estimation 13
3.5 Sliding DCT Algorithm 14
3.6 Wiener Filter 16
3.7 The Sparsity Problem 17
3.8 Experimental Demonstration 17
4 Introduction to Dictionary Learning 18
4.1 Theoretical Properties of Overcomplete Systems 18
4.2 Regularization and Sparse Recovery 19
4.3 Towards Sparsity: ℓ_0 and ℓ_1 Regularization 20
5 Sparse Coding In ℓ_0 sense 21
5.1 The Sparse Coding Problem 21
5.2 Matching Pursuit Algorithm 23
5.3 Orthogonal Matching Pursuit 25
5.4 OMP-Based Image Denoising 26
5.5 Linearity Analysis of Sparse Coding Algorithms 27
5.6 Uniqueness Guarantees for Sparse Solutions 28
5.7 Application to Image Inpainting 29
6 Sparse Coding In ℓ_1 sense 30
6.1 The ℓ_1 Optimization Problem 30
6.2 Problem Components Analysis 30
6.3 Proximal Gradient Methods 31
6.4 FISTA 34
7 Structured Sparsity 36
7.1 Joint Sparsity and Mixed Norms 37
7.2 Joint Sparse Coding 38
7.3 Group Sparsity and Extensions 39
7.4 LASSO 40
7.5 Elastic Net 41
7.6 Algorithmic Summary 41
8 Dictionary Learning 42

8.1 Introduction to Dictionary Learning 42
8.2 Problem Formulation 42
8.3 Sparse Coding Phase 43
8.4 Dictionary Update Phase 43
9 Local Polynomial Approximation 47
9.1 Local Polynomial Model 47
9.2 Weighted Local Polynomial Approximation 49
9.3 QR Decomposition Approach 50
9.4 Convolution Implementation 51
9.5 Special Cases 51
9.6 Statistical Properties and Performance Analysis 52
9.7 Adaptive Neighborhood Selection 53
9.8 The Intersection of Confidence Intervals (ICI) Rule 53
10 Robust Fitting 55
10.1 Limitations of Vertical Distance Minimization 55
10.2 The Direct Linear Transformation (DLT) Algorithm 56
10.3 Robust Estimation Methods 59
10.4 RANSAC: Random Sample Consensus 60
10.5 MSAC and MLESAC Variants 61
11 Multiple Model Fitting 62
11.1 Outlier Complexity in Multi-Model Settings 62
11.2 Sequential Multi-Model Fitting 63
11.3 Simultaneous Multi-Model Fitting 65
12 Appendix: Fundamental Concepts in Linear Algebra 68
12.1 Vector Spaces and Linear Combinations 68
12.2 Linear Independence and Basis 68
12.3 Orthogonal Vectors 69
12.4 The ℓ_0 Norm 69
12.5 Matrix Spark 70
13 Appendix: Optimization Theory for Non-Differentiable Functions 71
13.1 Lipschitz Continuity 71
13.2 Majorization-Minimization 72

1 Introduction to Sparsity

The principle of sparsity or "parsimony" consists in representing some phenomenon with as few variable as possible. Stretch back to philosopher William Ockham in the 14th century, Wrinch and Jeffreys relate simplicity to parsimony:

The **existence of simple laws** is, then, apparently, to be regarded as **a quality of nature**; and accordingly we may infer that it is justifiable to **prefer a simple law to a more complex one that fits our observations slightly better**.

1.1 Sparsity in Statistics

Sparsity is used to **prevent overfitting and improve interpretability of learned models**. In model fitting, the number of parameters is typically used as a criterion to perform model selection. See Bayes Information Criterion (BIC), Akaike Information Criterion (AIC), …., Lasso.

1.2 Sparsity in Signal Processing

Signal Processing: similar concepts but different terminology. **Vectors corresponds to signals** and data modeling is crucial for performing various operations such as **restoration, compression, solving inverse problems**.

Signals are approximated by sparse linear combinations of **prototypes**(basis elements / atoms of a dictionary), resulting in simpler and compact model.

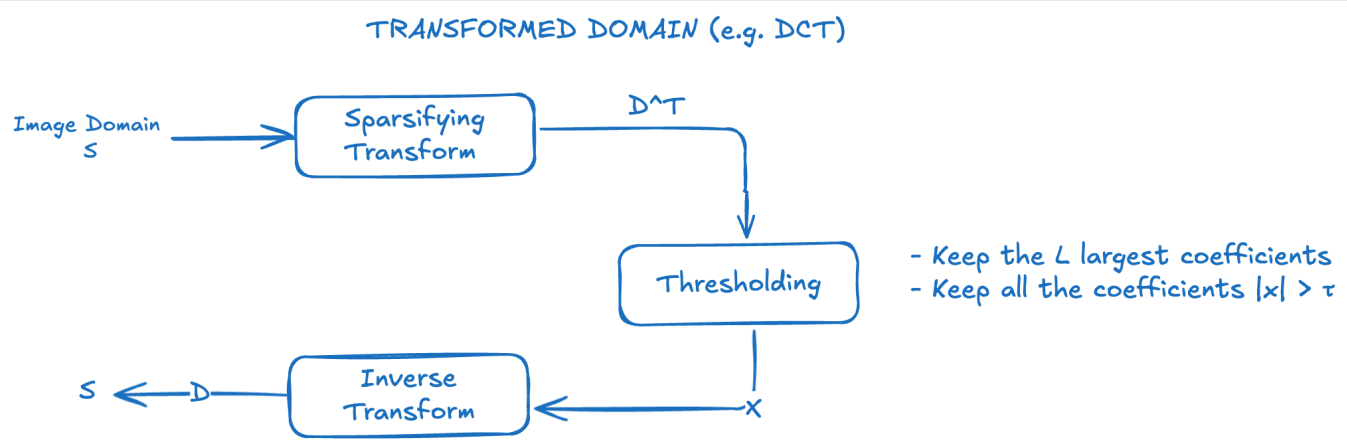

Figure 1.1: Enforce sparsity in signal processing

2 Image Prior

2.1 Image Denoising Problem

Image denoising provides a simple and clear problem formulation example. The ***observation model*** is:

$$z(x) = y(x) + \eta(x) \tag{2.1}$$

where:

- $z(x)$ is noisy observation at pixel coordinate x
- $y(x)$ is ideal (noise-free) image
- $\eta(x)$is the noise component
- x is pixel coordinate

This formulation assumes *additive white Gaussian noise*(AWGN).

We assume that the noise is:

- **Additive Gaussian noise**: $\eta(x) \sim N(0, \sigma^2)$.
- **Independent and identically distributed (i.i.d.)**: Noise realizations at different pixels are independent.

While real-world noise may exhibit correlations or non-Gaussian characteristics, the AWGN model remains prevalent for algorithm design. Practical systems often transform raw sensor data to approximate this model. The denoising objective is formalized as finding an estimator $\hat{y}$ minimizing the the mean squared error (MSE):

$$\text{MSE}(\hat{\boldsymbol{y}}) = \mathbb{E}[\|\hat{y} - y\|_2^2] \tag{2.2}$$

The observation model provides a **prior on noise** but we also need a **prior on images**.

Definition 2.1.1 (Noise Prior): A **noise prior** refers to the assumed statistical properties or characteristics of the noise itself that is corrupting the image. Knowing how the noise behaves helps in modeling and subsequently removing it. For AWGN, the true image y is likely to be a in a circular neighborhood around the observation $z(x)$.

Definition 2.1.2 (Image Prior): An **image prior** is a statistical model that captures the expected structure of natural images. It is used to regularize the denoising process, guiding the estimator towards plausible solutions.

In this chapter, we will explore various image priors and their applications in denoising. The goal is to leverage the statistical properties of natural images to improve the quality of denoised outputs.

2.2 Local Constancy Prior

Assumption: Images are locally constant within small patches. For a constant signal corrupted by Gaussian noise:

$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} z(x_i) \tag{2.3}$$

Properties of the this estimator:

- **Unbiased:** $\mathbb{E}[\hat{y}] = y$ (true signal)
- **Reduced Variance:** $\text{Var}[\hat{y}] = \frac{\sigma^2}{M}$

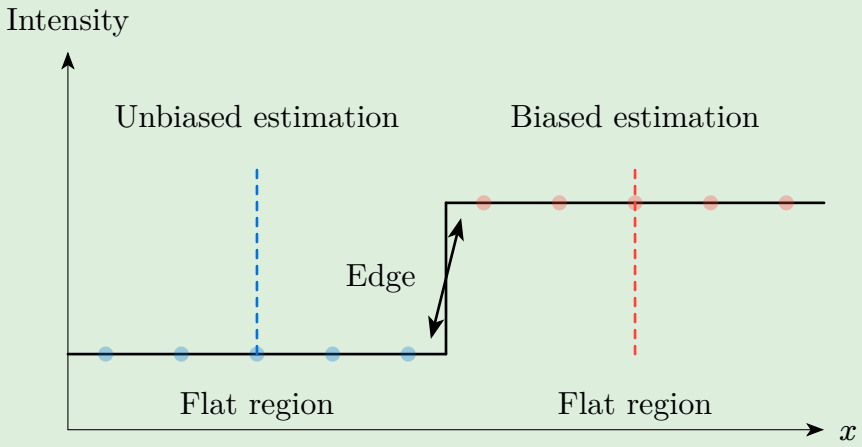

Figure 2.1: Bias-variance tradeoff in local averaging

Consider the classical approach of estimating pixel intensities through local averaging. For a pixel at location (i, j), a naive estimate would be:

$$\hat{y}_{i,j} = \frac{1}{|U_{i,j}|} \sum_{(k,l) \in U_{i,j}} z_{k,l} \tag{2.4}$$

where $U_{i,j}$ denotes a neighborhood around pixel (i, j), and $|U_{i,j}|$ is its cardinality.

This approach is equivalent to convolution with a normalized box kernel:

$$\hat{y} = z * h \tag{2.5}$$

where h is the box kernel with support $U_{i,j}$.

The primary limitation of local averaging becomes apparent near image discontinuities. Consider an ideal edge scenario where:

$$z_{i,j} = \begin{cases} a \text{ if } & (i,j) \in \Omega_1 \\ b \text{ if } & (i,j) \in \Omega_2 \end{cases} \tag{2.6}$$

where Ω_1 and Ω_2 are disjoint regions separated by an edge, and $a \neq b$.

For a pixel (i, j) near the edge boundary, the local average becomes:

$$\begin{aligned} \hat{y}_{i,j} &= \frac{1}{|U_{i,j}|} \left(\sum_{(k,l) \in U_{i,j} \cap \Omega_1} y_{k,l} + \sum_{(k,l) \in U_{i,j} \cap \Omega_2} y_{k,l} \right) \\ &\approx \frac{|\Omega_1 \cap U_{i,j}|}{|U_{i,j}|} a + \frac{|\Omega_2 \cap U_{i,j}|}{|U_{i,j}|} b \end{aligned} \tag{2.7}$$

This results in a blurred edge transition, motivating the need for adaptive filtering strategies.

2.3 Non-local Means Prior

The Non-Local Means (NLM) algorithm addresses the limitations of local averaging by exploiting the *self-similarity* property of natural images. **The key insight is that similar patches exist throughout the image, not just in local neighborhoods.**

Definition 2.3.1 (Self-Similarity Prior): For a natural image $\boldsymbol{Z}$, there exist multiple patches $\{P_k\}$ such that:

$$\|P_i - P_j\|_2 < \varepsilon \tag{2.8}$$

for some small threshold $\varepsilon > 0$, where P_k represents a patch extracted from the image.

The NLM estimate for pixel (i, j) is given by:

$$\hat{y}_{i,j} = \sum_{(k,l)\in S_{i,j}} w_{i,j}(k,l) \cdot z_{k,l} \tag{2.9}$$

where $S_{i,j}$ represents the search window around pixel (i, j), and $w_{i,j}(k, l)$ are the similarity weights.

2.3.1 Weight Computation

The similarity weights are computed based on patch distances:

$$w_{i,j}(k,l) = \frac{1}{Z_{i,j}} \exp\left(-\frac{d^2\big(P_{i,j}, P_{k,l}\big)}{h^2}\right) \tag{2.10}$$

where:

- $P_{i,j}$ and $P_{k,l}$ are patches centered at (i, j) and (k, l) respectively
- $d^2\big(P_{i,j}, P_{k,l}\big)$ is the squared Euclidean distance between patches
- h is the filtering parameter controlling the decay rate
- $Z_{i,j}$ is the normalization constant ensuring $\sum w_{i,j}(k, l) = 1$

2.3.2 Patch Distance Computation

The patch distance is computed as:

$$d^2\big(P_{i,j}, P_{k,l}\big) = \frac{1}{|P|} \sum_{(u,v)\in P} |z_{i+u,j+v} - z_{k+u,l+v}|^2 \tag{2.11}$$

where P represents the patch domain and $|P|$ is the number of pixels in the patch.

Noise-Aware Distance In the presence of noise, the distance can be corrected as:

$$d^2_{\text{corrected}(P_{i,j},P_{k,l})} = \max\big(d^2\big(P_{i,j}, P_{k,l}\big) - 2\sigma^2, 0\big) \tag{2.12}$$

This correction accounts for the noise contribution to the patch distance.

2.3.3 Normalization and Properties

The normalization constant is given by:

$$Z_{i,j} = \sum_{(k,l)\in S_{i,j}} \exp\left(-\frac{d^2\big(P_{i,j}, P_{k,l}\big)}{h^2}\right) \tag{2.13}$$

Theorem 2.3.3.1 (NLM Consistency): For a noiseless image, the NLM estimate satisfies:

$$\lim_{\sigma\to 0} \hat{y}_{i,j} = y_{i,j} \tag{2.14}$$

Proof: As $\sigma \to 0$, the patch distances approach their true values, and the weight distribution becomes increasingly concentrated around patches identical to the reference patch.

2.3.4 Algorithm Framework

Algorithm 1: Non-Local Means Algorithm

procedure NLM(Noisy image y, patch size p, search window size s, filtering parameter h)
  ▷ **Input:** Noisy image y, patch size p, search window size s, filtering parameter h
  ▷ **Output:** Denoised image $\hat{x}$

  ▷ For each pixel (i, j) in the image:
  for each pixel (i, j) in the image **do**
    ▷ Initialize weight matrix $W = 0$
    $W \leftarrow 0$

    ▷ Define search window $S_{i,j}$ of size $s \times s$
    $S_{i,j} \leftarrow$ search window of size $s \times s$ around (i, j)

    ▷ For each pixel (k, l) in $S_{i,j}$:
    for each pixel (k, l) in $S_{i,j}$ **do**
      ▷ Extract patches $P_{i,j}$ and $P_{k,l}$ of size $p \times p$
      $P_{i,j}, P_{k,l} \leftarrow$ patches of size $p \times p$

      ▷ Compute distance $d^2(P_{i,j}, P_{k,l})$ using patch distance formula

      $d^2(P_{i,j}, P_{k,l}) \leftarrow \frac{1}{|P|} \sum_{(u,v)\in P} \left|y_{i+u,j+v} - y_{k+u,l+v}\right|^2$

      ▷ Compute weight $w_{i,j}(k, l)$ using NLM weights formula

      $w_{i,j}(k, l) \leftarrow \exp\left(-\frac{d^2(P_{i,j},P_{k,l})}{h^2}\right)$

      $W(k, l) \leftarrow w_{i,j}(k, l)$
    end

    ▷ Normalize weights: $W = \frac{W}{\sum} W$
    $W \leftarrow \frac{W}{\sum} W$

    ▷ Compute estimate: $\hat{x}_{i,j} = \sum_{(k,l)} W(k, l) \cdot y_{k,l}$
    $\hat{x}_{i,j} \leftarrow \sum_{(k,l)} W(k, l) \cdot y_{k,l}$
  end

  return $\hat{x}$
end

Near image boundaries, patches may extend beyond the image domain. Common strategies include:

1. **Symmetric Padding**: Reflect image content across boundaries
2. **Periodic Padding**: Wrap image content periodically
3. **Zero Padding**: Extend with zeros (not recommended)

The symmetric padding approach is preferred as it maintains image statistics:

$$\tilde{x}_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \text{ is inside image} \\ x_{2N-i,j} & \text{if } i > N \text{ (bottom boundary)} \\ x_{i,2M-j} & \text{if } j > M \text{ (right boundary)} \end{cases} \tag{2.15}$$

2.3.5 Parameter Selection

Typical Parameter Values

- Patch size: $p = 7 \times 7$ (provides good balance between detail and computational cost)
- Search window: $s = 21 \times 21$ (sufficient for finding similar patches)
- Filtering parameter: $h = 10\sigma$ (empirically determined)

Adaptive Parameter Selection

The filtering parameter can be adapted based on local image characteristics:

$$h_{i,j} = \alpha \cdot \sigma \cdot \sqrt{\text{LocalVariance}(P_{i,j})} \tag{2.16}$$

where α is a scaling factor and LocalVariance measures local image activity.

2.4 Block-Matching 3D (BM3D) Algorithm

The Block-Matching 3D (BM3D) algorithm extends the self-similarity concept by combining it with sparsity-based denoising. The key innovations include:

1. **Grouping**: Collect similar patches into 3D arrays
2. **Collaborative Filtering**: Process groups jointly using 3D transforms
3. **Aggregation**: Combine processed patches back into the image

2.4.1 Patch Grouping

For a reference patch $P_{i,j}$, we define the group $G_{i,j}$ as:

$$G_{i,j} = \{P_{k,l} : d^2(P_{i,j}, P_{k,l}) < \tau\} \tag{2.17}$$

where τ is a similarity threshold.

2.4.2 3D Array Construction

The group is arranged as a 3D array $\mathcal{G} \in \mathbb{R}^{p \times p \times K}$, where $K = |G_{i,j}|$ is the number of patches in the group.

The BM3D algorithm applies a 3D transform to exploit both spatial and inter-patch correlations:

$$\mathcal{T} = \mathcal{W}_{3D} \cdot \mathcal{G} \tag{2.18}$$

where $\mathcal{W}_{3D}$ represents the 3D transform operator.

2.4.3 Separable Transform

The 3D transform is typically implemented as a separable transformation:

$$\begin{aligned} \mathcal{T} &= \mathcal{W}_1 \cdot (\mathcal{W}_2 \cdot (\mathcal{W}_3 \cdot \mathcal{G})) \\ &= (\mathcal{W}_1 \times \mathcal{W}_2 \times \mathcal{W}_3) \cdot \mathcal{G} \end{aligned} \tag{2.19}$$

where $\mathcal{W}_1$ and $\mathcal{W}_2$ are 2D transforms (e.g., DCT) applied to each patch, and $\mathcal{W}_3$ is a 1D transform applied across the grouping dimension.

2.4.4 Hard Thresholding Stage

In the first stage, BM3D applies hard thresholding to the transform coefficients:

$$\hat{\mathcal{T}} = \mathcal{H}_\lambda(\mathcal{T}) \tag{2.20}$$

where the hard thresholding operator is defined as:

$$\mathcal{H}_\lambda(t) = \begin{cases} t & \text{if } |t| > \lambda \\ 0 & \text{if } |t| \leq \lambda \end{cases} \tag{2.21}$$

2.4.5 Threshold Selection

The threshold is typically chosen as:

$$\lambda = \beta \cdot \sigma \tag{2.22}$$

where β is a parameter controlling the aggressiveness of denoising (typically $\beta = 2.7$).

2.4.6 Collaborative Filtering Stage

The second stage performs collaborative filtering using both the noisy and first-stage estimates:

$$\hat{\mathcal{T}}^{(2)} = \frac{|\mathcal{T}^{(1)}|^2}{|\mathcal{T}^{(1)}|^2 + \sigma^2} \cdot \mathcal{T}^{(0)} \tag{2.23}$$

where:

- $\mathcal{T}^{(0)}$ represents the transform of the noisy group
- $\mathcal{T}^{(1)}$ represents the transform of the first-stage estimate
- $|\cdot|^2$ denotes element-wise squared magnitude

2.4.7 Aggregation and Weighting

After processing, patches must be aggregated back into the image. Due to overlapping patches, multiple estimates exist for each pixel:

$$\hat{x}_{i,j} = \frac{\sum_{G \ni (i,j)} w_G \cdot \hat{x}_{i,j}^{(G)}}{\sum_{G \ni (i,j)} w_G} \tag{2.24}$$

where the sum is over all groups G containing pixel (i, j).

Sparsity-Aware Weighting

The weights are chosen based on the sparsity of the processed group:

$$w_G = \frac{1}{\|\mathcal{T}_G\|_0} \tag{2.25}$$

where $\|\cdot\|_0$ denotes the number of non-zero coefficients.

2.5 Sparsity-Based Image Prior

Natural images have **sparse representations** in certain **transform domains** (e.g., DCT), as evidenced by the success of JPEG compression. There are two types of sparsity models:

- **Transform-domain sparsity**: Images can be represented as sparse linear combinations of basis functions in a specific transform domain.
- **Synthesis Sparse Model**: The synthetic sparse model assumes that a signal (such as an image) can be synthesized through a linear combination of a small number of "atoms." These "atoms" are drawn from a collection known as a **dictionary**.

2.5.1 Transform-domain Sparsity

Transform-domain sparsity is a fundamental concept in signal and image processing, asserting that many real-world signals and images, though complex in their original form, can be represented much more efficiently in a different mathematical domain. This efficiency is achieved when, after a specific mathematical transformation, the majority of the resulting coefficients are zero or close to zero, leaving only a few significant, non-zero values that capture the essential information of the original data. Some of the most common and powerful transforms include:

- **Discrete Cosine Transform (DCT)**: Widely used in image compression (e.g., JPEG), it transforms spatial domain data into frequency domain, emphasizing low-frequency components.
- **Wavelet Transform**: Decomposes signals into different frequency components, allowing for multi-resolution analysis.
- **Fourier Transform (DFT)**: Converts signals from time domain to frequency domain, revealing periodic structures.

2.5.2 Synthesis Sparse Model

The **synthesis sparse model** is a mathematical framework used to represent signals or images as linear combinations of a small number of basis functions, known as "atoms." This model is particularly useful in applications like image compression, denoising, and reconstruction, where the goal is to efficiently represent data while preserving essential features. There are two main approaches to synthesis sparse modeling:

- **Sparse Coding**: In this approach, the signal is expressed as a linear combination of a dictionary of atoms. The goal is to find a sparse representation where only a few coefficients are non-zero, indicating that only a small number of atoms are used to reconstruct the signal. This is often achieved through optimization techniques that minimize the reconstruction error while enforcing sparsity constraints.
- **Dictionary Learning**: This approach involves learning a dictionary of atoms from a set of training signals. The dictionary is optimized to capture the underlying structure of the data, allowing for efficient representation. The learned dictionary can then be used to represent new signals or images in a sparse manner. Dictionary learning algorithms aim to find a set of basis functions that best represent the training data, often through iterative optimization techniques.

3 Discrete Cosine Transform (DCT)

3.1 1D DCT

Generate the DCT basis according to the following formula, the k-th atom of the DCT basis in dimension M is defined as:

$$\mathrm{DCT}_{k(n)} = c_k \cos\left(k\pi \frac{2n+1}{2M}\right) \quad n, k = 0, 1, ..., M-1 \tag{3.1}$$

where $c_0 = \sqrt{\frac{1}{M}}$ and $c_k = \sqrt{\frac{2}{M}}$ for $k \neq 0$.

For each $k = 0, ..., M-1$, just sample each function

$$\mathrm{DCT}_{k(n)} = \cos\left(k\pi \frac{2n+1}{2M}\right) \tag{3.2}$$

at $n = 0, ..., M-1$, obtain a vector. Ignore the normalization coefficient. Divide each vector by its ℓ_2 norm.

Mathematically, suppose the image signal is $s \in \mathbb{R}^M$.

$$x = \mathrm{dct2}(s) = D^T s \tag{3.3}$$

where D^T represents the **DCT basis matrix**. x contains the **DCT coefficients**, which are a **sparse representation** of s.

The **inverse DCT transformation** reconstructs s from x:

$$s = \mathrm{idct2}(x) = Dx \tag{3.4}$$

3.2 2D DCT

2D Discrete Cosine Transform (DCT) can be used as a dictionary for representing image patches. A small patch of an image is extracted, represented as s, with dimension $p \times p$. This patch can be **flattened** into a vector of length $M = p^2$, meaning each patch is reshaped into a vector of length M. The **2D-DCT** is used to transform the patch s into DCT coefficients x.

Suppose the image signal is $S \in \mathbb{R}^{M \times N}$.The 2D DCT can be decomposed into two **1D DCT** operations:

1. **Column-wise DCT**: apply **1D DCT** to each column of the image patch: $\mathrm{Z} = D^T S$
2. **Row-wise DCT**: apply **1D DCT** to each column of the image patch: $X^T = D^T \mathrm{Z}^T \rightarrow X = D^T S D$

Example 3.2.1 (JPEG Compression): The image is divided into non-overlapping 8×8 blocks. Each block is treated separately during the compression process.

For each 8×8 block, the **DCT** is applied, transforming pixel values into frequency-domain coefficients. Each 8×8 block's coefficients are checked against a compression threshold τ, coefficients with absolute values below τ are **discarded**(set to zero). The larger the threshold τ, the more coefficients are discarded, leading to **higher compression**.

The compression ratio is defined as:

$$\text{Comp Ratio} = 1 - \frac{\#\text{Non-zero coefficients}}{\#\text{Pixels in the image}} \tag{3.5}$$

To measure how much the image quality is degraded after compression, **Peak Signal-to-Noise Ratio (PSNR)** is used:

$$\text{PSNR} = 10 \log_{10} \left(\frac{1}{\text{MSE}\left(Y, \hat{Y}\right)} \right) \tag{3.6}$$

3.3 DCT-Based Denoising Pipeline

Step 1 : Analysis

$$X = D^T S \tag{3.7}$$

where:

- S is the vectorized image patch
- D is the DCT basis matrix
- X is the DCT coefficients vector

Step 2: Enforce Sparsity (Thresholding)

$$\hat{X}_i = \begin{cases} X_i & \text{if } |X_i| \geq \gamma \\ 0 & \text{if } |X_i| < \gamma \end{cases} \tag{3.8}$$

Important: Apply thresholding only to the coefficients $i \geq 1$ (preserve the DC component).

Step 3: Synthesis

$$\hat{S} = D\hat{X} \tag{3.9}$$

Theorem 3.3.1 (Universal Thresholding): For AWGN with variance σ^2, the optimal threshold for denoising is given by:

$$\gamma = \sigma \sqrt{2 \log(n)} \tag{3.10}$$

where:

- σ is the noise standard deviation
- n is the dimension of coefficients vector

For 8×8 patches: $\gamma \approx 3\sigma$

3.4 Noise Standard Deviation Estimation

To use the universal thresholding, we need to estimate σ from the noisy image itself.

Theorem 3.4.1 (Robust Estimation of Noise Standard Deviation): Given a noisy image Z, the noise standard deviation can be estimated using the following robust method:

$$\hat{\sigma} = \frac{\text{MAD}}{0.6745 \times \sqrt{2}} \tag{3.11}$$

where MAD = Median Absolute Deviation, defined as:

$$\text{MAD}(D) = \text{median}(|D - \text{median}(D)|) \tag{3.12}$$

where D denotes the horizontal differences of the image.

3.5 Sliding DCT Algorithm

3.5.1 Non-Overlapping Tiles (No Aggregation)

The simplest approach processes the image in non-overlapping 8×8 blocks. Let Z be the noisy image of size $M \times N$, partitioned into non-overlapping blocks $B_{i,j}$ of size 8×8. For each block:

$$X_{i,j} = \text{DCT}_2\big(B_{i,j}\big) \tag{3.13}$$

Apply hard thresholding with universal threshold $\gamma = 3\sigma$:

$$\hat{X}_{i,j}(u,v) = \begin{cases} X_{i,j}(u,v) & \text{if } \big|X_{i,j}(u,v)\big| \geq \gamma \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

Reconstruct each block:

$$\hat{S}_{i,j} = \text{IDCT}_2\big(\hat{X}_{i,j}\big) \tag{3.15}$$

The final denoised image is the union of all processed blocks:

$$\hat{Y} = \bigcup_{i,j} \hat{B}_{i,j} \tag{3.16}$$

Properties:

- Complexity: $O(N)$ operations
- Blocking artifacts due to independent processing
- Fast but lower quality

3.5.2 Sliding Window with Uniform Weights

For overlapping patches, each pixel receives multiple estimates that must be aggregated.

For each patch position (i, j) with step size $\text{STEP} = 1$, extract $p \times p$ patch $S_{i,j}$ from the noisy image and do the same DCT processing as before, getting the reconstructed patch $\hat{S}_{i,j}$.

The denoised image $\hat{I}$ is obtained by weighted aggregation:

$$\hat{I}(m,n) = \frac{\sum_{i,j \in \Omega_{m,n}} w \cdot \hat{S}_{i,j}(m-i, n-j)}{\sum_{i,j \in \Omega_{m,n}} w + \varepsilon} \tag{3.17}$$

where $w = 1.0$ is the uniform weight, $\varepsilon = 10^{-8}$ prevents division by zero, and $\Omega_{m,n}$ denotes the set of patch positions (i, j) that contain pixel (m, n).

For an image of size $M \times N$, each pixel (m, n) can be covered by at most $p \times p$ overlapping patches when using unit step size, but the actual number depends on the pixel's position relative to image boundaries.

3.5.3 Sparsity-Adaptive Weight Aggregation

For overlapping patches, each pixel receives multiple estimates that must be aggregated with weights adapted to the sparsity of the thresholded DCT coefficients.

For each patch position (i, j) with step size $\text{STEP} = 1$, extract $p \times p$ patch $S_{i,j}$ from the noisy image, getting the reconstructed patch $\hat{S}_{i,j}$.

The sparsity-adaptive weight for each patch is computed based on the number of **non-zero coefficients** after thresholding:

$$w_{i,j} = \text{nnz}\big(\hat{X}_{i,j}\big) \tag{3.18}$$

where $\text{nnz}\big(\hat{X}_{i,j}\big)$ counts the number of non-zero elements in the thresholded DCT coefficient matrix.

The denoised image $\hat{I}$ is obtained by sparsity-weighted aggregation:

$$\hat{I}(m, n) = \frac{\sum_{i,j \in \Omega_{m,n}} w_{i,j} \cdot \hat{S}_{i,j}(m - i, n - j)}{\sum_{i,j \in \Omega_{m,n}} w_{i,j} + \varepsilon} \tag{3.19}$$

where $\varepsilon = 10^{-8}$ prevents division by zero, and $\Omega_{m,n}$ denotes the set of patch positions (i, j) that contain pixel (m, n).

3.6 Wiener Filter

The **Wiener filter** is a powerful tool for image denoising, particularly when the noise characteristics are known. It operates in the frequency domain, leveraging the DCT coefficients to perform adaptive filtering based on local statistics.

3.6.1 Empirical Wiener Filter

Let $\hat{\boldsymbol{y}}^{\mathrm{HT}}$ be the **hard threshold estimate**, with DCT coefficients:

$$\hat{\boldsymbol{x}}^{\mathrm{HT}} = D^T \hat{\boldsymbol{y}}^{\mathrm{HT}} \tag{3.20}$$

The empirical Wiener filter attenuates the DCT coefficients as:

$$\hat{x}_i^{\mathrm{Wie}} = \frac{\left(\hat{x}_i^{\mathrm{HT}}\right)^2}{\left(\hat{x}_i^{\mathrm{HT}}\right)^2 + \sigma^2} x_i \tag{3.21}$$

The empirical Wiener estimate is thus:

$$\hat{\boldsymbol{y}}^{\mathrm{HT}} = D\hat{\boldsymbol{x}}^{\mathrm{Wie}} \tag{3.22}$$

3.6.2 Transform Domain Patch Processing

Given an image $\boldsymbol{Y}$ of size $M \times M$, we extract overlapping patches $\boldsymbol{P}_{i,j}$ of size $p \times p$ centered at pixel (i, j):

$$\boldsymbol{P}_{i,j} = \boldsymbol{Y}\left[i - \left\lfloor \frac{p}{2} \right\rfloor : i + \left\lfloor \frac{p}{2} \right\rfloor, j - \left\lfloor \frac{p}{2} \right\rfloor : j + \left\lfloor \frac{p}{2} \right\rfloor\right] \tag{3.23}$$

For each patch $\boldsymbol{P}_{i,j}$, we apply the following procedure:

1. **Vectorization**: Convert patch to vector $\boldsymbol{p}_{i,j} \in \mathbb{R}^{p^2}$
2. **Transformation**: Apply orthogonal transform $\tilde{\boldsymbol{p}}_{i,j} = \boldsymbol{T}\boldsymbol{p}_{i,j}$
3. **Preliminary Estimation**: Obtain initial estimate $\hat{\tilde{\boldsymbol{p}}}_{i,j}^{(0)}$ using a simple denoising method
4. **Wiener Filtering**: Apply empirical Wiener filter coefficient-wise
5. **Inverse Transform**: Reconstruct patch $\hat{\boldsymbol{p}}_{i,j} = \boldsymbol{T}^{-1}\hat{\tilde{\boldsymbol{p}}}_{i,j}$

3.7 The Sparsity Problem

While orthonormal bases provide computational convenience and guarantee unique representations, they suffer from a fundamental limitation: *no single orthonormal basis can provide sparse representations for all signals of interest.*

Example 3.7.1 (DCT Basis Limitation): Consider a signal $\boldsymbol{s}_0 \in \mathbb{R}^n$ that admits a sparse representation with respect to the Discrete Cosine Transform (DCT) basis $\boldsymbol{D}_{\text{DCT}}$:

$$\boldsymbol{s_0} = \boldsymbol{D}_{\text{DCT}}\boldsymbol{x_0} \tag{3.24}$$

where $\boldsymbol{x_0}$ is sparse (most entries are zero).

Now consider the modified signal:

$$\boldsymbol{s} = \boldsymbol{s_0} + \lambda \boldsymbol{e_j} \tag{3.25}$$

where $\boldsymbol{e_j}$ is the j-th canonical basis vector and $\lambda \in \mathbb{R}$ is a scaling factor.

The DCT representation of $\boldsymbol{s}$ becomes:

$$\boldsymbol{x} = \boldsymbol{D}_{\text{DCT}}^T\boldsymbol{s} = \boldsymbol{D}_{\text{DCT}}^T\boldsymbol{s_0} + \lambda \boldsymbol{D}_{\text{DCT}}^T\boldsymbol{e_j} = \boldsymbol{x_0} + \lambda \boldsymbol{D}_{\text{DCT}}^T\boldsymbol{e_j} \tag{3.26}$$

Since $\boldsymbol{D}_{\text{DCT}}^T\boldsymbol{e_j}$ is typically dense (all entries are non-zero), the addition of a single spike destroys the sparsity of the representation.

3.8 Experimental Demonstration

The experimental verification of this limitation involves:

1. Generate a sparse signal $\boldsymbol{s_0}$ with respect to DCT basis
2. Add a single spike: $\boldsymbol{s} = \boldsymbol{s_0} + \lambda \boldsymbol{e_j}$
3. Compute DCT coefficients of both signals
4. Observe the loss of sparsity in the modified signal

The results consistently show that the addition of a single spike causes all DCT coefficients to become significant, effectively destroying the sparse structure that denoising algorithms rely upon.

4 Introduction to Dictionary Learning

The solution to the sparsity limitation lies in abandoning the constraint of orthonormality and embracing redundancy. Instead of using a single $n \times n$ orthonormal basis, we construct an $n \times m$ dictionary matrix $\boldsymbol{D}$ where $m > n$.

Definition 4.1 (Overcomplete Dictionary): An *overcomplete dictionary* is a matrix $\boldsymbol{D} \in \mathbb{R}^{n \times m}$ with $m > n$ such that:

$$\text{span}\{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_m\} = \mathbb{R}^n \tag{4.1}$$

where $\boldsymbol{d}_i$ are the columns of $\boldsymbol{D}$.

For the DCT-spike example, we construct the overcomplete dictionary by concatenating the DCT basis with the **canonical basis**:

$$\boldsymbol{D} = (\boldsymbol{D}_{\text{DCT}} \;\; \boldsymbol{I}) \in \mathbb{R}^{n \times 2n} \tag{4.2}$$

This construction ensures that:

- Signals sparse in DCT domain remain sparse
- Signals sparse in canonical domain remain sparse
- Mixed signals (DCT-sparse + spikes) admit sparse representations

Example 4.1 (Sparse Representation with Overcomplete Dictionary): Consider the signal $\boldsymbol{s} = \boldsymbol{s}_0 + \lambda \boldsymbol{e}_j$ where $\boldsymbol{s}_0 = \boldsymbol{D}_{\text{DCT}} \boldsymbol{x}_0$ with sparse $\boldsymbol{x}_0$.

The representation with respect to the overcomplete dictionary is:

$$\boldsymbol{s} = \boldsymbol{D} \begin{pmatrix} \boldsymbol{x}_0 \\ \lambda \boldsymbol{e}_j \end{pmatrix} \tag{4.3}$$

The coefficient vector $\begin{pmatrix} \boldsymbol{x}_0 \\ \lambda \boldsymbol{e}_j \end{pmatrix} \in \mathbb{R}^{2n}$ is sparse, containing only the non-zero entries of $\boldsymbol{x}_0$ plus the single entry λ at position j in the second block.

4.1 Theoretical Properties of Overcomplete Systems

Theorem 4.1.1 (Rouché-Capelli Theorem): Consider the linear system $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{s}$ where $\boldsymbol{D} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{s} \in \mathbb{R}^n$. The system admits a solution if and only if:

$$\text{rank}(\boldsymbol{D}) = \text{rank}((\boldsymbol{D} \;\; \boldsymbol{s})) \tag{4.4}$$

When $m > n$ and $\text{rank}(\boldsymbol{D}) = n$, the system has infinitely many solutions forming an affine subspace of dimension $m - n$.

Theorem 4.1.2 (Solution Space Dimension): If $\boldsymbol{D} \in \mathbb{R}^{n \times m}$ with $m > n$ and $\text{rank}(\boldsymbol{D}) = n$, then for any $\boldsymbol{s} \in \mathbb{R}^n$, the solution set of $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{s}$ forms an affine subspace of dimension $m - n$.

The abundance of solutions in overcomplete systems necessitates additional criteria for solution selection. This is where **regularization theory** becomes essential.

4.2 Regularization and Sparse Recovery

Definition 4.2.1 (Regularization): Given an **ill-posed** problem $\boldsymbol{Dx} = \boldsymbol{s}$ with multiple solutions, *regularization* involves solving:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} J(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{Dx} = \boldsymbol{s} \tag{4.5}$$

where $J : \mathbb{R}^m \to \mathbb{R}_+$ is a regularization functional encoding our prior knowledge about the desired solution.

4.2.1 ℓ_2 Regularization: Ridge Regression

The most mathematically tractable regularization is the ℓ_2 norm:

$$J(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{x}\|_2^2 = \frac{1}{2} \sum_{i=1}^{m} x_i^2 \tag{4.6}$$

This leads to the constrained optimization problem:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{Dx} = \boldsymbol{s} \tag{4.7}$$

Alternatively, we can formulate the unconstrained version:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{Dx} - \boldsymbol{s}\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{x}\|_2^2 \tag{4.8}$$

Theorem 4.2.1.1 (Ridge Regression Solution): The solution to the ridge regression problem:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{Dx} - \boldsymbol{s}\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{x}\|_2^2 \tag{4.9}$$

is given by:

$$\hat{\boldsymbol{x}} = \left(\boldsymbol{D}^T\boldsymbol{D} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{D}^T \boldsymbol{s} \tag{4.10}$$

where $\lambda > 0$ ensures the matrix $\left(\boldsymbol{D}^T\boldsymbol{D} + \lambda \boldsymbol{I}\right)$ is invertible.

Proof: Define the objective function:

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{Dx} - \boldsymbol{s}\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{x}\|_2^2 \tag{4.11}$$

Expanding the squared norms:

$$\begin{aligned} f(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{Dx} - \boldsymbol{s})^T(\boldsymbol{Dx} - \boldsymbol{s}) + \frac{\lambda}{2}\boldsymbol{x}^T\boldsymbol{x} \\ &= \frac{1}{2}\boldsymbol{x}^T\boldsymbol{D}^T\boldsymbol{Dx} - \boldsymbol{Dx}^T\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T\boldsymbol{s} + \frac{\lambda}{2}\boldsymbol{x}^T\boldsymbol{x} \end{aligned} \tag{4.12}$$

Taking the gradient with respect to $\boldsymbol{x}$:

$$\nabla f(\boldsymbol{x}) = \boldsymbol{D}^T\boldsymbol{Dx} - \boldsymbol{D}^T\boldsymbol{s} + \lambda \boldsymbol{x} \tag{4.13}$$

Setting $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$:

$$(\boldsymbol{D}^T\boldsymbol{D} + \lambda \boldsymbol{I})\boldsymbol{x} = \boldsymbol{D}^T\boldsymbol{s} \tag{4.14}$$

Since $\lambda > 0$, the matrix $\left(\boldsymbol{D}^T\boldsymbol{D} + \lambda \boldsymbol{I}\right)$ is positive definite and therefore invertible, yielding the stated solution.

4.2.2 Limitations of ℓ_2 Regularization

While ℓ_2 regularization provides a computationally efficient solution, it does not promote sparsity.

The solution $\hat{\boldsymbol{x}}$ typically has all non-zero entries, which contradicts our goal of sparse representation.

⚠ Sparsity vs. ℓ_2 Regularization **attention 4.2.1**

The ℓ_2 norm penalizes large coefficients but does not drive small coefficients to zero. For sparse recovery, we need regularization functionals that promote sparsity, such as the ℓ_1 norm or ℓ_0 pseudo-norm.

4.3 Towards Sparsity: ℓ_0 and ℓ_1 Regularization

4.3.1 The ℓ_0 "Norm" and True Sparsity

The most natural regularization for sparse recovery is the ℓ_0 "norm" (technically a pseudo-norm):

$$\|\boldsymbol{x}\|_0 = |\{i : x_i \neq 0\}| \tag{4.15}$$

This counts the number of non-zero entries in $\boldsymbol{x}$. The corresponding optimization problem:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0 \quad \text{subject to} \quad \boldsymbol{D}\boldsymbol{x} = \boldsymbol{s} \tag{4.16}$$

directly seeks the sparsest representation.

4.3.2 Computational Challenges

The ℓ_0 minimization problem is NP-hard in general, making it computationally intractable for large-scale problems. This has led to the development of convex relaxations and approximation algorithms.

5 Sparse Coding In ℓ_0 sense

5.1 The Sparse Coding Problem

Given the desire for sparse representations, a natural formulation is to seek the sparsest possible $\boldsymbol{\alpha}$ such that:

$$\boldsymbol{x} = D\boldsymbol{\alpha} \tag{5.1}$$

This leads to the following optimization problem:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \|\boldsymbol{\alpha}\|_0 \quad \text{subject to} \quad \boldsymbol{x} = D\boldsymbol{\alpha} \tag{5.2}$$

This is often referred to as the $\boldsymbol{P_0}$ **problem**.

> **Goal:** Among all solutions $\boldsymbol{\alpha}$ such that $D\boldsymbol{\alpha} = \boldsymbol{x}$, select the sparsest one.

Challenge: The ℓ_0 norm is non-convex, discontinuous, and leads to combinatorial complexity. Solving Equation (5.2) exactly is NP-hard in general.

5.1.1 Union of Subspaces Interpretation

Let us assume $D \in \mathbb{R}^{m\times n}$ is a dictionary with $n > m$, i.e., an overcomplete dictionary.

Suppose we restrict $\boldsymbol{\alpha}$ to have at most s non-zero entries. Then the image $D\boldsymbol{\alpha}$ lies in a subspace spanned by s columns of D.

Definition 5.1.1.1 (Sparsity-Induced Subspace): If $\boldsymbol{\alpha}$ has $\|\boldsymbol{\alpha}\|_0 \le s$, then $\boldsymbol{x} = D\boldsymbol{\alpha}$ lies in a subspace $S_\omega := \operatorname{span}(D_\omega)$, where ω is the support of $\boldsymbol{\alpha}$ and D_ω denotes the sub-matrix of D restricted to columns indexed by ω.

Therefore, the space of all s-sparse representations is the union of all such s-dimensional subspaces:

$$\mathcal{M}_s := \bigcup_{\omega\subset\{1,\dots,n\},\ |\omega|\le s} \operatorname{span}(D_\omega) \tag{5.3}$$

> **Interpretation:** Sparse modeling corresponds to finding the best subspace (among exponentially many) in which to approximate the signal $\boldsymbol{x}$.

Key Point: Unlike PCA (which projects onto a single global subspace), sparse coding projects onto a *union of low-dimensional subspaces*, selected adaptively based on the input $\boldsymbol{x}$.

5.1.2 A 2D Illustration of Sparsity

Suppose we are working in $\mathbb{R}^2$ and D has 3 atoms:

$$D = (\boldsymbol{d}_1 \ \ \boldsymbol{d}_2 \ \ \boldsymbol{d}_3), \quad D \in \mathbb{R}^{2\times 3} \tag{5.4}$$

Each $\boldsymbol{d}_i$ is a column vector in $\mathbb{R}^2$.

Let $\boldsymbol{x} \in \mathbb{R}^2$ be a signal we wish to approximate. If we restrict $\|\boldsymbol{\alpha}\|_0 = 1$, then the approximant $D\boldsymbol{\alpha}$ must lie along one of the directions $\boldsymbol{d}_1$, $\boldsymbol{d}_2$, or $\boldsymbol{d}_3$.

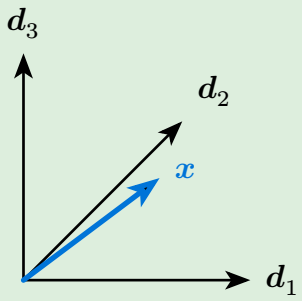

Figure 5.1: Approximation of $\boldsymbol{x}$ via projections onto sparse atoms

Interpretation:

- We seek the atom $\boldsymbol{d}_i$ such that the projection of $\boldsymbol{x}$ onto $\text{span}(\boldsymbol{d}_i)$ minimizes the residual.
- This is the best 1-sparse approximation.

In general: If $\|\boldsymbol{\alpha}\|_0 \leq s$, the approximation lives in a union of $\binom{n}{s}$ subspaces.

5.1.3 Combinatorial Intractability

Why is P_0 hard? To find the optimal s-sparse representation of $\boldsymbol{x}$, one must:

1. Enumerate all subsets $\omega \subset \{1, ..., n\}$ of size s
2. Solve the least squares problem:

$$\boldsymbol{\alpha}_\omega = \arg\min_{\boldsymbol{z}\in\mathbb{R}^s} \|D_\omega \boldsymbol{z} - \boldsymbol{x}\|_2^2 \tag{5.5}$$

3. Select the best ω minimizing the residual.

The number of subsets grows exponentially:

$$\#\text{subspaces} = \binom{n}{s} \sim \left(n\frac{e}{s}\right)^s \tag{5.6}$$

Illustrative Example: Let $n = 1000$, $s = 20$. Then:

$$\binom{1000}{20} \approx 10^{34} \tag{5.7}$$

Assuming a billion operations per second, exhaustive search would take more time than the age of the universe.

> **Conclusion:** The ℓ_0 sparse coding problem is combinatorially explosive. Efficient approximations are necessary.

5.2 Matching Pursuit Algorithm

The **Matching Pursuit (MP)** algorithm embodies the greedy principle for sparse coding:
Input: Signal $\boldsymbol{y}$, dictionary $\boldsymbol{D}$ (normalized), stopping criterion
Output: Sparse representation $\boldsymbol{x}$

1. **Initialize:**

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \mathbf{0} \quad \text{(coefficient vector)} \\ \boldsymbol{r}^{(0)} &= \boldsymbol{y} \qquad\quad \text{(residual)} \\ \Omega^{(0)} &= \varnothing \qquad\quad \text{(active set)} \\ k &= 0 \quad \text{(iteration counter)} \end{aligned} \tag{5.8}$$

2. **Sweep Stage:** For each atom $j = 1, ..., n$, compute the approximation error:

$$E_j^{(k)} = \left\| \boldsymbol{r}^{(k)} - \langle \boldsymbol{d}_j, \boldsymbol{r}^k \rangle \boldsymbol{d}_j \right\|_2^2 \tag{5.9}$$

3. **Atom Selection:** Choose the atom with minimum error:

$$j^* = \arg\min_{j=1,\dots,n} E_j^{(k)} \tag{5.10}$$

Equivalently (by maximizing correlation):

$$j^* = \arg\max_{j=1,\dots,n} \langle \boldsymbol{d}_j, \boldsymbol{r}^k \rangle \tag{5.11}$$

4. **Coefficient Update:** Compute the projection coefficient:

$$z_{j^*}^{(k)} = \langle \boldsymbol{d}_{j^*}, \boldsymbol{r}^k \rangle \tag{5.12}$$

5. **Solution Update:**

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + z_{j^*}^{(k)} \boldsymbol{e}_{j^*} \tag{5.13}$$

where $\boldsymbol{e}_{j^*}$ is the j^*-th standard basis vector.

6. **Residual Update:**

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - z_{j^*}^{(k)} \boldsymbol{d}_{j^*} \tag{5.14}$$

7. **Active Set Update:**

$$\Omega^{(k+1)} = \Omega^{(k)} \cup \{j^*\} \tag{5.15}$$

8. **Stopping Criteria:** Terminate if:
   - $|\Omega^{(k+1)}| \geq k_{\max}$ (maximum sparsity reached)
   - $\left\| \boldsymbol{r}^{(k+1)} \right\|_2 \leq \varepsilon$ (residual threshold met)

   Otherwise, set $k \leftarrow k + 1$ and return to step 2.

5.2.1 Residual Monotonicity

The Matching Pursuit algorithm produces a monotonically decreasing sequence of residual norms:

$$\left\|\boldsymbol{r}^{(k+1)}\right\|_2 \leq \left\|\boldsymbol{r}^{(k)}\right\|_2 \tag{5.16}$$

with strict inequality unless $\boldsymbol{r}^{(k)}$ is orthogonal to all dictionary atoms.

5.2.2 Atom Reselection

Unlike orthogonal methods, Matching Pursuit may select the same atom multiple times in successive iterations. This occurs because:

1. The algorithm does not enforce orthogonality of residuals to previously selected atoms
2. Residual components may align with previously selected atoms after updates
3. This can lead to slower convergence compared to orthogonal variants

5.2.3 Convergence Analysis

For any finite dictionary $\boldsymbol{D}$ and signal $\boldsymbol{y}$, the Matching Pursuit algorithm converges in the sense that:

$$\lim_{k\to\infty} \left\|\boldsymbol{r}^{(k)}\right\|_2 = \min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}\|_2 \tag{5.17}$$

Furthermore, if $\boldsymbol{y} \in \text{span}(\boldsymbol{D})$, then the algorithm achieves exact recovery in finite steps.

5.2.4 Approximation Quality

While Matching Pursuit provides computational tractability, it may not achieve the globally optimal sparse solution. The quality of approximation depends on the coherence structure of the dictionary.

The **coherence** of a dictionary $\boldsymbol{D}$ with normalized columns is:

$$\mu(\boldsymbol{D}) = \max_{i\neq j} |\boldsymbol{d}_i^T \boldsymbol{d}_j| \tag{5.18}$$

Under certain conditions on dictionary coherence and signal sparsity, Matching Pursuit provides approximation guarantees. Specifically, if the true sparse representation has sparsity k and the dictionary satisfies appropriate coherence conditions, then MP recovers a solution with controlled approximation error.

5.2.5 Computational Complexity

Each iteration of Matching Pursuit requires:

- $\mathcal{O}(mn)$ operations for the sweep stage (computing all correlations)
- $\mathcal{O}(m)$ operations for residual update
- Total per-iteration complexity: $\mathcal{O}(mn)$

For k iterations, the total complexity is $\mathcal{O}(kmn)$, which is polynomial and practically feasible.

5.3 Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a variant of Matching Pursuit that enforces orthogonality of the residuals to previously selected atoms. This leads to improved convergence properties and better approximation quality.

The OMP algorithm embodies the greedy principle for sparse coding with orthogonal projections:

Input: Signal $\boldsymbol{y}$, dictionary $\boldsymbol{D}$ (normalized), stopping criterion
Output: Sparse representation $\boldsymbol{x}$

1. **Initialize:**

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \mathbf{0} \quad \text{(coefficient vector)} \\ \boldsymbol{r}^{(0)} &= \boldsymbol{y} \qquad\qquad \text{(residual)} \\ \Omega^{(0)} &= \varnothing \qquad\qquad \text{(active set)} \\ k &= 0 \quad \text{(iteration counter)} \end{aligned} \tag{5.19}$$

2. **Atom Selection:** Choose the atom with maximum correlation:

$$j^* = \arg\max_{j=1,\dots,n} |\langle \boldsymbol{d}_j, \boldsymbol{r}^{(k)} \rangle| \tag{5.20}$$

3. **Active Set Update:**

$$\Omega^{(k+1)} = \Omega^{(k)} \cup \{j^*\} \tag{5.21}$$

4. **Orthogonal Projection:** Solve the least squares problem over the active set:

$$\boldsymbol{\alpha}_{\Omega}^{(k+1)} = \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{D}_{\Omega}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 \tag{5.22}$$

   where $\boldsymbol{D}_{\Omega}$ is the sub-matrix of $\boldsymbol{D}$ with columns indexed by $\Omega^{(k+1)}$.

5. **Solution Update:**

$$\boldsymbol{x}_j^{(k+1)} = \begin{cases} \alpha_j^{(k+1)} & \text{if } j \in \Omega^{(k+1)} \\ 0 & \text{otherwise} \end{cases} \tag{5.23}$$

6. **Residual Update:** Compute the orthogonal residual:

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{y} - \boldsymbol{D}_{\Omega}\boldsymbol{\alpha}_{\Omega}^{(k+1)} \tag{5.24}$$

7. **Stopping Criteria:** Terminate if:
   - $|\Omega^{(k+1)}| \geq k_{\max}$ (maximum sparsity reached)
   - $\left\|\boldsymbol{r}^{(k+1)}\right\|_2 \leq \varepsilon$ (residual threshold met)

   Otherwise, set $k \leftarrow k+1$ and return to step 2.

5.4 OMP-Based Image Denoising

The integration of OMP into image denoising frameworks requires careful consideration of patch processing, dictionary design, and aggregation strategies.

Natural images exhibit strong local correlations but varying global statistics. The patch-based approach decomposes the image into overlapping patches, each processed independently:

Algorithm 2: OMP-Based Image Denoising

procedure OMP-Image-Denoising($\boldsymbol{Y}$, $\boldsymbol{D}$, s)
  ▷ **Input:** Noisy image $\boldsymbol{Y} \in \mathbb{R}^{N \times M}$, dictionary $\boldsymbol{D} \in \mathbb{R}^{n \times m}$, sparsity level s
  ▷ **Output:** Denoised image $\hat{\boldsymbol{Y}}$

  ▷ Patch Extraction:
  ▷ For image $\boldsymbol{Y} \in \mathbb{R}^{N \times M}$, extract patches $(\boldsymbol{y}_i,\ i = 1, ..., P)$, P is the number of patches
  ▷ where each $\boldsymbol{y}_i \in \mathbb{R}^n$ represents a vectorized $\sqrt{n} \times \sqrt{n}$ patch
  for $i = 1, ..., P$ **do**
    $\boldsymbol{y}_i \leftarrow$ extract $\sqrt{n} \times \sqrt{n}$ patch and vectorize
  end

  ▷ Mean Computation and Centering:
  for $i = 1, ..., P$ **do**
    ▷ Compute mean: $\mu_i = \frac{1}{n} \sum_{j=1}^{n} y_{i,j}$
    $\mu_i \leftarrow \frac{1}{n} \sum_{j=1}^{n} y_{i,j}$

    ▷ Center patch: $\tilde{\boldsymbol{y}}_i = \boldsymbol{y}_i - \mu_i \mathbf{1}$
    $\tilde{\boldsymbol{y}}_i \leftarrow \boldsymbol{y}_i - \mu_i \mathbf{1}$
    ▷ where $\mathbf{1} \in \mathbb{R}^n$ is the vector of all ones
  end

  ▷ Sparse Coding:
  for $i = 1, ..., P$ **do**
    ▷ Apply OMP to each mean-centered patch
    $\hat{\boldsymbol{\alpha}}_i \leftarrow \text{OMP}(\boldsymbol{D}, \tilde{\boldsymbol{y}}_i, s)$
  end

  ▷ Reconstruction:
  for $i = 1, ..., P$ **do**
    ▷ Compute denoised patches by adding back the mean
    $\hat{\boldsymbol{x}}_i \leftarrow \boldsymbol{D}\hat{\boldsymbol{\alpha}}_i + \mu_i \mathbf{1}$
  end

  ▷ Aggregation:
  ▷ Reconstruct the full image by averaging overlapping reconstructions
  ▷ at each pixel location
  $\hat{\boldsymbol{Y}} \leftarrow \text{Aggregate}\left(\{\hat{\boldsymbol{x}}_i\}_{i=1}^{P}\right)$

  return $\hat{\boldsymbol{Y}}$
end

5.5 Linearity Analysis of Sparse Coding Algorithms

A fundamental question in sparse coding concerns the linearity properties of the resulting algorithms. An algorithm $\mathcal{A}$ is considered linear if and only if it satisfies:

Definition 5.5.1 (Linear Algorithm): An algorithm $\mathcal{A} : \mathbb{R}^m \to \mathbb{R}^n$ is linear if and only if for all $\alpha, \beta \in \mathbb{R}$ and $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^m$:

$$\mathcal{A}(\alpha \boldsymbol{y}_1 + \beta \boldsymbol{y}_2) = \alpha \mathcal{A}(\boldsymbol{y}_1) + \beta \mathcal{A}(\boldsymbol{y}_2) \tag{5.25}$$

5.5.1 Nonlinearity of OMP

Despite the final solution taking the form of a linear projection:

$$\hat{\boldsymbol{x}}_{\mathcal{S}} = \left(\boldsymbol{D}_{\mathcal{S}}^T \boldsymbol{D}_{\mathcal{S}}\right)^{-1} \boldsymbol{D}_{\mathcal{S}}^T \boldsymbol{y} \tag{5.26}$$

the OMP algorithm is fundamentally nonlinear due to the adaptive selection of the support set $\mathcal{S}$.

Proposition 5.5.1.1 (Nonlinearity of OMP): The OMP algorithm is nonlinear because the support selection depends on the input signal: $\mathcal{S}(\boldsymbol{y})$ is a function of $\boldsymbol{y}$.

Consider two signals $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ that would result in different support sets under OMP. The support of $\boldsymbol{y}_1 + \boldsymbol{y}_2$ may differ from the union of individual supports, violating the linearity condition.

Comparison with Linear Denoising Methods

Fixed linear methods like convolution-based filtering or PCA projection onto the first k principal components are linear but less adaptive. The nonlinearity of OMP enables signal-dependent subspace selection, providing superior performance for sparse signals.

5.6 Uniqueness Guarantees for Sparse Solutions

The following theorem provides conditions under which the solution to the ℓ_0 constraint problem is unique.

Theorem 5.6.1 (Uniqueness of ℓ_0 Solutions): Consider the system $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{y}$ where $\boldsymbol{D} \in \mathbb{R}^{\{m \times n\}}$. If there exists a solution $\hat{\boldsymbol{x}}$ such that:

$$\|\hat{\boldsymbol{x}}\|_0 < \frac{1}{2} \operatorname{spark}(\boldsymbol{D}) \tag{5.27}$$

then $\hat{\boldsymbol{x}}$ is the unique solution to the ℓ_0 constraint problem.

Proof: Suppose, for the sake of contradiction, that there exists another solution $\tilde{\boldsymbol{x}} \neq \hat{\boldsymbol{x}}$ such that $\boldsymbol{D}\tilde{\boldsymbol{x}} = \boldsymbol{y}$.

Since both $\hat{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}$ satisfy the linear system:

$$\begin{aligned} \boldsymbol{D}\hat{\boldsymbol{x}} &= \boldsymbol{y} \\ \boldsymbol{D}\tilde{\boldsymbol{x}} &= \boldsymbol{y} \end{aligned} \tag{5.28}$$

Subtracting these equations yields:

$$\boldsymbol{D}(\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}) = \boldsymbol{0} \tag{5.29}$$

This shows that $\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}$ is a non-zero solution to the homogeneous system. By Lemma 12.5.1:

$$\operatorname{spark}(\boldsymbol{D}) \leq \|\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|_0 \tag{5.30}$$

The definition of spark see Section 12.5.

Using the triangle inequality for the ℓ_0 pseudo-norm:

$$\|\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|_0 \leq \|\hat{\boldsymbol{x}}\|_0 + \|\tilde{\boldsymbol{x}}\|_0 \tag{5.31}$$

Combining these inequalities:

$$\operatorname{spark}(\boldsymbol{D}) \leq \|\hat{\boldsymbol{x}}\|_0 + \|\tilde{\boldsymbol{x}}\|_0 \tag{5.32}$$

Since $\tilde{\boldsymbol{x}}$ is also assumed to be a solution to the ℓ_0 problem, and $\hat{\boldsymbol{x}}$ is the optimal solution:

$$\|\tilde{\boldsymbol{x}}\|_0 \geq \|\hat{\boldsymbol{x}}\|_0 \tag{5.33}$$

Therefore:

$$\operatorname{spark}(\boldsymbol{D}) \leq 2\|\hat{\boldsymbol{x}}\|_0 \tag{5.34}$$

This contradicts our assumption that $\|\hat{\boldsymbol{x}}\|_0 < \frac{1}{2} \operatorname{spark}(\boldsymbol{D})$. Hence, $\hat{\boldsymbol{x}}$ is unique.

⚠ Practical Limitations **attention 5.6.1**

While theoretically elegant, the uniqueness conditions are often too restrictive in practice:

- Computing $\operatorname{spark}(\boldsymbol{D})$ is computationally intractable for large matrices
- The bound $\|\hat{\boldsymbol{x}}\|_0 < \frac{1}{2} \operatorname{spark}(\boldsymbol{D})$ is often very conservative
- Real-world signals may not satisfy the sparsity requirements

5.7 Application to Image Inpainting

Image inpainting addresses the reconstruction of missing or corrupted pixels in digital images. Using sparse coding theory, we can formulate inpainting as a sparse reconstruction problem.

Consider an image patch $\boldsymbol{s}_0 \in \mathbb{R}^n$ (vectorized) and its corrupted version $\boldsymbol{s} \in \mathbb{R}^n$ where some pixels are missing or corrupted. The relationship between them can be expressed as:

$$\boldsymbol{s} = \boldsymbol{\Omega}\boldsymbol{s}_0 \tag{5.35}$$

where $\boldsymbol{\Omega} \in \mathbb{R}^{\{n\times n\}}$ is a diagonal matrix with:

$$\Omega_{ii} = \begin{cases} 1 & \text{if pixel } \ i \ \text{ is known} \\ 0 & \text{if pixel } \ i \ \text{ is missing} \end{cases} \tag{5.36}$$

Assuming the original patch admits a sparse representation:

$$\boldsymbol{s}_0 = \boldsymbol{D}\boldsymbol{x}_0 \tag{5.37}$$

where $\boldsymbol{D} \in \mathbb{R}^{n\times m}$ is an overcomplete dictionary and $\boldsymbol{x}_0$ is sparse, the corrupted patch becomes:

$$\boldsymbol{s} = \boldsymbol{\Omega}\boldsymbol{D}\boldsymbol{x}_0 = \boldsymbol{D}_\Omega\boldsymbol{x}_0 \tag{5.38}$$

where $\boldsymbol{D}_\Omega = \boldsymbol{\Omega}\boldsymbol{D}$ represents the "inpainted dictionary."

The key insight is that the spark of the inpainted dictionary relates to the original dictionary:

Proposition 5.7.1 (Spark of Inpainted Dictionary): For the inpainted dictionary $\boldsymbol{D}_\Omega = \boldsymbol{\Omega}\boldsymbol{D}$:

$$\operatorname{spark}(\boldsymbol{D}_\Omega) \geq \operatorname{spark}(\boldsymbol{D}) \tag{5.39}$$

Proof: Removing rows (zeroing out pixels) from a matrix cannot decrease the spark, as linear dependencies between columns are preserved or potentially eliminated.

Let $\boldsymbol{s}_0$ be an image patch with sparse representation $\boldsymbol{s}_0 = \boldsymbol{D}\boldsymbol{x}_0$ where $\|\boldsymbol{x}_0\|_0 < \frac{1}{2}\operatorname{spark}(\boldsymbol{D}_\Omega)$. Then:

1. The sparse coding problem $\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0$ subject to $\boldsymbol{D}_\Omega\boldsymbol{x} = \boldsymbol{s}$ has a unique solution $\boldsymbol{x}_0$
2. The reconstruction $\hat{\boldsymbol{s}}_0 = \boldsymbol{D}\boldsymbol{x}_0$ perfectly recovers the original patch

Proof: The proof follows directly from Theorem 5.6.1 applied to the inpainted dictionary $\boldsymbol{D}_\Omega$.

Sparse Coding Inpainting Algorithm

Input: Corrupted image patch $\boldsymbol{s}$, dictionary $\boldsymbol{D}$, mask $\boldsymbol{\Omega}$

1. **Construct inpainted dictionary:** $\boldsymbol{D}_\Omega = \boldsymbol{\Omega}\boldsymbol{D}$
2. **Solve sparse coding:** $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0$ subject to $\boldsymbol{D}_\Omega\boldsymbol{x} = \boldsymbol{s}$
3. **Reconstruct patch:** $\hat{\boldsymbol{s}}_0 = \boldsymbol{D}\hat{\boldsymbol{x}}$

Output: Inpainted patch $\hat{\boldsymbol{s}}_0$

In practice, step 2 is solved using OMP with the inpainted dictionary:

$$\hat{\boldsymbol{x}} = \operatorname{OMP}(\boldsymbol{D}_\Omega, \boldsymbol{s}, K) \tag{5.40}$$

where K is a predetermined sparsity level. The key insight is that the synthesis step uses the original dictionary $\boldsymbol{D}$, not the inpainted dictionary $\boldsymbol{D}_\Omega$.

6 Sparse Coding In ℓ_1 sense

6.1 The ℓ_1 Optimization Problem

The ℓ_0 norm, defined as the number of non-zero components in a vector, provides the most intuitive measure of sparsity. However, optimization problems involving the ℓ_0 norm are NP-hard due to their combinatorial nature. This computational intractability necessitates the exploration of alternative sparsity-promoting norms that maintain favorable optimization properties.

The ℓ_1 optimization problem for sparse coding can be formulated in two equivalent ways:

Definition 6.1.1 (Constrained ℓ_1 Problem):

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha}\|_1 \quad \text{subject to} \quad \|\boldsymbol{D\alpha} - \boldsymbol{x}\|_2 \leq \varepsilon \tag{6.1}$$

Definition 6.1.2 (Regularized ℓ_1 Problem):

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\|\boldsymbol{D\alpha} - \boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1 \tag{6.2}$$

6.2 Problem Components Analysis

6.2.1 Data Fidelity Term

The term $\frac{1}{2}\|\boldsymbol{D\alpha} - \boldsymbol{x}\|_2^2$ serves as the data fidelity term, ensuring that the solution $\boldsymbol{\alpha}$ produces a reconstruction $\boldsymbol{D\alpha}$ that is close to the observed signal $\boldsymbol{x}$.

Proposition 6.2.1.1 (Properties of Data Fidelity Term): The data fidelity term $g(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{D\alpha} - \boldsymbol{x}\|_2^2$ is:

1. Convex (as a composition of convex functions)
2. Differentiable with gradient $\nabla g(\boldsymbol{\alpha}) = \boldsymbol{D}^T(\boldsymbol{D\alpha} - \boldsymbol{x})$
3. Strongly convex if $\boldsymbol{D}$ has full column rank

6.2.2 Regularization Term

The term $\lambda\|\boldsymbol{\alpha}\|_1$ acts as a regularization term, promoting sparsity in the solution.

Proposition 6.2.2.1 (Properties of ℓ_1 Regularization): The regularization term $h(\boldsymbol{\alpha}) = \|\boldsymbol{\alpha}\|_1$ is:

1. Convex
2. Non-differentiable at $\alpha_i = 0$ for any component i
3. Promotes sparsity through its geometric properties

6.3 Proximal Gradient Methods

6.3.1 Proximal Operator

For the composite optimization problem $\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) + g(\boldsymbol{\alpha})$, where f is differentiable and g is convex but not necessarily differentiable, we can use the proximal operator to handle the non-differentiable part (See Section 13.2 for details).

Definition 6.3.1.1 (Proximal Operator): The proximal operator of a convex function g is defined as:

$$\text{prox}_{\lambda g}(\boldsymbol{v}) = \arg\min_{\boldsymbol{x}} \left\{ \frac{1}{2\lambda} \|\boldsymbol{x} - \boldsymbol{v}\|_2^2 + g(\boldsymbol{x}) \right\} \tag{6.3}$$

The aim of the proximal operator is to find a point that minimizes $g(x)$, while being close to v.

6.3.2 Proximal Gradient Descent

The proximal gradient descent algorithm iteratively updates the solution by combining the gradient step of the differentiable part and the proximal operator of the non-differentiable part.

For the problem $\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) + g(\boldsymbol{\alpha})$, the update rule is: For $k = 0, 1, 2, ...$ until convergence:

$$\boldsymbol{\alpha}^{k+1} = \text{prox}_{tg}\big(\boldsymbol{\alpha}^k - t\nabla f(\boldsymbol{\alpha}^k)\big) \tag{6.4}$$

where t is a step size parameter. Usually, t is chosen based on the Lipschitz constant of ∇f. $t = \frac{1}{L}$, where L is the Lipschitz constant equal to the largest eigenvalue of $\boldsymbol{D}^T\boldsymbol{D}$ (See Section 13.1 for details).

6.3.3 Iterative Soft Thresholding Algorithm (ISTA)

Theorem 6.3.3.1 (Soft Thresholding): The proximal operator of the ℓ_1 norm is given by the soft thresholding operator:

$$\left[\text{prox}_{\lambda\|\cdot\|_1}(\boldsymbol{v})\right]_i = \text{sign}(v_i)\max\{|v_i| - \lambda, 0\} \tag{6.5}$$

where $\text{sign}(v_i)$ is the sign function.

Proof: The proximal operator problem becomes:

$$\min_{\boldsymbol{x}} \left\{ \frac{1}{2\lambda} \|\boldsymbol{x} - \boldsymbol{v}\|_2^2 + \|\boldsymbol{x}\|_1 \right\} \tag{6.6}$$

For each component i, we solve:

$$\min_{x_i} g_{i(x_i)} = \frac{1}{2\lambda}(x_i - v_i)^2 + |x_i| \tag{6.7}$$

The function $g_{i(x_i)}$ is convex but non-differentiable at $x_i = 0$. We use the subdifferential calculus:

- For $x_i > 0$: $\partial g_{i(x_i)} = \frac{1}{\lambda}(x_i - v_i) + 1$
- For $x_i < 0$: $\partial g_{i(x_i)} = \frac{1}{\lambda}(x_i - v_i) - 1$
- For $x_i = 0$: $\partial g_{i(0)} = -\frac{v_i}{\lambda} + [-1, 1]$

Setting the subdifferential to zero:

Case 1: If $x_i^* > 0$, then $\frac{1}{\lambda}(x_i^* - v_i) + 1 = 0$, giving $x_i^* = v_i - \lambda$. This is valid only if $v_i - \lambda > 0$, i.e., $v_i > \lambda$.

Case 2: If $x_i^* < 0$, then $\frac{1}{\lambda}(x_i^* - v_i) - 1 = 0$, giving $x_i^* = v_i + \lambda$. This is valid only if $v_i + \lambda < 0$, i.e., $v_i < -\lambda$.

Case 3: If $x_i^* = 0$, then $0 \in -\frac{v_i}{\lambda} + [-1, 1]$, which means $-1 \leq -\frac{v_i}{\lambda} \leq 1$, or equivalently $-\lambda \leq v_i \leq \lambda$.

Combining all cases:

$$x_i^* = \begin{cases} v_i - \lambda \text{ if } v_i > \lambda \\ 0 \qquad \text{if} - \lambda \leq v_i \leq \lambda \\ v_i + \lambda \text{ if } v_i < -\lambda \end{cases} \tag{6.8}$$

This can be written compactly as:

$$x_i^* = \text{sign}(v_i) \max\{|v_i| - \lambda, 0\} \tag{6.9}$$

The solution is the soft thresholding operator due to the subdifferential analysis of the absolute value function.

Algorithm 3: Iterative Soft Thresholding Algorithm (ISTA)

procedure ISTA($\boldsymbol{D}$, $\boldsymbol{x}$, λ, ε)
  ▷ **Input:**
  ▷ $\boldsymbol{D} \in \mathbb{R}^{m \times n}$: dictionary/measurement matrix
  ▷ $\boldsymbol{x} \in \mathbb{R}^m$: observation vector
  ▷ $\lambda > 0$: regularization parameter
  ▷ $\varepsilon > 0$: convergence tolerance
  ▷ **Output:**
  ▷ $\boldsymbol{\alpha}^* \in \mathbb{R}^n$: sparse coefficient vector

  ▷ Initialize
  $\boldsymbol{\alpha}^0 \leftarrow \mathbf{0}$
  $L \leftarrow$ largest eigenvalue of $\boldsymbol{D}^T\boldsymbol{D}$
  $t \leftarrow \frac{1}{L}$
  $k \leftarrow 0$

  while not converged **do**
    ▷ Gradient step
    $\boldsymbol{u}^k \leftarrow \boldsymbol{\alpha}^k - t\boldsymbol{D}^T\big(\boldsymbol{D}\boldsymbol{\alpha}^k - \boldsymbol{x}\big)$

    ▷ Proximal step (soft thresholding)
    for $i = 1 < n$ **do**
      $\alpha_i^{k+1} \leftarrow \text{sign}\big(u_i^k\big) \cdot \max\big\{|u_i^k| - \lambda t, 0\big\}$
    end

    ▷ Check convergence
    if $\|\boldsymbol{\alpha}^{k+1} - \boldsymbol{\alpha}^k\|_2 < \varepsilon$ **then**
      return $\boldsymbol{\alpha}^{k+1}$
    end
    $k \leftarrow k + 1$
  end
  return $\boldsymbol{\alpha}^k$
end

6.3.4 Backtracking Line Search

When computing eigenvalues is impractical, adaptive step size selection via backtracking provides a robust alternative:

Algorithm 4: Backtracking Line Search for ISTA

procedure BACKTRACKING-LINE-SEARCH-ISTA($\boldsymbol{D}$, $\boldsymbol{x}$, λ, ε)
  ▷ **Input:**
  ▷ $\boldsymbol{D} \in \mathbb{R}^{m \times n}$: dictionary/measurement matrix
  ▷ $\boldsymbol{x} \in \mathbb{R}^m$: observation vector
  ▷ $\lambda > 0$: regularization parameter
  ▷ $\varepsilon > 0$: convergence tolerance
  ▷ **Parameters:** $\beta \in (0, 1)$ (typically $\beta = 0.5$), $\eta \in (0, 1)$ (typically $\eta = 0.9$)

  ▷ **Output:**
  ▷ $\boldsymbol{\alpha}^* \in \mathbb{R}^n$: sparse coefficient vector

  ▷ Initialize
  $t \leftarrow t_0$
  ▷ initial step size, e.g., $t_0 = 1$
  $\boldsymbol{\alpha}^0 \leftarrow \mathbf{0}$
  $k \leftarrow 0$

  while not converged **do**
    ▷ Compute gradient step
    $\boldsymbol{g}^k \leftarrow \boldsymbol{D}^T\left(\boldsymbol{D}\boldsymbol{\alpha}^k - \boldsymbol{x}\right)$

    ▷ Compute proximal operator (soft thresholding)
    $\boldsymbol{\alpha}^+ \leftarrow \operatorname{prox}_{t\lambda \|\cdot\|_1}\left(\boldsymbol{\alpha}^k - t\boldsymbol{g}^k\right)$

    ▷ Define objective function: $F(\boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{D}\boldsymbol{\alpha} - \boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{\alpha}\|_1$
    ▷ Backtracking line search
    while $F(\boldsymbol{\alpha}^+) > F\left(\boldsymbol{\alpha}^k\right) - \frac{\eta}{t} \|\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^k\|_2^2$ **do**
      ▷ Decrease step size
      $t \leftarrow \beta t$
      ▷ Recompute proximal step with new step size
      $\boldsymbol{\alpha}^+ \leftarrow \operatorname{prox}_{t\lambda \|\cdot\|_1}\left(\boldsymbol{\alpha}^k - t\boldsymbol{g}^k\right)$
    end

    ▷ Check convergence
    if $\|\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^k\|_2 < \varepsilon$ **then**
      return $\boldsymbol{\alpha}^+$
    end
    ▷ Update for next iteration
    $\boldsymbol{\alpha}^{k+1} \leftarrow \boldsymbol{\alpha}^+$
    $k \leftarrow k + 1$
  end
  return $\boldsymbol{\alpha}^k$
end

6.4 FISTA

While ISTA achieves $O(\frac{1}{k})$ convergence, Nesterov's acceleration technique can improve this to $O(\frac{1}{k^2})$ without additional computational cost per iteration. This acceleration is achieved through a momentum-like mechanism that exploits the history of iterates.

Algorithm 5: Fast Iterative Shrinkage-Thresholding Algorithm (FISTA)

procedure FISTA($\boldsymbol{D}$, $\boldsymbol{x}$, λ, ε)
  ▷ **Input:**
  ▷ $\boldsymbol{D} \in \mathbb{R}^{m \times n}$: dictionary/measurement matrix
  ▷ $\boldsymbol{x} \in \mathbb{R}^m$: observation vector
  ▷ $\lambda > 0$: regularization parameter
  ▷ $\varepsilon > 0$: convergence tolerance

  ▷ **Output:**
  ▷ $\boldsymbol{\alpha}^* \in \mathbb{R}^n$: sparse coefficient vector

  ▷ Initialize
  $\boldsymbol{\alpha}^0 \leftarrow \mathbf{0}$
  $\boldsymbol{y}^1 \leftarrow \boldsymbol{\alpha}^0$
  $t_1 \leftarrow 1$
  $L \leftarrow$ largest eigenvalue of $\boldsymbol{D}^T\boldsymbol{D}$
  $\alpha \leftarrow \frac{1}{L}$
  $k \leftarrow 1$

  while not converged **do**
    ▷ (a) Proximal gradient step:
    $\boldsymbol{\alpha}^k \leftarrow \text{prox}_{\alpha\lambda\|\cdot\|_1}\big(\boldsymbol{y}^k - \alpha\boldsymbol{D}^T\big(\boldsymbol{D}\boldsymbol{y}^k - \boldsymbol{x}\big)\big)$

    ▷ (b) Update momentum parameter:

    $t_{k+1} \leftarrow \frac{1+\sqrt{1+4t_k^2}}{2}$

    ▷ (c) Compute extrapolated point:

    $\boldsymbol{y}^{k+1} \leftarrow \boldsymbol{\alpha}^k + \frac{t_k - 1}{t_{k+1}}\big(\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}\big)$

    ▷ Check convergence
    if $\left\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}\right\|_2 < \varepsilon$ **then**
      return $\boldsymbol{\alpha}^k$
    end
    $k \leftarrow k + 1$
  end
  return $\boldsymbol{\alpha}^k$
end

6.4.1 The Momentum Sequence

The sequence $\{t_k\}$ satisfies the recurrence relation:

$$t_{k+1}^2 - t_{k+1} - t_k^2 = 0 \tag{6.10}$$

This yields the closed-form expression:

$$t_k = \frac{k+1}{2} + O(1) \approx \frac{k}{2} \quad \text{for large} \quad k \tag{6.11}$$

6.4.2 The Extrapolation Step

The extrapolation coefficient:

$$\beta_k = \frac{t_k - 1}{t_{k+1}} \approx \frac{k-2}{k+1} \to 1 \quad \text{as} \quad k \to \infty \tag{6.12}$$

This creates an "overshoot" effect that accelerates convergence by anticipating the trajectory of the iterates.

6.4.3 Convergence Rate

Theorem 6.4.3.1 (FISTA Convergence Rate): For FISTA with step size $\alpha = \frac{1}{L}$, the following bound holds:

$$F\big(\boldsymbol{x}^k\big) - F(\boldsymbol{x}^*) \leq \frac{2L\big\|\boldsymbol{x}^0 - \boldsymbol{x}^*\big\|_2^2}{(k+1)^2} \tag{6.13}$$

The $O\big(\frac{1}{k^2}\big)$ rate is optimal for first-order methods on the class of convex functions with Lipschitz continuous gradients.

7 Structured Sparsity

The classical sparse coding framework seeks representations with minimal non-zero coefficients, treating each coefficient independently. However, in many practical applications, the *locations* of non-zero coefficients exhibit inherent structure that standard sparsity models fail to exploit. Structured sparsity extends the sparse coding paradigm by incorporating prior knowledge about coefficient patterns, leading to more robust and interpretable representations.

Consider the fundamental **sparse coding problem**:

$$\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{x}\|_0 \tag{7.1}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ represents the observed signal, $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ denotes an overcomplete dictionary with $n > m$, and $\boldsymbol{x} \in \mathbb{R}^n$ contains the sparse coefficients.

This formulation promotes sparsity uniformly across all coefficients. However, structured sparsity recognizes that coefficients often exhibit dependencies or groupings that reflect the underlying data generation process.

1. **Multi-channel Signal Processing**: When processing multiple signals acquired from the same source (e.g., multi-electrode recordings, hyperspectral imaging), the active atoms in the dictionary tend to be consistent across channels.
2. **Texture Analysis**: Patches extracted from textured images share common structural elements, suggesting that their sparse representations should utilize similar dictionary atoms.
3. **Statistical Variable Selection**: In high-dimensional regression problems, predictors often form natural groups (e.g., dummy variables for categorical features, measurements from the same instrument).

7.1 Joint Sparsity and Mixed Norms

7.1.1 Problem Formulation

Let $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_L] \in \mathbb{R}^{m \times L}$ represent a collection of L signals sharing common structural properties. The joint sparse coding problem seeks a coefficient matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_L] \in \mathbb{R}^{n \times L}$ such that:

$$\boldsymbol{Y} \approx \boldsymbol{D}\boldsymbol{X} \tag{7.2}$$

The key insight is that columns of $\boldsymbol{X}$ should not only be individually sparse but should also share common support patterns.

7.1.2 Mixed Norms for Matrices

To enforce joint sparsity, we introduce the (p, q)-mixed norm for matrices.

Definition 7.1.2.1 ($\ell_{p,q}$ Mixed Norm): For a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times L}$, the (p, q)-mixed norm is defined as:

$$\|\boldsymbol{X}\|_{p,q} = \left(\sum_{i=1}^{n} \left(\sum_{j=1}^{L} |X_{ij}|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}} \tag{7.3}$$

This can be interpreted as:

$$\begin{aligned} \|\boldsymbol{X}\|_{p,q} &= \|\left(\|\boldsymbol{x}^{(1)}\|_p, \|\boldsymbol{x}^{(2)}\|_p, ..., \|\boldsymbol{x}^{(n)}\|_p\right)^T\|_q \\ &= \|\boldsymbol{v}\|_q \end{aligned} \tag{7.4}$$

where $\boldsymbol{x}^{(i)}$ denotes the i-th row of $\boldsymbol{X}$, and $v_i = \|\boldsymbol{x}^{(i)}\|_p$.

Frobenius Norm: When $p = q = 2$:

$$\|\boldsymbol{X}\|_{2,2} = \sqrt{\sum_{i,j} |X_{ij}|^2} = \|\boldsymbol{X}\|_F \tag{7.5}$$

Entry-wise Norms: When $p = q$, the mixed norm reduces to the vectorized ℓ_p norm:

$$\|\boldsymbol{X}\|_{p,p} = \|\text{vec}(\boldsymbol{X})\|_p \tag{7.6}$$

Joint Sparsity Norm: The $\ell_{2,1}$ norm:

$$\|\boldsymbol{X}\|_{2,1} = \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{L} |X_{ij}|^2} = \sum_{i=1}^{n} \|\boldsymbol{x}^{(i)}\|_2 \tag{7.7}$$

Proposition 7.1.2.1 (Joint Sparsity Property): The $\ell_{2,1}$ norm promotes row-sparsity in $\boldsymbol{X}$, meaning entire rows become zero. This corresponds to selecting the same dictionary atoms across all signals.

7.2 Joint Sparse Coding

The joint sparse coding problem with the $\ell_{2,1}$ norm regularization is formulated as:

$$\min_{\boldsymbol{X}\in\mathbb{R}^{n\times L}} \frac{1}{2}\left\|\boldsymbol{Y}-\boldsymbol{D}\boldsymbol{X}\right\|_F^2+\lambda\left\|\boldsymbol{X}\right\|_{2,1} \tag{7.8}$$

This optimization problem combines a smooth, convex data fidelity term with a non-smooth but convex regularizer, making it amenable to proximal gradient methods.

The iterative solution via proximal gradient descent follows:

$$\boldsymbol{X}^{(k+1)}=\operatorname{prox}_{\gamma\lambda\left\|\cdot\right\|_{2,1}}\left(\boldsymbol{X}^{(k)}-\gamma\nabla f\left(\boldsymbol{X}^{(k)}\right)\right) \tag{7.9}$$

where $f(\boldsymbol{X})=\frac{1}{2}\left\|\boldsymbol{Y}-\boldsymbol{D}\boldsymbol{X}\right\|_F^2$ is the smooth part, and $\gamma>0$ is the step size.

The key computational challenge lies in evaluating the proximal mapping:

$$\operatorname{prox}_{\tau\left\|\cdot\right\|_{2,1}}(\boldsymbol{Z})=\arg\min_{\boldsymbol{X}}\left\{\tau\left\|\boldsymbol{X}\right\|_{2,1}+\frac{1}{2}\left\|\boldsymbol{X}-\boldsymbol{Z}\right\|_F^2\right\} \tag{7.10}$$

Theorem 7.2.1 (Row-wise Separability of $\ell_{2,1}$ Proximal Mapping): The proximal mapping of the $\ell_{2,1}$ norm can be computed row-wise:

$$\left[\operatorname{prox}_{\tau\left\|\cdot\right\|_{2,1}}(\boldsymbol{Z})\right]_i=\operatorname{shrink}_\tau^{(2)}\left(\boldsymbol{z}^{(i)}\right) \tag{7.11}$$

where $\boldsymbol{z}^{(i)}$ is the i-th row of $\boldsymbol{Z}$, and $\operatorname{shrink}_\tau^{(2)}$ is the multivariate soft-thresholding operator.

Proof: The objective function in Equation (7.10) can be rewritten as:

$$\begin{aligned}\tau\left\|\boldsymbol{X}\right\|_{2,1}+\frac{1}{2}\left\|\boldsymbol{X}-\boldsymbol{Z}\right\|_F^2&=\tau\sum_{i=1}^{n}\|\boldsymbol{x}^{(i)}\|_2+\frac{1}{2}\sum_{i=1}^{n}\|\boldsymbol{x}^{(i)}-\boldsymbol{z}^{(i)}\|_2^2\\&=\sum_{i=1}^{n}\left[\tau\left\|\boldsymbol{x}^{(i)}\right\|_2+\frac{1}{2}\left\|\boldsymbol{x}^{(i)}-\boldsymbol{z}^{(i)}\right\|_2^2\right]\end{aligned} \tag{7.12}$$

Since the objective decomposes into independent row-wise problems, the minimization can be performed separately for each row.

Definition 7.2.1 (Multivariate Soft-Thresholding): For a vector $\boldsymbol{v}\in\mathbb{R}^L$ and threshold $\tau>0$:

$$\operatorname{shrink}_\tau^{(2)}(\boldsymbol{v})=\begin{cases}\frac{\boldsymbol{v}}{\|}\boldsymbol{v}\|_2\cdot\max(0,\|\boldsymbol{v}\|_2-\tau) & \text{if } \boldsymbol{v}\neq\boldsymbol{0}\\ \boldsymbol{0} & \text{if } \boldsymbol{v}=\boldsymbol{0}\end{cases} \tag{7.13}$$

This operator exhibits two key behaviors:

1. **Nullification**: If $\|\boldsymbol{v}\|_2\leq\tau$, the entire vector is set to zero.
2. **Shrinkage**: If $\|\boldsymbol{v}\|_2>\tau$, the vector is scaled down while preserving its direction.

7.3 Group Sparsity and Extensions

Consider a dictionary $\boldsymbol{D}$ partitioned into G groups:

$$\boldsymbol{D} = [\boldsymbol{D}_1 \mid \boldsymbol{D}_2 \mid \ldots \mid \boldsymbol{D}_G] \tag{7.14}$$

where $\boldsymbol{D}_g \in \mathbb{R}^{m \times n_g}$ contains atoms corresponding to group g, and $\sum_{g=1}^{G} n_g = n$.

The coefficient vector $\boldsymbol{x}$ is correspondingly partitioned:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_{[1]} \\ \boldsymbol{x}_{[2]} \\ \vdots \\ \boldsymbol{x}_{[G]} \end{pmatrix} \tag{7.15}$$

where $\boldsymbol{x}_{[g]} \in \mathbb{R}^{n_g}$ contains coefficients for group g.

The group sparse coding problem seeks representations that activate entire groups rather than individual atoms:

$$\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \lambda \sum_{g=1}^{G} w_g \, \|\boldsymbol{x}_{[g]}\|_2 \tag{7.16}$$

where $w_g > 0$ are **group-specific weights**, typically set as $w_g = \sqrt{n_g}$ to account for group size differences.

⚠ Group Selection Property **attention 7.3.1**

The group LASSO penalty induces sparsity at the group level: either all coefficients within a group are zero, or the group is active with potentially multiple non-zero coefficients.

The proximal mapping for the group LASSO penalty decomposes into group-wise operations:

$$\left[\operatorname{prox}_{\tau \sum_g w_g \|\cdot\|_2}(\boldsymbol{z})\right]_{[g]} = \operatorname{shrink}^{(2)}_{\tau w_g}\left(\boldsymbol{z}_{[g]}\right) \tag{7.17}$$

This allows efficient computation by applying multivariate soft-thresholding to each group independently.

7.4 LASSO

In the statistical setting, we observe m samples with response variable y_i and n predictors $\boldsymbol{x}_i = [x_{i1}, x_{i2}, ..., x_{in}]^T$. The linear model assumes:

$$y_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i = 1, 2, ..., m \tag{7.18}$$

In matrix notation:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{7.19}$$

where $\boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is the design matrix, $\boldsymbol{\beta} \in \mathbb{R}^n$ contains regression coefficients, and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.

The classical least squares estimator:

$$\hat{\boldsymbol{\beta}}_{\text{LS}} = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{7.20}$$

This estimator is unbiased ($\mathbb{E}\left[\hat{\boldsymbol{\beta}}_{\text{LS}}\right] = \boldsymbol{\beta}$) but may have high variance, especially when predictors are correlated or n is large relative to m.

The Least Absolute Shrinkage and Selection Operator (LASSO) adds ℓ_1 regularization:

$$\hat{\boldsymbol{\beta}}_{\text{LASSO}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\} \tag{7.21}$$

Theorem 7.4.1 (Variable Selection Property): For sufficiently large λ, the LASSO estimator $\hat{\boldsymbol{\beta}}_{\text{LASSO}}$ contains exact zeros, performing automatic variable selection.

The LASSO introduces bias to reduce variance:

$$\begin{aligned} \text{MSE}\left(\hat{\boldsymbol{\beta}}\right) &= \mathbb{E}\left[\left[|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2\right]\right] \\ &= \|\mathbb{E}\left[\hat{\boldsymbol{\beta}}\right] - \boldsymbol{\beta}\|_2^2 + \text{trace}\left(\text{Var}\left(\hat{\boldsymbol{\beta}}\right)\right) \\ &= \text{Bias}^2 + \text{Variance} \end{aligned} \tag{7.22}$$

While OLS minimizes bias, LASSO accepts some bias in exchange for substantially reduced variance through sparsity.

Theorem 7.4.2 (LASSO in High Dimensions): When $n > m$ (more predictors than observations), OLS is not unique. However, LASSO provides a unique sparse solution for appropriate $\lambda > 0$.

This property makes LASSO particularly valuable in modern applications like genomics, where the number of features vastly exceeds the sample size.

7.5 Elastic Net

The LASSO has two notable limitations:

1. In the $n > m$ setting, it selects at most m variables
2. When predictors are highly correlated, LASSO tends to arbitrarily select one from each group

The Elastic Net addresses these issues by combining ℓ_1 and ℓ_2 penalties:

$$\hat{\boldsymbol{\beta}}_{\text{EN}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \left\| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \right\|_2^2 + \lambda_1 \left\| \boldsymbol{\beta} \right\|_1 + \lambda_2 \left\| \boldsymbol{\beta} \right\|_2^2 \right\} \tag{7.23}$$

7.5.1 Geometric Interpretation

The constraint region for Elastic Net is:

$$\mathcal{C}_{\text{EN}} = \left\{ \boldsymbol{\beta} : \alpha \left\| \boldsymbol{\beta} \right\|_1 + (1-\alpha) \left\| \boldsymbol{\beta} \right\|_2^2 \leq t \right\} \tag{7.24}$$

This creates a compromise between the diamond-shaped ℓ_1 ball and the spherical ℓ_2 ball, maintaining sparsity-inducing corners while allowing smoother boundaries.

7.5.2 Proximal Gradient Solution

The Elastic Net optimization can be solved efficiently using proximal gradient methods. The key insight is that the ℓ_2 penalty can be absorbed into the smooth part:

$$f(\boldsymbol{\beta}) = \frac{1}{2} \left\| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \right\|_2^2 + \lambda_2 \left\| \boldsymbol{\beta} \right\|_2^2 \tag{7.25}$$

with gradient:

$$\nabla f(\boldsymbol{\beta}) = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) + 2\lambda_2\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X} + 2\lambda_2\boldsymbol{I})\boldsymbol{\beta} - \boldsymbol{X}^T\boldsymbol{y} \tag{7.26}$$

The proximal gradient update becomes:

$$\boldsymbol{\beta}^{(k+1)} = \text{shrink}_{\gamma\lambda_1}\left(\boldsymbol{\beta}^{(k)} - \gamma\nabla f\left(\boldsymbol{\beta}^{(k)}\right)\right) \tag{7.27}$$

where shrink_τ is the element-wise soft-thresholding operator.

7.6 Algorithmic Summary

Problem	**Regularizer**	**Proximal Operator**
Standard Sparsity	$\left\| \boldsymbol{x} \right\|_1$	$\text{shrink}_\tau(x_i) = \text{sign}(x_i)\max(0, \lvert x_i \rvert - \tau)$
Joint Sparsity	$\left\| \boldsymbol{X} \right\|_{2,1}$	Row-wise multivariate soft-thresholding
Group Sparsity	$\sum_g w_g \left\| \boldsymbol{x}_{[g]} \right\|_2$	Group-wise multivariate soft-thresholding
Elastic Net	$\lambda_1 \left\| \boldsymbol{x} \right\|_1 + \lambda_2 \left\| \boldsymbol{x} \right\|_2^2$	Modified soft-thresholding with ℓ_2 in gradient

Table 1: Summary of sparsity-inducing regularizers and their proximal operators

8 Dictionary Learning

8.1 Introduction to Dictionary Learning

Definition 8.1.1 (Dictionary Learning Problem): Given a set of training signals $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N \in \mathbb{R}^n$, the dictionary learning problem seeks to find:
1. A dictionary matrix $\boldsymbol{D} \in \mathbb{R}^{\{n \times m\}}$ with $m > n$ (redundant dictionary)
2. Sparse coefficient vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N \in \mathbb{R}^m$

such that $\boldsymbol{y}_i \approx \boldsymbol{D}\boldsymbol{x}_i$ for all $i = 1, 2, ..., N$, where each $\boldsymbol{x}_i$ has at most T_0 non-zero entries.

Dictionary learning employs a block coordinate descent strategy, alternating between two phases:
1. **Sparse Coding Phase:** Fix D and solve for X
2. **Dictionary Update Phase:** Fix X and update D

8.2 Problem Formulation

Let $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N] \in \mathbb{R}^{n \times N}$ denote the training matrix, where each column represents a training signal. Similarly, let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N] \in \mathbb{R}^{m \times N}$ represent the sparse coefficient matrix.

The dictionary learning problem can be formulated as the following optimization:

$$\min_{\boldsymbol{D}, \boldsymbol{X}} \|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F^2 \quad \text{subject to} \quad \|\boldsymbol{x}_i\|_0 \leq T_0, \quad \forall i = 1, 2, ..., N \tag{8.1}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and $\|\cdot\|_0$ is the ℓ_0 pseudo-norm counting non-zero entries.

(Normalization Constraint:). To resolve scaling ambiguities, we impose the constraint that each column of $\boldsymbol{D}$ has unit ℓ_2 norm:

$$\left\|\boldsymbol{d}_j\right\|_2 = 1, \quad \forall j = 1, 2, ..., m \tag{8.2}$$

The optimization problem Equation (8.1) presents several fundamental challenges:
1. **Non-convexity**: The objective function is non-convex in the joint variables $\boldsymbol{D}, \boldsymbol{X}$, even though it is convex in each variable individually when the other is fixed.
2. **Sparsity Constraint**: The ℓ_0 pseudo-norm constraint is non-convex and combinatorial, making direct optimization intractable.
3. **Solution ambiguity**: Multiple equivalent solutions exist due to:
   - Column permutations of $\boldsymbol{D}$ with corresponding row permutations of $\boldsymbol{X}$
   - Sign ambiguities: $\boldsymbol{d}_j, \boldsymbol{x}_j) \equiv \left(-\boldsymbol{d}_j, -\boldsymbol{x}_j\right)$

8.3 Sparse Coding Phase

The sparse coding phase solves the following problem for each training signal:

$$\boldsymbol{x}_i^{(k+1)} = \arg\min_{\boldsymbol{x}} \left\|\boldsymbol{y}_i - \boldsymbol{D}^{(k)}\boldsymbol{x}\right\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{x}\|_0 \leq T_0 \tag{8.3}$$

This is precisely the sparse coding problem discussed in previous sections, which can be solved using greedy algorithms such as:

- **Orthogonal Matching Pursuit (OMP)**: Iteratively selects atoms that best correlate with the current residual
- **Matching Pursuit (MP)**: Similar to OMP but without orthogonalization
- **Basis Pursuit**: Convex relaxation using ℓ_1 norm

The sparse coding phase is computationally the most expensive part of dictionary learning, as it requires solving N sparse coding problems (one for each training signal) at each iteration.

8.4 Dictionary Update Phase

The dictionary update phase constitutes the core innovation of K-SVD. Rather than updating the entire dictionary simultaneously, K-SVD updates one column at a time while simultaneously updating the corresponding sparse coefficients.

8.4.1 Matrix Factorization Perspective

Consider the error matrix:

$$\boldsymbol{E} = \boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X} \tag{8.4}$$

Using the fundamental matrix identity, we can decompose the product $\boldsymbol{D}\boldsymbol{X}$ as:

$$\boldsymbol{D}\boldsymbol{X} = \sum_{j=1}^{m} \boldsymbol{d}_j \boldsymbol{x}_j^T \tag{8.5}$$

where $\boldsymbol{x}_j^T$ denotes the j-th row of $\boldsymbol{X}$.

8.4.2 Isolated Column Update

To update the j_0-th column of $\boldsymbol{D}$, we rewrite Equation (8.5) as:

$$\boldsymbol{D}\boldsymbol{X} = \sum_{j \neq j_0} \boldsymbol{d}_j \boldsymbol{x}_j^T + \boldsymbol{d}_{j_0} \boldsymbol{x}_{j_0}^T \tag{8.6}$$

Define the error matrix excluding the j_0-th atom:

$$\boldsymbol{E}_{j_0} = \boldsymbol{Y} - \sum_{j \neq j_0} \boldsymbol{d}_j \boldsymbol{x}_j^T \tag{8.7}$$

The update problem becomes:

$$\min_{\boldsymbol{d}_{j_0}, \boldsymbol{x}_{j_0}^T} \left\|\boldsymbol{E}_{j_0} - \boldsymbol{d}_{j_0}\boldsymbol{x}_{j_0}^T\right\|_F^2 \quad \text{subject to} \quad \left\|\boldsymbol{d}_{j_0}\right\|_2 = 1 \tag{8.8}$$

This is a rank-one matrix approximation problem, optimally solved using the Singular Value Decomposition (SVD).

8.4.3 SVD solution

Theorem 8.4.3.1 (Rank-One Matrix Approximation): Let $\boldsymbol{A} \in \mathbb{R}^{\{n \times N\}}$ be given. The solution to

$$\min_{\boldsymbol{u}, \boldsymbol{v}} \left\|\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T\right\|_F^2 \quad \text{subject to} \quad \|\boldsymbol{u}\|_2 = 1 \tag{8.9}$$

is given by $\boldsymbol{u} = \boldsymbol{u}_1$ and $\boldsymbol{v}^T = \sigma_1 \boldsymbol{v}_1^T$, where $\boldsymbol{A} = \sum_{i=1}^{\min(n,N)} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ is the SVD of $\boldsymbol{A}$.

Proof: The Frobenius norm can be expressed as:

$$\begin{aligned} \left\|\boldsymbol{A}-\boldsymbol{u}\boldsymbol{v}^T\right\|_F^2 &= \|\boldsymbol{A}\|_F^2 - 2\ \mathrm{trace}(\boldsymbol{A}^T\boldsymbol{u}\boldsymbol{v}^T) + \left\|\boldsymbol{u}\boldsymbol{v}^T\right\|_F^2 \\ &= \|\boldsymbol{A}\|_F^2 - 2\boldsymbol{v}^T\boldsymbol{A}^T\boldsymbol{u} + \|\boldsymbol{v}\|_2^2 \end{aligned} \tag{8.10}$$

Since $\|\boldsymbol{u}\|_2 = 1$, maximizing $\boldsymbol{v}^T\boldsymbol{A}^T\boldsymbol{u}$ is equivalent to finding the leading singular vectors of $\boldsymbol{A}$.

8.4.4 Sparsity Preservation

A critical challenge in the dictionary update is preserving the sparsity structure of $\boldsymbol{X}$. The naive application of Theorem 8.4.3.1 would yield a dense row vector $\boldsymbol{x}_{j_0}^T$, violating the sparse coding constraint.

Solution - Restricted SVD: K-SVD addresses this by restricting the update to only those training signals that actually use the j_0-th atom:

$$\Omega_{j_0} = \left\{i : x_{j_0,i} \neq 0\right\} \tag{8.11}$$

Define the restricted error matrix:

$$\boldsymbol{E}_{j_0}^R = \boldsymbol{E}_{j_0}\left(:, \Omega_{j_0}\right) \tag{8.12}$$

The restricted update problem becomes:

$$\min_{\boldsymbol{d}_{j_0}, \boldsymbol{x}_{j_0}^R} \left\|\boldsymbol{E}_{j_0}^R - \boldsymbol{d}_{j_0}\left(\boldsymbol{x}_{j_0}^R\right)^T\right\|_F^2 \quad \text{subject to} \quad \left\|\boldsymbol{d}_{j_0}\right\|_2 = 1 \tag{8.13}$$

where $\boldsymbol{x}_{j_0}^R$ contains only the non-zero elements of $\boldsymbol{x}_{j_0}^T$.

Dictionary Update Algorithm for Column j_0:

1. Compute error matrix: $\boldsymbol{E}_{j_0} = \boldsymbol{Y} - \sum_{j \neq j_0} \boldsymbol{d}_j \boldsymbol{x}_j^T$
2. Identify support: $\Omega_{j_0} = \left\{i : x_{j_0,i} \neq 0\right\}$
3. Extract restricted matrix: $\boldsymbol{E}_{j_0}^R = \boldsymbol{E}_{j_0}\left(:, \Omega_{j_0}\right)$
4. Compute SVD: $\boldsymbol{E}_{j_0}^R = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
5. Update dictionary: $\boldsymbol{d}_{j_0} = \boldsymbol{u}_1$
6. Update coefficients: $\boldsymbol{x}_{j_0}^R = \sigma_1 \boldsymbol{v}_1$
7. Restore to full representation: $\boldsymbol{x}_{j_0}^T\left(\Omega_{j_0}\right) = \boldsymbol{x}_{j_0}^R$

Algorithm 6: K-SVD Dictionary Learning Algorithm

procedure K-SVD($\boldsymbol{Y}$, m, T_0, K)
  ▷ **Input:** Training signals $\boldsymbol{Y} \in \mathbb{R}^{n \times N}$, dictionary size m, sparsity level T_0, max iterations K
  ▷ **Output:** Dictionary $\boldsymbol{D} \in \mathbb{R}^{n \times m}$ and sparse coefficients $\boldsymbol{X} \in \mathbb{R}^{m \times N}$

  ▷ Initialize dictionary:
  $\boldsymbol{D}^{(0)} \leftarrow$ random matrix with unit-norm columns
  $k \leftarrow 0$

  ▷ Main K-SVD loop:
  while $k < K \wedge$ not converged **do**
    ▷ SPARSE CODING PHASE:
    ▷ For each training signal $\boldsymbol{y}_i$, solve sparse coding problem
    for $i = 1, ..., N$ **do**
      $\boldsymbol{x}_i^{(k+1)} \leftarrow \arg\min_{\boldsymbol{x}} \left\|\boldsymbol{y}_i - \boldsymbol{D}^{(k)}\boldsymbol{x}\right\|_2^2$ subject to $\|\boldsymbol{x}\|_0 \leq T_0$
      ▷ Use OMP, MP, or other sparse coding algorithm
    end

    ▷ DICTIONARY UPDATE PHASE:
    ▷ Update each dictionary column sequentially
    for $j = 1, ..., m$ **do**
      ▷ (a) Compute error matrix excluding j-th atom:
      $\boldsymbol{E}_j \leftarrow \boldsymbol{Y} - \sum_{\ell \neq j} \boldsymbol{d}_\ell \boldsymbol{x}_\ell^T$

      ▷ (b) Identify support set:
      $\Omega_j \leftarrow \left\{i : x_{j,i} \neq 0\right\}$

      ▷ (c) Check if atom is used:
      if $\Omega_j = \emptyset$ **then**
        ▷ Replace unused atom with random unit vector
        $\boldsymbol{d}_j \leftarrow$ random unit vector
      end
      else
        ▷ (d) Extract restricted error matrix:
        $\boldsymbol{E}_j^R \leftarrow \boldsymbol{E}_j(:, \Omega_j)$

        ▷ (e) Compute SVD of restricted matrix:
        $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V} \leftarrow \mathrm{SVD}\left(\boldsymbol{E}_j^R\right)$

        ▷ (f) Update dictionary column:
        $\boldsymbol{d}_j \leftarrow \boldsymbol{u}_1$
        ▷ First left singular vector

        ▷ (g) Update sparse coefficients:
        $\boldsymbol{x}_j^R \leftarrow \sigma_1 \boldsymbol{v}_1$
        ▷ Scaled first right singular vector

        ▷ (h) Restore to full representation:
        $\boldsymbol{x}_j^T(\Omega_j) \leftarrow \boldsymbol{x}_j^R$
        $\boldsymbol{x}_j^T\left(\mathrm{complement}(\Omega_j)\right) \leftarrow \boldsymbol{0}$
      end
    end

```
        ▷ Check convergence:
        if ‖Y − D^(k+1) X^(k+1)‖_F < ε then
            ▷ Convergence achieved
            BREAK
        end

        k ← k + 1
    end

    return D^(k), X^(k)
end
```

9 Local Polynomial Approximation

Local Polynomial Approximation (LPA) represents a fundamental shift in signal processing methodology, moving away from global sparsity constraints toward localized polynomial modeling. This approach recognizes that many real-world signals exhibit piecewise smooth behavior that can be effectively captured through local polynomial representations.

Traditional signal processing approaches often rely on global assumptions about signal structure, such as sparsity in transformed domains (e.g., DCT, wavelet transforms). While these methods have proven successful in many applications, they impose uniform constraints across the entire signal domain. In contrast, LPA methods adapt to local signal characteristics, providing more flexible and often more accurate approximations.

The transition from sparsity-based to polynomial-based modeling represents a significant conceptual advancement:

- **Sparsity-based approach**: Assumes signal can be represented as a linear combination of few basis functions from a fixed dictionary
- **Polynomial-based approach**: Assumes signal can be locally approximated by polynomials of appropriate degree

This shift enables adaptive processing where each spatial location can have its own optimal approximation parameters, leading to superior performance in regions with varying signal characteristics.

Definition 9.1 (Local Polynomial Approximation): Given a signal $f : \mathbb{R} \to \mathbb{R}$ and a point $x_0 \in \mathbb{R}$, a local polynomial approximation of degree L is a polynomial $P_L(x)$ of the form:

$$P_L(x) = \sum_{j=0}^{L} \beta_j (x - x_0)^j \tag{9.1}$$

that minimizes a local fitting criterion within a neighborhood of x_0.

9.1 Local Polynomial Model

Consider a one-dimensional signal model:

$$y(t) = f(t) + \eta(t) \tag{9.2}$$

where $f(t)$ represents the underlying smooth signal and $\eta(t)$ denotes additive white Gaussian noise with variance σ^2.

For discrete processing, we work with sampled versions. Let $\boldsymbol{y} = [y_1, y_2, ..., y_M]^T$ represent an M -dimensional signal vector extracted from a local neighborhood around a central pixel.

Within a local neighborhood, we assume the signal can be well approximated by a polynomial of degree L:

$$f(t_i) \approx \sum_{j=0}^{L} \beta_j t_i^j, \quad i = 1, 2, ..., M \tag{9.3}$$

where $\left\{\beta_j\right\}_{j=0}^{L}$ are the polynomial coefficients to be estimated, and $\{t_i\}_{i=1}^{M}$ are the spatial locations within the neighborhood.

⚠ Degree Constraint **attention 9.1.1**

The polynomial degree L must satisfy $L+1 \leq M$ to ensure an over-determined system. This constraint prevents overfitting and ensures stable coefficient estimation.

9.1.1 Matrix Formulation

The polynomial approximation can be expressed in matrix form. Define the design matrix $\boldsymbol{T} \in \mathbb{R}^{M\times(L+1)}$:

$$\boldsymbol{T} = \begin{pmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^L \\ 1 & t_2 & t_2^2 & \cdots & t_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_M & t_M^2 & \cdots & t_M^L \end{pmatrix} \tag{9.4}$$

The polynomial coefficient vector is $\boldsymbol{\beta} = [\beta_0, \beta_1, ..., \beta_L]^T \in \mathbb{R}^{L+1}$.

The polynomial approximation becomes:

$$\boldsymbol{f} \approx \boldsymbol{T}\boldsymbol{\beta} \tag{9.5}$$

9.1.2 Least Squares Formulation

The optimal polynomial coefficients are obtained by minimizing the squared approximation error:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{L+1}}{\arg\min} \|\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta}\|^2 \tag{9.6}$$

Theorem 9.1.2.1 (Unweighted LPA Solution): The least squares solution to the local polynomial approximation problem is:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{T}^T\boldsymbol{T})^{-1}\boldsymbol{T}^T\boldsymbol{y} \tag{9.7}$$

provided that $\boldsymbol{T}^T\boldsymbol{T}$ is invertible.

Proof: Taking the derivative of the objective function Equation (9.6) with respect to $\boldsymbol{\beta}$ and setting it to zero:

$$\begin{aligned} \frac{\partial}{\partial\boldsymbol{\beta}}\|\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta}\|^2 &= \frac{\partial}{\partial\boldsymbol{\beta}}(\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta}) \\ &= -2\boldsymbol{T}^T(\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta}) = 0 \end{aligned} \tag{9.8}$$

Solving for $\boldsymbol{\beta}$ yields the normal equations:

$$\boldsymbol{T}^T\boldsymbol{T}\boldsymbol{\beta} = \boldsymbol{T}^T\boldsymbol{y} \tag{9.9}$$

The solution follows directly when $\boldsymbol{T}^T\boldsymbol{T}$ is invertible.

9.2 Weighted Local Polynomial Approximation

In many practical scenarios, not all samples within a neighborhood should contribute equally to the polynomial fit. Samples closer to the center of the neighborhood are typically more relevant for estimating the signal at that location. Weighted LPA addresses this by introducing spatially varying weights.

The weighted LPA problem incorporates a diagonal weight matrix $\boldsymbol{W} \in \mathbb{R}^{M\times M}$:

$$\hat{\boldsymbol{\beta}}_w = \underset{\boldsymbol{\beta}\in\mathbb{R}^{L+1}}{\arg\min} \|\boldsymbol{W}(\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta})\|^2 \tag{9.10}$$

where $\boldsymbol{W} = \text{diag}(w_1, w_2, ..., w_M)$ with $w_i \geq 0$ for all i.

Theorem 9.2.1 (Weighted LPA Solution): The weighted least squares solution is:

$$\hat{\boldsymbol{\beta}}_w = (\boldsymbol{T}^T\boldsymbol{W}^2\boldsymbol{T})^{-1}\boldsymbol{T}^T\boldsymbol{W}^2\boldsymbol{y} \tag{9.11}$$

Proof: The weighted objective function can be written as:

$$\|\boldsymbol{W}(\boldsymbol{y} - \boldsymbol{T}\boldsymbol{\beta})\|^2 = \|\boldsymbol{W}\boldsymbol{y} - \boldsymbol{W}\boldsymbol{T}\boldsymbol{\beta}\|^2 \tag{9.12}$$

This is equivalent to solving the unweighted problem:

$$\underset{\boldsymbol{\beta}}{\arg\min} \left\|\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{T}}\boldsymbol{\beta}\right\|^2 \tag{9.13}$$

where $\tilde{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{y}$ and $\tilde{\boldsymbol{T}} = \boldsymbol{W}\boldsymbol{T}$.

Applying Theorem 9.1.2.1:

$$\begin{aligned}\hat{\boldsymbol{\beta}}_w &= \left(\tilde{\boldsymbol{T}}^T\tilde{\boldsymbol{T}}\right)^{-1}\tilde{\boldsymbol{T}}^T\tilde{\boldsymbol{y}} \\ &= \left((\boldsymbol{W}\boldsymbol{T})^T(\boldsymbol{W}\boldsymbol{T})\right)^{-1}(\boldsymbol{W}\boldsymbol{T})^T\boldsymbol{W}\boldsymbol{y} \\ &= (\boldsymbol{T}^T\boldsymbol{W}^2\boldsymbol{T})^{-1}\boldsymbol{T}^T\boldsymbol{W}^2\boldsymbol{y}\end{aligned} \tag{9.14}$$

Common weight functions include:

1. **Uniform weights**: $w_i = 1$ for all i (reduces to unweighted case)
2. **Gaussian weights**: $w_i = \exp\left(-(t_i - t_0)^2/(2\sigma_w^2)\right)$
3. **Binary weights**: $w_i \in \{0, 1\}$ for adaptive support selection

Example 9.2.1 (Binary Weight Example): Consider a signal with a discontinuity. Using binary weights allows selective processing:

- Left-side filter: $\boldsymbol{w} = [1, 1, 1, 0, 0]^T$
- Right-side filter: $\boldsymbol{w} = [0, 0, 1, 1, 1]^T$

This prevents blurring across discontinuities while maintaining smoothing within homogeneous regions.

9.3 QR Decomposition Approach

Direct computation of $(\boldsymbol{T}^T\boldsymbol{T})^{-1}$ can be numerically unstable, especially when $\boldsymbol{T}$ is ill-conditioned. The QR decomposition provides a numerically stable alternative while revealing the underlying geometric structure of the problem.

Theorem 9.3.1 (QR Decomposition): Any matrix $\boldsymbol{T} \in \mathbb{R}^{M\times(L+1)}$ with $M \geq L+1$ and full column rank can be decomposed as:

$$\boldsymbol{T} = \boldsymbol{Q}\boldsymbol{R} \tag{9.15}$$

where $\boldsymbol{Q} \in \mathbb{R}^{M\times(L+1)}$ has orthonormal columns ($\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_{L+1}$) and $\boldsymbol{R} \in \mathbb{R}^{(L+1)\times(L+1)}$ is upper triangular.

Using the QR decomposition, the LPA solution becomes:

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= (\boldsymbol{T}^T\boldsymbol{T})^{-1}\boldsymbol{T}^T\boldsymbol{y} \\
&= ((\boldsymbol{Q}\boldsymbol{R})^T(\boldsymbol{Q}\boldsymbol{R}))^{-1}(\boldsymbol{Q}\boldsymbol{R})^T\boldsymbol{y} \\
&= (\boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{R})^{-1}\boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{y} \\
&= (\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{y} \\
&= \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y}
\end{aligned} \tag{9.16}$$

The signal estimate is:

$$\hat{\boldsymbol{f}} = \boldsymbol{T}\hat{\boldsymbol{\beta}} = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{y} \tag{9.17}$$

Proposition 9.3.1 (Projection Interpretation): The matrix $\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{Q}^T$ represents the orthogonal projection onto the column space of $\boldsymbol{T}$. The LPA estimate is the orthogonal projection of the noisy signal onto the space of polynomials of degree L.

For weighted LPA, we apply QR decomposition to the weighted design matrix $\boldsymbol{W}\boldsymbol{T}$:

$$\boldsymbol{W}\boldsymbol{T} = \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{R}} \tag{9.18}$$

The weighted solution becomes:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_w &= \tilde{\boldsymbol{R}}^{-1}\tilde{\boldsymbol{Q}}^T\boldsymbol{W}\boldsymbol{y} \\
\hat{\boldsymbol{f}}_w &= \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{Q}}^T\boldsymbol{W}\boldsymbol{y}
\end{aligned} \tag{9.19}$$

9.4 Convolution Implementation

A key insight in LPA is that the estimation can be implemented as a convolution operation, enabling efficient computation across entire signals or images.

Consider the estimation of the signal value at the center of the neighborhood. Let i_c denote the central index. The estimate at this location is:

$$\hat{f}\left(t_{i_c}\right) = \boldsymbol{e}_{i_c}^T \boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{y} \tag{9.20}$$

where $\boldsymbol{e}_{i_c}$ is the unit vector with 1 in the i_c-th position.

Theorem 9.4.1 (Convolution Kernel Formula): The LPA estimation at the central location can be expressed as:

$$\hat{f}\left(t_{i_c}\right) = \sum_{i=1}^{M} h_i y_i = \boldsymbol{h}^T \boldsymbol{y} \tag{9.21}$$

where the convolution kernel is:

$$\boldsymbol{h} = \boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{e}_{i_c} \tag{9.22}$$

The convolution kernel can be computed explicitly as:

$$\boldsymbol{h} = \sum_{j=0}^{L} \beta_j \boldsymbol{q}_j \tag{9.23}$$

where $\boldsymbol{q}_j$ are the columns of $\boldsymbol{Q}$ and:

$$\beta_j = \boldsymbol{q}_j^T \boldsymbol{e}_{i_c} = q_{j,i_c} \tag{9.24}$$

9.5 Special Cases

Example 9.5.1 (Zero-Order Polynomial (Moving Average)): For $L = 0$, the design matrix is $\boldsymbol{T} = \mathbf{1} = [1, 1, ..., 1]^T$.

The QR decomposition gives:

$$\begin{aligned} \boldsymbol{Q} &= \frac{1}{\sqrt{M}}\mathbf{1} \\ \boldsymbol{R} &= \sqrt{M} \end{aligned} \tag{9.25}$$

The convolution kernel becomes:

$$\boldsymbol{h} = \frac{1}{M}\mathbf{1} \tag{9.26}$$

This is the standard moving average filter.

Example 9.5.2 (Weighted Zero-Order Polynomial): For weighted zero-order polynomial with normalized weights $\sum_{i=1}^{M} w_i^2 = 1$:

$$\begin{aligned} \tilde{\boldsymbol{Q}} &= \boldsymbol{w} \\ \tilde{\boldsymbol{R}} &= 1 \end{aligned} \tag{9.27}$$

The convolution kernel is:

$$\boldsymbol{h} = \boldsymbol{w} \tag{9.28}$$

This shows that Gaussian smoothing corresponds to weighted zero-order polynomial fitting.

9.6 Statistical Properties and Performance Analysis

9.6.1 Bias-Variance Decomposition

The mean squared error (MSE) of the LPA estimator can be decomposed into bias and variance components:

$$\mathrm{MSE}\big(\hat{f}(t_0)\big) = \mathrm{Bias}^2\big(\hat{f}(t_0)\big) + \mathrm{Var}\big(\hat{f}(t_0)\big) \tag{9.29}$$

9.6.2 Bias Analysis

Theorem 9.6.2.1 (Bias of LPA Estimator): For a signal $f(t)$ that is $(L+1)$-times differentiable, the bias of the LPA estimator is:

$$\mathrm{Bias}\big(\hat{f}(t_0)\big) = \frac{f^{(L+1)}(t_0)}{(L+1)!} \sum_{i=1}^{M} h_i (t_i - t_0)^{L+1} + O(h^{L+2}) \tag{9.30}$$

where h is the neighborhood size.

Theorem 9.6.2.2 (Bias for Polynomial Signals): If the true signal is a polynomial of degree L or less, the LPA estimator is unbiased.

9.6.3 Variance Analysis

Theorem 9.6.3.1 (Variance of LPA Estimator): For additive white Gaussian noise with variance σ^2, the variance of the LPA estimator is:

$$\mathrm{Var}\big(\hat{f}(t_0)\big) = \sigma^2 \|\boldsymbol{h}\|^2 = \sigma^2 \boldsymbol{e}_{i_c}^T \boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{e}_{i_c} \tag{9.31}$$

The variance decreases as the effective number of samples increases, but the bias may increase due to the larger neighborhood size. This creates a fundamental bias-variance tradeoff.

9.6.4 Optimal Neighborhood Selection

The optimal neighborhood size balances bias and variance:

$$h_{\text{opt}} = \arg\min_{h} [\mathrm{Bias}^2(h) + \mathrm{Var}(h)] \tag{9.32}$$

In practice, this leads to adaptive algorithms that select different neighborhood sizes and shapes based on local signal characteristics.

9.7 Adaptive Neighborhood Selection

Fixed neighborhood sizes and shapes are suboptimal for signals with varying local characteristics. Adaptive methods adjust the approximation parameters based on local signal properties.

9.7.1 Directional Filtering

Binary weights enable directional filtering, which is particularly useful near discontinuities:

Definition 9.7.1.1 (Directional Kernels): A set of directional kernels $\{\boldsymbol{h}_d\}_{d=1}^{D}$ provides estimates along different directions or orientations. Each kernel uses binary weights to select samples from a specific spatial direction.

Example 9.7.1.1 (One-Dimensional Directional Kernels): For a 1D signal with neighborhood size $M = 5$:

- $\boldsymbol{h}_{\text{left}}$: weights $[1, 1, 1, 0, 0]$
- $\boldsymbol{h}_{\text{right}}$: weights $[0, 0, 1, 1, 1]$
- $\boldsymbol{h}_{\text{center}}$: weights $[0, 1, 1, 1, 0]$

9.7.2 Intersection of Confidence Intervals

The Intersection of Confidence Intervals (ICI) rule provides a principled approach for adaptive neighborhood selection:

1. Compute estimates $\left\{\hat{f}_d\right\}$ and confidence intervals $\{\text{CI}_d\}$ for each directional kernel
2. Find the intersection of all confidence intervals: $\text{CI}_{\text{intersect}} = \bigcap_{d=1}^{D} \text{CI}_d$
3. Select the estimator with the largest neighborhood whose confidence interval contains $\text{CI}_{\text{intersect}}$

9.8 The Intersection of Confidence Intervals (ICI) Rule

The ICI rule provides a data-driven approach to support selection based on statistical confidence intervals:

Definition 9.8.1 (Confidence Interval for LPA): For a given support size h and confidence parameter $\gamma > 0$, the confidence interval for $\hat{f}_h(x_0)$ is:

$$\text{CI}_h(x_0) = \left[\hat{f}_h(x_0) - \gamma\sqrt{V(h, x_0)}, \hat{f}_h(x_0) + \gamma\sqrt{V(h, x_0)}\right] \tag{9.33}$$

For Gaussian noise with known variance σ^2, choosing $\gamma = z_{\alpha/2}$ (the $\alpha/2$ quantile of the standard normal distribution) yields a $(1 - \alpha)$ confidence interval.

Theorem 9.8.1 (ICI Principle): Consider a sequence of support sizes $h_1 < h_2 < \cdots < h_K$. Define:

$$\mathcal{J}_k(x_0) = \bigcap_{j=1}^{k} \text{CI}_{h_j}(x_0) \tag{9.34}$$

The optimal scale $k^*(x_0)$ is chosen as:

$$k^*(x_0) = \max\{k : \mathcal{J}_k(x_0) \neq \emptyset\} \tag{9.35}$$

Algorithm 7: Intersection of Confidence Intervals (ICI)

procedure ICI$\Big($signal $\boldsymbol{y} \in \mathbb{R}^n$, polynomial degree p, scales $\{h_k\}_{k=1}^K$, parameter $\gamma\Big)$

▷ **Input:** Signal $\boldsymbol{y} \in \mathbb{R}^n$, polynomial degree p, scales $\{h_k\}_{k=1}^K$, parameter γ

▷ **Output:** Optimal scales $\{k^*(x_i)\}_{i=1}^n$ and estimates $\left\{\hat{f}(x_i)\right\}_{i=1}^n$

▷ Initialization:
for $i = 1, ..., n$ **do**
  $L_0(x_i) \leftarrow -\infty$
  $U_0(x_i) \leftarrow +\infty$
  $k^*(x_i) \leftarrow 0$
end

▷ For each scale $k = 1, 2, ..., K$:
for $i = 1, ..., n$ **do**
  ▷ (a) Compute LPA estimates $\hat{f}_{h_k}(x_i)$ for all i
  for $i = 1, ..., n$ **do**
    $\hat{f}_{h_k}(x_i) \leftarrow$ LPA estimate with support h_k
  end

  ▷ (b) Compute variances $V(h_k, x_i) = \sigma^2 \sum_j h_{j,k}^2(x_i)$
  for $i = 1, ..., n$ **do**
    $V(h_k, x_i) \leftarrow \sigma^2 \sum_j h_{j,k}^2(x_i)$
  end

  ▷ (c) Calculate confidence bounds:
  for $i = 1, ..., n$ **do**
    $l_k(x_i) \leftarrow \hat{f}_{h_k}(x_i) - \gamma\sqrt{V(h_k, x_i)}$
    $u_k(x_i) \leftarrow \hat{f}_{h_k}(x_i) + \gamma\sqrt{V(h_k, x_i)}$
  end

  ▷ (d) Update intersection bounds:
  for $i = 1, ..., n$ **do**
    $L_k(x_i) \leftarrow \max\{L_{k-1}(x_i), l_k(x_i)\}$
    $U_k(x_i) \leftarrow \min\{U_{k-1}(x_i), u_k(x_i)\}$
  end

  ▷ (e) Check intersection validity:
  for $i = 1, ..., n$ **do**
    if $U_k(x_i) < L_k(x_i) \wedge k^*(x_i) = 0$ **then**
      $k^*(x_i) \leftarrow k - 1$
    end
  end
end

▷ Final estimates:
for $i = 1, ..., n$ **do**
  $\hat{f}(x_i) \leftarrow \hat{f}_{h_{k^*(x_i)}}(x_i)$
end
return $\{k^*(x_i)\}_{i=1}^n$, $\left\{\hat{f}(x_i)\right\}_{i=1}^n$

end

10 Robust Fitting

The problem of *robust model fitting* arises ubiquitously in computer vision applications where we must estimate analytical expressions derived from geometric constraints in the presence of outliers. Unlike standard noise, outliers represent data points that significantly deviate from the expected model with statistical characteristics that are fundamentally different from the inlier distribution.

Definition 10.1 (Robust Fitting Problem): Given a set of observations $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the robust fitting problem seeks to estimate parameters $\boldsymbol{\theta} \in \mathbb{R}^p$ of a model $f(\boldsymbol{x}; \boldsymbol{\theta})$ such that:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \rho(r_i(\boldsymbol{\theta})) \tag{10.1}$$

where $r_i(\boldsymbol{\theta}) = y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta})$ is the residual and $\rho : \mathbb{R} \to \mathbb{R}^+$ is a robust loss function.

The choice of ρ fundamentally determines the robustness properties of the estimator. Classical least squares uses $\rho(r) = r^2$, which, as we shall demonstrate, lacks robustness to outliers.

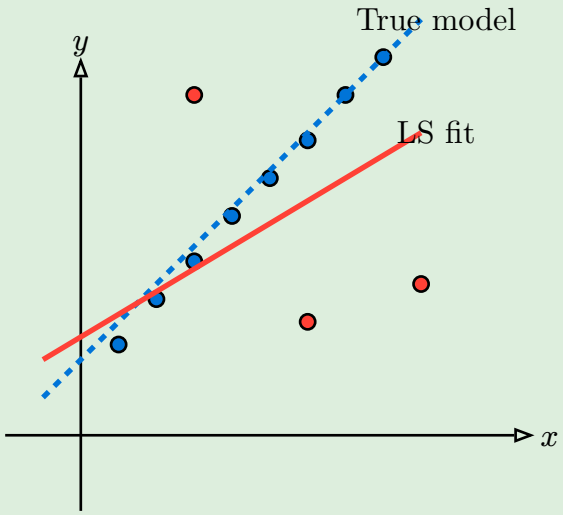

Figure 10.1: Illustration of outlier influence on least squares fitting

10.1 Limitations of Vertical Distance Minimization

We begin with the fundamental problem of fitting a straight line to a set of 2D points. The parametric form of a line is:

$$y = mx + b \tag{10.2}$$

where m is the slope and b is the y-intercept. Given observations $\{(x_i, y_i)\}_{i=1}^n$, ordinary least squares (OLS) seeks:

$$(m^*, b^*) = \arg\min_{m,b} \sum_{i=1}^{n} (y_i - mx_i - b)^2 \tag{10.3}$$

The ordinary least squares approach measures error along the vertical axis, which introduces several fundamental limitations:

1. **Inability to fit vertical lines**: When the line approaches vertical ($m \to \infty$), the formulation breaks down as we cannot express x as a function of y.

2. **Asymmetric treatment of variables**: The choice of dependent vs. independent variable artificially privileges one coordinate axis.
3. **Scale dependency**: The error metric depends on the coordinate system orientation.

10.2 The Direct Linear Transformation (DLT) Algorithm

To overcome the limitations of parametric representations, we adopt the implicit form:

$$ax + by + c = 0 \tag{10.4}$$

where $(a, b, c) \in \mathbb{R}^3$ are the line parameters. This representation elegantly handles all line orientations, including vertical lines (where $b = 0$).

⚠ Scale Ambiguity **attention 10.2.1**

The implicit representation introduces a scale ambiguity: the lines defined by (a, b, c) and $(\lambda a, \lambda b, \lambda c)$ for any $\lambda \neq 0$ are identical. This necessitates a normalization constraint.

10.2.1 Algebraic Error Minimization

Rather than minimizing geometric distance (which leads to nonlinear optimization), we minimize the *algebraic error*:

Definition 10.2.1.1 (Algebraic Error): For a point (x_i, y_i) and line parameters $\boldsymbol{\theta} = [a, b, c]^T$, the algebraic error is:

$$r_i^{\text{alg}(\boldsymbol{\theta})} = ax_i + by_i + c \tag{10.5}$$

Definition 10.2.1.2 (Geometric Error): The geometric error is the perpendicular distance from the point to the line:

$$r_i^{\text{geom}(\boldsymbol{\theta})} = \frac{|ax_i + by_i + c|}{\sqrt{a^2 + b^2}} \tag{10.6}$$

The relationship between algebraic and geometric errors is:

$$r_i^{\text{geom}(\boldsymbol{\theta})} = \frac{\left|r_i^{\text{alg}(\boldsymbol{\theta})}\right|}{\sqrt{a^2 + b^2}} \tag{10.7}$$

10.2.2 Constrained Least Squares Formulation

The DLT problem is formulated as:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{A}\boldsymbol{\theta}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{\theta}\|_2 = 1 \tag{10.8}$$

where the design matrix is:

$$\boldsymbol{A} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix} \in \mathbb{R}^{n \times 3} \tag{10.9}$$

Theorem 10.2.2.1 (DLT Solution via SVD): Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the singular value decomposition of $\boldsymbol{A}$, where:

- $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{3\times 3}$ are orthogonal matrices
- $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \sigma_3)$ with $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$

Then the optimal solution is $\boldsymbol{\theta}^* = \boldsymbol{v}_3$, the last column of $\boldsymbol{V}$.

Proof: Using the orthogonality of $\boldsymbol{U}$ and the substitution $\boldsymbol{w} = \boldsymbol{V}^T\boldsymbol{\theta}$:

$$\begin{aligned} \|\boldsymbol{A}\boldsymbol{\theta}\|_2^2 &= \left\|\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{\theta}\right\|_2^2 \\ &= \left\|\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{\theta}\right\|_2^2 \quad \text{(orthogonal invariance)} \\ &= \|\boldsymbol{\Sigma}\boldsymbol{w}\|_2^2 = \sum_{i=1}^{3} \sigma_i^2 w_i^2 \end{aligned} \tag{10.10}$$

Since $\|\boldsymbol{\theta}\|_2 = \|\boldsymbol{w}\|_2 = 1$, minimizing the objective requires placing all weight on the smallest singular value:

$$\boldsymbol{w}^* = [0, 0, 1]^T \Rightarrow \boldsymbol{\theta}^* = \boldsymbol{V}\boldsymbol{w}^* = \boldsymbol{v}_3 \tag{10.11}$$

Numerical conditioning is crucial for stable DLT computation, especially when dealing with higher-order geometric entities.

Algorithm 8: Direct Linear Transformation (DLT)

procedure DLT$\left(\text{data points } \{(x_i, y_i)\}_{i=1}^n\right)$
  ▷ **Input:** Data points $\{(x_i, y_i)\}_{i=1}^n$
  ▷ **Output:** Line parameters $\boldsymbol{\theta}^* = [a, b, c]^T$

  ▷ Construct design matrix:
  $\boldsymbol{A} \leftarrow \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix} \in \mathbb{R}^{n\times 3}$

  ▷ Compute SVD of design matrix:
  $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V} \leftarrow \text{SVD}(\boldsymbol{A})$
  ▷ where $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$

  ▷ Extract solution from smallest singular vector:
  $\boldsymbol{\theta}^* \leftarrow \boldsymbol{v}_3$
  ▷ last column of $\boldsymbol{V}$ corresponding to smallest singular value

  ▷ Optional: Normalize solution
  if normalization desired **then**
    $\boldsymbol{\theta}^* \leftarrow \frac{\boldsymbol{\theta}^*}{\|\boldsymbol{\theta}^*\|_2}$
  end

  return $\boldsymbol{\theta}^*$
end

10.2.3 Preconditioning for Numerical Stability

Numerical conditioning is crucial for stable DLT computation, especially when dealing with higher-order geometric entities.

Algorithm 9: Normalized DLT

procedure NORMALIZED-DLT$\left(\text{data points } \{(x_i, y_i)\}_{i=1}^{n}\right)$
  ▷ **Input:** Data points $\{(x_i, y_i)\}_{i=1}^{n}$
  ▷ **Output:** Line parameters $\boldsymbol{\theta}^* = [a, b, c]^T$
  ▷ Compute data centroid and scale:
  $|(x) \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$
  $|(y) \leftarrow \frac{1}{n}\sum_{i=1}^{n} y_i$
  ▷ Apply normalization transformation:
  $\boldsymbol{T} \leftarrow \begin{pmatrix} s_x & 0 & -s_x\,|(x) \\ 0 & s_y & -s_y\,|(y) \\ 0 & 0 & 1 \end{pmatrix}$
  ▷ where s_x and s_y are chosen such that the normalized data has unit variance
  ▷ Solve DLT for normalized points:
  for $i = 1, ..., n$ **do**
    $\tilde{\boldsymbol{p}}_i \leftarrow \boldsymbol{T}\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$
  end
  $\tilde{\boldsymbol{\theta}}^* \leftarrow \text{DLT}(\{\tilde{\boldsymbol{p}}_i\})$
  ▷ Denormalize the solution:
  $\boldsymbol{\theta}^* \leftarrow \boldsymbol{T}^T\tilde{\boldsymbol{\theta}}^*$
  return $\boldsymbol{\theta}^*$
end

10.3 Robust Estimation Methods

Definition 10.3.1 (Breakdown Point): The breakdown point ε^* of an estimator is the largest fraction of arbitrarily corrupted observations that the estimator can handle while still producing a bounded estimate close to the true value.

For ordinary least squares, the breakdown point is $\varepsilon^* = 0$, meaning a single outlier with sufficient leverage can arbitrarily corrupt the estimate. This motivates the development of robust alternatives.

M-estimators generalize maximum likelihood estimation by replacing the squared loss with robust alternatives:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \rho(r_i(\boldsymbol{\theta})) \tag{10.12}$$

Common choices for ρ include:

Name	$\boldsymbol{\rho(r)}$	**Influence Function** $\boldsymbol{\psi(r) = \rho'(r)}$
L2 (Least Squares)	r^2	$2r$
L1 (Least Absolute)	$\lvert r\rvert$	$\operatorname{sign}(r)$
Huber	$\begin{cases} r^2/2 & \lvert r\rvert \le k \\ k\lvert r\rvert - k^2/2 & \lvert r\rvert > k \end{cases}$	$\begin{cases} r & \lvert r\rvert \le k \\ k\operatorname{sign}(r) & \lvert r\rvert > k \end{cases}$
Tukey's Bisquare	$\begin{cases} k^2/6\left[1-(1-r^2/k^2)^3\right] & \lvert r\rvert \le k \\ k^2/6 & \lvert r\rvert > k \end{cases}$	$\begin{cases} r(1-r^2/k^2)^2 & \lvert r\rvert \le k \\ 0 & \lvert r\rvert > k \end{cases}$

Table 2: Common M-estimator loss functions and their derivatives

10.4 RANSAC: Random Sample Consensus

RANSAC takes a fundamentally different approach by explicitly modeling the presence of outliers through a consensus framework.

10.4.1 The Consensus Set

Definition 10.4.1.1 (Consensus Set): Given parameters $\boldsymbol{\theta}$ and threshold ε, the consensus set is:

$$\mathcal{C}(\boldsymbol{\theta}) = \{i : |r_i(\boldsymbol{\theta})| \leq \varepsilon\} \tag{10.13}$$

RANSAC maximizes the cardinality of the consensus set:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} |\mathcal{C}(\boldsymbol{\theta})| \tag{10.14}$$

10.4.2 The RANSAC Algorithm

Algorithm 10: RANSAC

procedure RANSAC(Data points $\mathcal{D}$, inlier threshold ε, confidence p)
  ▷ **Input:** Data points $\mathcal{D}$, inlier threshold ε, confidence p
  ▷ **Output:** Model parameters $\boldsymbol{\theta}^*$

  ▷ Initialize:
  $\mathcal{C}^* \leftarrow \emptyset$
  $k \leftarrow 0$

  while $k < N$ **do**
    ▷ (a) Randomly sample minimal set $\mathcal{S} \subset \mathcal{D}$ with $|\mathcal{S}| = m$
    $\mathcal{S} \leftarrow$ randomly sample m points from $\mathcal{D}$

    ▷ (b) Fit model:
    $\boldsymbol{\theta}_k \leftarrow \text{FitModel}(\mathcal{S})$

    ▷ (c) Evaluate consensus:
    $\mathcal{C}_k \leftarrow \{i : |r_i(\boldsymbol{\theta}_k)| \leq \varepsilon\}$

    ▷ (d) if $|\mathcal{C}_k| > |\mathcal{C}^*|$ then update best model
    if $|\mathcal{C}_k| > |\mathcal{C}^*|$ **then**
      $\mathcal{C}^* \leftarrow \mathcal{C}_k$
      $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_k$
    end

    $k \leftarrow k + 1$
  end

  ▷ Refine:
  $\boldsymbol{\theta}^* \leftarrow \text{LeastSquares}(\mathcal{C}^*)$

  return $\boldsymbol{\theta}^*$
end

10.4.3 Determining the Number of Iterations

The number of iterations N is determined probabilistically to ensure finding at least one outlier-free sample with confidence p:

Theorem 10.4.3.1 (RANSAC Iteration Count): Given inlier ratio w and desired confidence p, the required number of iterations is:

$$N = \frac{\log(1-p)}{\log(1-w^m)} \tag{10.15}$$

where m is the minimal sample size.

Proof: The probability of selecting an all-inlier minimal sample is w^m. The probability of failing to select such a sample in N attempts is $(1-w^m)^N$. Setting this equal to $1-p$ and solving for N yields the result.

Example 10.4.3.1 (RANSAC Iteration Example): For line fitting ($m=2$) with 50% inliers ($w=0.5$) and 99% confidence ($p=0.99$):

$$N = \frac{\log(0.01)}{\log(1-0.5^2)} = \frac{\log(0.01)}{\log(0.75)} \approx 17 \tag{10.16}$$

10.5 MSAC and MLESAC Variants

10.5.1 MSAC: M-estimator Sample Consensus

MSAC refines RANSAC by considering the magnitude of residuals within the inlier band:

$$L_{\text{MSAC}(\boldsymbol{\theta})} = \sum_{i=1}^{n} \min(|r_i(\boldsymbol{\theta})|, \varepsilon) \tag{10.17}$$

This formulation corresponds to a truncated L1 loss, providing better discrimination between competing models with similar consensus set sizes.

10.5.2 MLESAC: Maximum Likelihood Sample Consensus

MLESAC models the error distribution explicitly as a mixture:

$$p(r_i) = \gamma \mathcal{N}(0, \sigma^2) + (1-\gamma)\mathcal{U}(-v, v) \tag{10.18}$$

where γ is the inlier ratio, and maximizes the likelihood of the observed residuals.

11 Multiple Model Fitting

The extension from single-model to multi-model fitting introduces fundamental challenges that require sophisticated approaches. The core difficulty lies in the chicken-and-egg problem: clustering points requires knowledge of models, while fitting models requires knowledge of point clusters.

Definition 11.1 (Multi-Model Fitting Problem): Given a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ and a model family $\mathcal{M}$, find:

$$\Theta = \{\theta_1, \theta_2, ..., \theta_K\} \tag{11.1}$$

$$\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_K\} \tag{11.2}$$

where Θ represents the set of model parameters and $\mathcal{P}$ represents a partition of the data such that:

$$\bigcup_{k=1}^{K} \mathcal{P}_k = \mathcal{D} \setminus \mathcal{O} \tag{11.3}$$

$$\mathcal{P}_i \cap \mathcal{P}_j = \emptyset \quad \forall i \neq j \tag{11.4}$$

and each θ_k optimally fits the points in $\mathcal{P}_k$.

The fundamental challenge in multi-model fitting can be formalized as follows:

1. **Clustering Requirement**: To fit model θ_k, we need to identify the subset $\mathcal{P}_k \subset \mathcal{D}$ of points that belong to model k
2. **Model Requirement**: To cluster point $\boldsymbol{x}_i$ into partition $\mathcal{P}_k$, we need to know model θ_k to compute the distance $d(\boldsymbol{x}_i, \theta_k)$

This circular dependency necessitates iterative or simultaneous approaches that can break the cycle through:

- *Sequential strategies*: Fit models one at a time, removing inliers after each fit
- *Simultaneous strategies*: Jointly optimize over all models and partitions
- *Voting strategies*: Use parameter space voting to identify multiple models

11.1 Outlier Complexity in Multi-Model Settings

In multi-model fitting, the effective outlier ratio increases significantly. Consider fitting model θ_k in the presence of K models:

$$\varepsilon_{\text{effective}} = \varepsilon_{\text{true}} + \frac{K-1}{K}(1 - \varepsilon_{\text{true}}) \tag{11.5}$$

where $\varepsilon_{\text{true}}$ is the true outlier ratio and $\varepsilon_{\text{effective}}$ is the effective outlier ratio seen by each individual model.

Example: With 10 equally represented models and 10% true outliers:

$$\varepsilon_{\text{effective}} = 0.1 + \frac{9}{10}(1 - 0.1) = 0.1 + 0.9 \cdot 0.9 = 0.91 \tag{11.6}$$

This dramatic increase in effective outlier ratio severely impacts the performance of standard RANSAC, motivating specialized multi-model approaches.

11.2 Sequential Multi-Model Fitting

Sequential approaches address multi-model fitting by iteratively applying single-model fitting techniques, removing inliers after each successful fit. While conceptually simple, these methods can suffer from order dependency and accumulation of errors.

Algorithm 11: Sequential RANSAC

procedure SEQUENTIAL-RANSAC(dataset $\mathcal{D}$, minimum consensus $\tau_{\min}$, maximum models $K_{\max}$)
  ▷ **Input:** Dataset $\mathcal{D}$, minimum consensus $\tau_{\min}$, maximum models $K_{\max}$
  ▷ **Output:** Set of models Θ

  ▷ Initialize:
  $\mathcal{D}_{\text{current}} \leftarrow \mathcal{D}$
  $\Theta \leftarrow \emptyset$
  $k \leftarrow 0$

  ▷ While $k < K_{\max}$ and $|\mathcal{D}_{\text{current}}| \geq \tau_{\min}$:
  while $k < K_{\max} \wedge |\mathcal{D}_{\text{current}}| \geq \tau_{\min}$ **do**
    ▷ (a) Apply RANSAC to $\mathcal{D}_{\text{current}}$ to find model θ_k
    $\theta_k \leftarrow \text{RANSAC}(\mathcal{D}_{\text{current}})$

    ▷ (b) Compute consensus set
    $\mathcal{C}_k \leftarrow \{\boldsymbol{x}_i \in \mathcal{D}_{\text{current}} : d(\boldsymbol{x}_i, \theta_k) \leq \tau\}$

    ▷ (c) If $|\mathcal{C}_k| < \tau_{\min}$, terminate
    if $|\mathcal{C}_k| < \tau_{\min}$ **then**
      ▷ Terminate - insufficient consensus
      BREAK
    end

    ▷ (d) Refine θ_k using least squares on $\mathcal{C}_k$
    $\theta_k \leftarrow \text{LeastSquares}(\mathcal{C}_k)$

    ▷ (e) Add θ_k to Θ
    $\Theta \leftarrow \Theta \cup \{\theta_k\}$

    ▷ (f) Update: remove inliers from current dataset
    $\mathcal{D}_{\text{current}} \leftarrow \mathcal{D}_{\text{current}} \setminus \mathcal{C}_k$

    ▷ (g) Increment model counter
    $k \leftarrow k + 1$
  end

  return Θ
end

Theorem 11.2.1 (Sequential RANSAC Convergence): Under the assumption that models are well-separated and the minimum consensus threshold is appropriately chosen, Sequential RANSAC will find all models with probability:

$$P_{\text{success}} = \prod_{k=1}^{K} \left(1 - \left(1 - (1 - \varepsilon_k)^m\right)^{T_k}\right) \tag{11.7}$$

where ε_k is the outlier ratio when fitting model k and T_k is the number of RANSAC iterations for model k.

Limitations of Sequential RANSAC:

1. *Order Dependency*: The sequence in which models are found depends on their relative support and may not reflect the true underlying structure
2. *Error Accumulation*: Misclassified points in early iterations affect subsequent model fitting
3. *Threshold Sensitivity*: Performance heavily depends on the choice of inlier threshold and minimum consensus
4. *Suboptimal Solutions*: Greedy selection may lead to locally optimal but globally suboptimal solutions

Sequential RANSAC can be understood through the lens of preference matrices, which provide insight into simultaneous multi-model approaches.

Definition 11.2.1 (Preference Matrix): The preference matrix $\boldsymbol{P} \in \mathbb{R}^{N \times M}$ is defined as:

$$P_{ij} = \begin{cases} 1 \text{ if } \ d(\boldsymbol{x}_i, \theta_j) \leq \tau \\ 0 \text{ otherwise} \end{cases} \tag{11.8}$$

where N is the number of data points and M is the number of model hypotheses.

Alternatively, the preference matrix can store residuals:

$$P_{ij} = d(\boldsymbol{x}_i, \theta_j) \tag{11.9}$$

Conceptual Insights:

- Each row represents a data point's affinity to all model hypotheses
- Each column represents a model's support across all data points
- Sequential RANSAC selects the column with maximum support, then removes corresponding rows
- Simultaneous approaches can exploit the full matrix structure

11.3 Simultaneous Multi-Model Fitting

Simultaneous approaches to multi-model fitting attempt to identify all models jointly, avoiding the limitations of sequential methods. The Hough transform represents a classical voting-based approach that has been widely successful in computer vision applications.

Instead of sequentially selecting columns from the preference matrix, simultaneous approaches seek to:

1. *Cluster rows*: Group data points with similar preference patterns
2. *Regularize solutions*: Incorporate sparsity constraints to prefer simpler models
3. *Optimize jointly*: Minimize a global objective function over all models and assignments

Definition 11.3.1 (Joint Optimization Problem): The simultaneous multi-model fitting problem can be formulated as:

$$\min_{\Theta,\mathcal{P}} \quad \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in \mathcal{P}_k} \rho(d(\boldsymbol{x}_i, \theta_k)) + \lambda \left\|\Theta\right\|_0 \tag{11.10}$$

$$\text{subject to} \quad \mathcal{P}_i \cap \mathcal{P}_j = \emptyset \quad \forall i \neq j \tag{11.11}$$

$$\bigcup_{k=1}^{K} \mathcal{P}_k \subset \mathcal{D} \tag{11.12}$$

where $\rho(\cdot)$ is a robust loss function and λ controls model complexity.

11.3.1 The Hough Transform

The Hough transform provides an elegant solution to multi-model fitting by transforming the point clustering problem into a peak detection problem in parameter space.

The key insight of the Hough transform is the duality between point space and parameter space:

- Each point in data space corresponds to a curve in parameter space
- Each model in parameter space corresponds to a point in data space
- Points lying on the same model generate curves that intersect at the model's parameters

11.3.2 Line Detection via Hough Transform

For line detection, we parameterize lines using the normal form:

$$\rho = x\cos\theta + y\sin\theta \tag{11.13}$$

where ρ is the perpendicular distance from the origin to the line, and θ is the angle of the normal vector.

Algorithm 12: Hough Transform for Line Detection

procedure HOUGH-TRANSFORM(edge points $\{(x_i, y_i)\}_{i=1}^{N}$)
  ▷ **Input:** Edge points $\{(x_i, y_i)\}_{i=1}^{N}$
  ▷ **Output:** Line parameters $\{(\rho_k, \theta_k)\}$

  ▷ Initialize accumulator array $A[\rho, \theta]$ with appropriate discretization
  $A[\rho, \theta] \leftarrow 0$ for all ρ, θ

  ▷ For each edge point (x_i, y_i):
  for $i = 1, ..., N$ **do**
    ▷ (a) For $\theta = -\frac{\pi}{2}$ to $\frac{\pi}{2}$ with step $\Delta\theta$:
    for $\theta = -\frac{\pi}{2}, ..., \frac{\pi}{2}$, Step: $\Delta\theta$ **do**
      ▷ (i) Compute $\rho = x_i\cos\theta + y_i\sin\theta$
      $\rho \leftarrow x_i\cos\theta + y_i\sin\theta$

      ▷ (ii) Increment $A[\rho, \theta]$
      $A[\rho, \theta] \leftarrow A[\rho, \theta] + 1$
    end
  end

  ▷ Find local maxima in $A[\rho, \theta]$ above threshold
  Peaks $\leftarrow$ FindLocalMaxima$(A, \text{threshold})$

  ▷ Each maximum corresponds to a line with parameters (ρ, θ)
  for each peak (ρ_k, θ_k) in Peaks **do**
    ▷ Line detected with parameters (ρ_k, θ_k)
  end

  return "Peaks"
end

Advantages of Normal Parameterization:

1. *Bounded Parameter Space*: $\rho \in [0, \rho_{\max}]$ and $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
2. *No Singularities*: Unlike slope-intercept form, vertical lines are handled naturally
3. *Uniform Discretization*: Parameter space can be uniformly discretized

11.3.3 Extension to Circle Detection

Circle detection requires a three-dimensional parameter space (x_c, y_c, r) where (x_c, y_c) is the center and r is the radius. The circle equation is:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{11.14}$$

Algorithm 13: Hough Transform for Circle Detection

procedure HOUGH-TRANSFORM-CIRCLE$\left(\text{edge points } \{(x_i, y_i)\}_{i=1}^{N}, \text{known radius } r\right)$
  ▷ **Input:** Edge points $\{(x_i, y_i)\}_{i=1}^{N}$, known radius r
  ▷ **Output:** Circle centers $\left\{(x_{c,k}, y_{c,k})\right\}$

  ▷ Initialize 2D accumulator $A[x_c, y_c]$
  $A[x_c, y_c] \leftarrow 0$ for all x_c, y_c

  ▷ For each edge point (x_i, y_i):
  for $i = 1, ..., N$ **do**
    ▷ (a) For $\theta = 0$ to 2π with step $\Delta\theta$:
    for $\theta = 0, ..., 2\pi$ **do**
      ▷ (i) Compute center coordinates
      $x_c \leftarrow x_i + r\cos\theta$
      $y_c \leftarrow y_i + r\sin\theta$

      ▷ (ii) Increment $A[x_c, y_c]$
      $A[x_c, y_c] \leftarrow A[x_c, y_c] + 1$
    end
  end

  ▷ Find peaks in $A[x_c, y_c]$
  Peaks $\leftarrow$ FindLocalMaxima$(A, \text{threshold})$

  return "Peaks"
end

For unknown radius, a 3D accumulator is required, significantly increasing computational cost.

12 Appendix: Fundamental Concepts in Linear Algebra

12.1 Vector Spaces and Linear Combinations

Definition 12.1.1 (Span of Vectors): Given a set of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n\} \subset \mathbb{R}^m$, the *span* of these vectors is defined as:

$$\text{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n\} = \left\{ \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i : \lambda_i \in \mathbb{R} \right\} \tag{12.1}$$

The span represents the set of all possible linear combinations of the given vectors, forming a vector subspace of $\mathbb{R}^m$. This concept is fundamental to understanding how different sets of vectors can generate different subspaces.

12.2 Linear Independence and Basis

Definition 12.2.1 (Linear Independence): A set of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n\}$ is *linearly independent* if and only if:

$$\sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i = \boldsymbol{0} \quad \Rightarrow \quad \lambda_i = 0 \ \text{ for all } \ i = 1, 2, ..., n \tag{12.2}$$

This definition captures the fundamental property that no vector in the set can be expressed as a linear combination of the others. The importance of linear independence becomes clear when we consider the uniqueness of representations.

Theorem 12.2.1 (Uniqueness of Representation): Let $\{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n\} \subset \mathbb{R}^m$ be a linearly independent set of vectors. If $\boldsymbol{s} \in \text{span}\{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n\}$, then there exists a unique representation:

$$\boldsymbol{s} = \sum_{i=1}^{n} x_i \boldsymbol{e}_i \tag{12.3}$$

where the coefficients $x_i \in \mathbb{R}$ are uniquely determined.

Proof: Suppose $\boldsymbol{s}$ admits two different representations:

$$\begin{aligned} \boldsymbol{s} &= \sum_{i=1}^{n} x_i \boldsymbol{e}_i \\ \boldsymbol{s} &= \sum_{i=1}^{n} y_i \boldsymbol{e}_i \end{aligned} \tag{12.4}$$

Subtracting these equations:

$$\boldsymbol{0} = \sum_{i=1}^{n} (x_i - y_i) \boldsymbol{e}_i \tag{12.5}$$

By linear independence, $(x_i - y_i) = 0$ for all i, implying $x_i = y_i$ for all i. Therefore, the representation is unique.

12.3 Orthogonal Vectors

Definition 12.3.1 (Orthonormal Basis): A set of vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n\} \subset \mathbb{R}^n$ forms an *orthonormal basis* if:
1. $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle = \delta_{ij}$ (orthonormality condition)
2. $\text{span}\{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n\} = \mathbb{R}^n$ (spanning condition)

where δ_{ij} is the Kronecker delta. Kronecker delta is defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{12.6}$$

The power of orthonormal bases lies in their computational convenience. For any signal $\boldsymbol{s} \in \mathbb{R}^n$ and orthonormal basis $\boldsymbol{D} = [\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n]$, the coefficient computation is straightforward:

$$\boldsymbol{x} = \boldsymbol{D}^T \boldsymbol{s} \tag{12.7}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$ and $x_i = \langle \boldsymbol{e}_i, \boldsymbol{s} \rangle$.

12.4 The ℓ_0 Norm

The ℓ_0 norm (more precisely, ℓ_0 pseudo-norm) of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is defined as:

$$\|\boldsymbol{x}\|_0 := |\{i : x_i \neq 0\}| = \sum_{i=1}^{n} \mathbf{1}_{x_i \neq 0} \tag{12.8}$$

where $\mathbf{1}_{\{x_i \neq 0\}}$ is the indicator function that equals 1 if $x_i \neq 0$ and 0 otherwise.

The ℓ_0 norm can be understood as the limit of ℓ_p norms as $p \to 0^+$:

$$\|\boldsymbol{x}\|_0 = \lim_{p \to 0^+} \|\boldsymbol{x}\|_p^p = \lim_{p \to 0^+} \left(\sum_{i=1}^{n} |x_i|^p \right) \tag{12.9}$$

The ℓ_0 norm satisfies the following properties:
1. **Non-negativity**: $\|\boldsymbol{x}\|_0 \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$
2. **Zero property**: $\|\boldsymbol{x}\|_0 = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$
3. **Triangle inequality**: $\|\boldsymbol{x} + \boldsymbol{y}\|_0 \leq \|\boldsymbol{x}\|_0 + \|\boldsymbol{y}\|_0$
4. **Failure of homogeneity**: $\|\lambda \boldsymbol{x}\|_0 \neq |\lambda| \, \|\boldsymbol{x}\|_0$ for $\lambda \neq 0, \pm 1$

Let us now interpret ℓ_0-sparsity geometrically in $\mathbb{R}^3$.
- $\|\boldsymbol{\alpha}\|_0 = 0$: Only the origin $(0, 0, 0)$
- $\|\boldsymbol{\alpha}\|_0 = 1$: Points on coordinate axes, e.g., $(7, 0, 0)$, $(0, 3, 0)$
- $\|\boldsymbol{\alpha}\|_0 = 2$: Points lying in coordinate planes, e.g., $(5, 2, 0)$
- $\|\boldsymbol{\alpha}\|_0 = 3$: All other points in $\mathbb{R}^3$

12.5 Matrix Spark

The concept of matrix spark provides the theoretical foundation for understanding when sparse solutions are unique.

Definition 12.5.1 (Matrix Spark): For a matrix $\boldsymbol{D} \in \mathbb{R}^{m\times n}$, the spark of $\boldsymbol{D}$, denoted $\text{spark}(\boldsymbol{D})$, is defined as:

$$\text{spark}(\boldsymbol{D}) = \min\{|\mathcal{S}| : \mathcal{S} \subseteq \{1, 2, ..., n\}, \boldsymbol{D}_{\mathcal{S}} \text{ is linearly dependent}\} \tag{12.10}$$

where $|\mathcal{S}|$ denotes the cardinality of the set $\mathcal{S}$ and $\boldsymbol{D}_{\mathcal{S}}$ represents the submatrix of $\boldsymbol{D}$ formed by columns indexed by $\mathcal{S}$.

Equivalently, the spark can be defined in terms of the ℓ_0 norm as:

$$\text{spark}(\boldsymbol{D}) = \min\{\|\boldsymbol{x}\|_0 : \boldsymbol{D}\boldsymbol{x} = \boldsymbol{0}, \boldsymbol{x} \neq \boldsymbol{0}\} \tag{12.11}$$

The spark and rank of a matrix are related but distinct concepts:

Proposition 12.5.1 (Spark-Rank Relationship): For any matrix $\boldsymbol{D} \in \mathbb{R}^{m\times n}$ with $\text{rank}(\boldsymbol{D}) = r$:

$$1 \leq \text{spark}(\boldsymbol{D}) \leq r + 1 \tag{12.12}$$

Proof: The lower bound follows from the definition. For the upper bound, consider that any $r + 1$ columns must be linearly dependent in an r-dimensional space, hence $\text{spark}(\boldsymbol{D}) \leq r + 1$.

Lemma 12.5.1 (Spark and Homogeneous Solutions): If $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{0}$ has a solution $\boldsymbol{x} \neq \boldsymbol{0}$, then:

$$\text{spark}(\boldsymbol{D}) \leq \|\boldsymbol{x}\|_0 \tag{12.13}$$

Proof: If $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{0}$ with $\boldsymbol{x} \neq \boldsymbol{0}$, then $\sum_{i\in \text{ supp}(\boldsymbol{x})} x_i \boldsymbol{d}_i = \boldsymbol{0}$, showing that the columns $\{\boldsymbol{d}_i : i \in \text{supp}(\boldsymbol{x})\}$ are linearly dependent. By definition of spark, $\text{spark}(\boldsymbol{D}) \leq |\text{supp}(\boldsymbol{x})| = \|\boldsymbol{x}\|_0$.

13 Appendix: Optimization Theory for Non-Differentiable Functions

13.1 Lipschitz Continuity

Definition 13.1.1 (Lipschitz Continuous Gradient): A function $f : \mathbb{R}^n \to \mathbb{R}$ has an L-**Lipschitz continuous gradient** if there exists a constant $L > 0$ such that:

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{13.1}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. The smallest such constant L is called the Lipschitz constant of the gradient.

Definition 13.1.2 (Lipschitz Continuity of Gradient): A function f has L-Lipschitz continuous gradient if:

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \tag{13.2}$$

For the quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$:

Proposition 13.1.1 (Lipschitz Constant for Quadratic Functions): For the quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$, the Lipschitz constant can be derived as follows:

$$L = \left\|\boldsymbol{A}^T\boldsymbol{A}\right\|_2 = \lambda_{\max(\boldsymbol{A}^T\boldsymbol{A})} \tag{13.3}$$

where $\lambda_{\max}$ denotes the largest eigenvalue.

Proof: The gradient is $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$. Thus:

$$\begin{aligned} \|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 &= \left\|\boldsymbol{A}^T\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\right\|_2 \\ &\leq \left\|\boldsymbol{A}^T\boldsymbol{A}\right\|_2 \|\boldsymbol{x} - \boldsymbol{y}\|_2 \\ &= \lambda_{\max(\boldsymbol{A}^T\boldsymbol{A})}\|\boldsymbol{x} - \boldsymbol{y}\|_2 \end{aligned} \tag{13.4}$$

where we used the fact that the spectral norm equals the largest eigenvalue for symmetric positive semidefinite matrices.

13.2 Majorization-Minimization

For smooth convex functions, we can construct quadratic majorizers that facilitate optimization (subgradient methods).

Lemma 13.2.1 (Descent Lemma): Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex, differentiable function with Lipschitz continuous gradient. Then for any $\boldsymbol{x}_k \in \mathbb{R}^n$, there exists $L > 0$ such that:

$$f(\boldsymbol{x}) \leq Q_L(\boldsymbol{x}; \boldsymbol{x}_k) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \tag{13.5}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$.

Definition 13.2.1 (Majorization-Minimization Algorithm): Given a convex function f, the majorization-minimization approach generates a sequence $\{\boldsymbol{x}_k\}$ by:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} Q_L(\boldsymbol{x}; \boldsymbol{x}_k) \\ &= \arg\min_{\boldsymbol{x}} \left[f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{L}{2}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right] \end{aligned} \tag{13.6}$$

Minimizing the majorizer $Q_L(\boldsymbol{x}; \boldsymbol{x}_k)$ with respect to $\boldsymbol{x}$:

$$\begin{aligned} \nabla_{\boldsymbol{x}} Q_L(\boldsymbol{x}; \boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k) + L(\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{0} \\ \Rightarrow \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k) \end{aligned} \tag{13.7}$$

This recovers the standard gradient descent update with step size $\gamma = \frac{1}{L}$.

For a convex, differentiable function f with Lipschitz continuous gradient, the gradient descent algorithm converges to the global minimum.