The Little Book of Multilinear Algebra

Version 0.1.0

Duc-Tam Nguyen

2025-09-05

Table of contents

Part I. Orientation & Motivation **7**
Chapter 1. What is Multilinear? . . . 7
1.1 Linear vs. Multilinear: From Lines to Volumes . . . 7
1.2 Three Faces of a Tensor: Array, Map, and Element of a Product Space . . . 9
1.3 Why It Matters: Graphics, Physics, ML, Data Compression . . . 10
Chapter 2. Minimal Prerequisites . . . 14
2.1 Vector Spaces, Bases, Dimension . . . 14
2.2 Linear Maps, Matrices, Change of Basis . . . 16
Exercises . . . 18
2.3 Inner Products, Dual Spaces, Adjoint . . . 18
2.4 Bilinear Forms and Quadratic Forms . . . 20

Part II. Tensors **23**
Chapter 3. Tensors as Indexed Arrays . . . 23
3.1 Order (Arity), Shape, and Indices . . . 23
3.2 Covariant vs. Contravariant Indices . . . 25
3.3 Change of Basis Rules in Coordinates . . . 27
3.4 Einstein Summation and Index Hygiene . . . 29
Chapter 4. Tensors as Multilinear Maps . . . 32
4.1 Multilinearity and Currying . . . 32
4.2 Evaluation with Vectors and Covectors . . . 33
4.3 From Maps to Arrays (and Back) via a Basis . . . 35
4.4 Universal Examples: Bilinear Forms, Trilinear Mixing . . . 37
Chapter 5. Tensors as Elements of Tensor Products . . . 39
5.1 Constructing $V \otimes W$: Intuition and Goals . . . 39
5.2 Simple vs. General Tensors . . . 41
5.3 Dimension and Bases of Tensor Products . . . 43
5.4 Three Viewpoints Reconciled . . . 45
Exercises . . . 47

Part III. Core Operations on Tensors **48**
Chapter 6. Building Blocks . . . 48
6.1 Tensor (Outer) Product . . . 48
6.2 Contraction: Summing Paired Indices . . . 50
6.3 Permutations of Modes: Transpose, Unfold, Matricize . . . 51

6.4 Kronecker Product vs. Tensor Product . . . 53
Chapter 7. Symmetry and (Anti)Symmetry . . . 55
7.1 Symmetrization Operators . . . 55
7.2 Alternation (Antisymmetrization) . . . 57
7.3 Decompositions by Symmetry Type . . . 59

Part IV. Exterior Algebra **63**
Chapter 8. Wedge Products and k-Vectors . . . 63
8.1 The Wedge Product and Geometric Meaning . . . 63
8.2 Areas, Volumes, Orientation, Determinant . . . 65
8.3 Basis, Dimension, Grassmann Algebra . . . 67
8.4 Differential Forms (Gentle Preview) . . . 68
Chapter 9. Hodge Dual and Metrics (Optional but Useful) . . . 70
9.1 Inner Products on Exterior Powers . . . 70
9.2 Hodge Star, Pseudo-Vectors, Cross Products . . . 72
9.3 Volume Forms and Integration Glimpses . . . 74

Part V. Symmetric Tensors and Polynomial View **76**
Chapter 10. Symmetric Powers and Homogeneous Polynomials . . . 76
10.1 Symmetric Tensors as Polynomial Coefficients . . . 76
10.2 Polarization Identities . . . 79
10.3 Moments and Cumulants . . . 81
10.4 Low-Rank Symmetric Decompositions . . . 83

Part VI. Linear Maps Between Tensor Spaces **86**
Chapter 11. Reshaping, Vectorization, and Commutation . . . 86
11.1 Mode-n Unfolding and Matricization . . . 86
11.2 Vec Operator and Kronecker Identities . . . 88
11.3 Linear Operators Acting on Tensors . . . 90
Chapter 12. Metrics, Forms, and Raising/Lowering Indices . . . 92
12.1 Using Inner Products to Move Indices . . . 92

Part VII. Tensor Ranks and Decompositions **98**
Chapter 13. Ranks for Tensors . . . 98
13.1 Matrix Rank vs. Tensor Rank . . . 98
13.2 Multilinear (Tucker) Rank and Mode Ranks . . . 100
Chapter 14. Cannonical Decompositions . . . 104
14.1 CP (CANDECOMP/PARAFAC) . . . 104
14.2 Tucker and HOSVD . . . 106
14.3 Tensor Trains (TT) and Hierarchical Formats . . . 108
14.4 Connections to SVD and PCA . . . 109

Part VIII. Computation and Numerical Practice **112**
Chapter 15. Working with Tensors in Code . . . 112
15.1 Efficient Indexing and Memory Layout . . . 112
Exercises . . . 113
15.2 BLAS, Einsum, Performance Patterns . . . 113
15.3 Stability, Conditioning, Scaling Tricks . . . 116
Chapter 16. Automatic Differentiation and Gradients . . . 117
16.1 Jacobians/Hessians as Tensors . . . 117
Exercises . . . 119
16.2 Backprop as Structured Contractions . . . 119
16.3 Practical Tips for PyTorch/JAX/NumPy . . . 122

Part IX. Applications you can touch **125**
Chapter 17. Data Science and Signal Processing . . . 125
17.1 Multilinear Regression . . . 125
17.2 Spatiotemporal Data and Video Tensors . . . 127
17.3 Blind Source Separation . . . 129
Chapter 18. Machine Learning and Deep Models . . . 130
18.1 Convolutions as Multilinear Maps . . . 130
18.2 Low-Rank Tensor Compression of Nets . . . 132
Chapter 19. Physics, Graphics, and Beyond . . . 136
19.1 Stress/Strain Tensors . . . 136
19.2 Inertia Tensors and Principal Axes . . . 138
19.3 3D Graphics: Transforms and Shading . . . 140
19.4 Quantum States and Operators . . . 143

Part IX. Glimpses Beyond This Book **145**
Chapter 20. Manifolds and Tensor Fields (Preview) . . . 145
20.1 Tangent and Cotangent Bundles . . . 145
20.2 Tensor Fields and Coordinate Changes . . . 146
20.3 Covariant Derivatives and Curvature . . . 148
Chapter 21. Representation Theory and Invariants (Preview) . . . 150
21.1 Group Actions on Tensor Spaces . . . 150
21.2 Invariant Tensors and Symmetry . . . 152
21.3 Why Invariants Matter in Algorithms . . . 154

End Matter **156**
A. Symbols and Notation Cheatsheet . . . 156
Vector Spaces and Duals . . . 156
Tensors . . . 156
Indices and Summation . . . 156
Linear Maps and Operators . . . 157
Tensor Operations . . . 157

Special Objects . . . 157
Appendix B. Proof Sketches of Core Theorems . . . 157
Appendix B.1 Universal Property of the Tensor Product (Proof Sketch) . . . 157
Appendix B.2 Dimension Formula for Tensor Products (Proof Sketch) . . . 159
Appendix B.3 Decomposition of Tensors into Symmetric and Antisymmetric Parts (Proof Sketch) . . . 161
Appendix D. Identities & "Cookbook" . . . 167
1. Kronecker Product & Vec Identities . . . 167
2. Trace Tricks . . . 168
3. Tensor Contractions . . . 168
4. Determinant & Volumes . . . 169
5. Differential & Gradient Identities . . . 169
6. Useful Einsum Patterns . . . 169
7. Symmetrization / Antisymmetrization . . . 170
Appendix E.1 Mini-Project: Implement CP Decomposition for Small 3-Way Tensors 170
1. Background . . . 170
2. Implementation Plan . . . 171
3. Coding Hints . . . 171
4. Extensions (Optional) . . . 172
5. Deliverables . . . 172
Appendix E.2 Mini-Project: Tucker Decomposition for Video Compression . . . 172
1. Background . . . 172
2. Implementation Plan . . . 173
3. Coding Hints . . . 174
4. Extensions (Optional) . . . 174
5. Deliverables . . . 174
Appendix E.3 Mini-Project: Strain Tensor for a Rotating Plate . . . 174
1. Background . . . 175
2. Setup: Rotating Plate . . . 175
3. Compute Strain Tensor . . . 175
4. Interpretation . . . 176
5. Coding Hints . . . 176
6. Extensions (Optional) . . . 176
7. Deliverables . . . 176
Appendix E.4 Mini-Project: Parallel Transport of a Vector on a Sphere . . . 177
1. Background . . . 177
2. Setup: The Sphere S^2 . . . 177
3. Parallel Transport along Geodesics . . . 177
4. Coding Hints . . . 178
5. Extensions (Optional) . . . 178
6. Deliverables . . . 178
Appendix E.5 Mini-Project: PCA vs. Tucker Decomposition on Real Data . . . 178
1. Background . . . 179

2. Dataset Options 179
3. Methodology 179
4. Coding Hints 180
5. Extensions (Optional) 180
6. Deliverables 180

Part I. Orientation & Motivation

Chapter 1. What is Multilinear?

1.1 Linear vs. Multilinear: From Lines to Volumes

When you first meet *linear algebra*, the word "linear" carries a specific flavor: we study maps that preserve straightness and scaling. A linear map $f : V \to W$ satisfies

$$f(av_1 + bv_2) = af(v_1) + bf(v_2),$$

for scalars a, b and vectors $v_1, v_2 \in V$. The picture is of functions that take lines to lines, planes to planes, preserving the essential structure of addition and scaling.

From Linear to Multilinear

A multilinear map involves *several vector inputs at once*. For instance, a bilinear map takes two vectors:

$$B : V \times W \to \mathbb{R}, \quad B(av_1 + bv_2, w) = aB(v_1, w) + bB(v_2, w),$$

and is linear in each argument separately. More generally, a k-linear map eats k vectors-each input behaves linearly while the others are held fixed.

Geometry of the Shift

- Linear (1-input): Think of scaling a line, stretching it along one axis.
- Bilinear (2-input): Now imagine two directions at once. The determinant of a 2×2 matrix is bilinear: it measures the signed *area- spanned by two vectors.
- Trilinear (3-input): With three inputs, multilinearity measures a *volume*. The scalar triple product $u \cdot (v \times w)$ is trilinear, giving the volume of the parallelepiped formed by u, v, w.
- Higher multilinearity: Beyond three inputs, we can measure 4D hyper-volumes and higher-dimensional "content."

Intuition: From 1D to Higher Dimensions

- 1D (linear): One direction $\rightarrow$ length.
- 2D (bilinear): Two directions $\rightarrow$ area.
- 3D (trilinear): Three directions $\rightarrow$ volume.
- kD (multilinear): k directions $\rightarrow$ hyper-volume.

Each new input adds a new dimension of measurement. In this sense, multilinear algebra is the natural extension of linear algebra: instead of studying transformations of single vectors, we study functions that combine several vectors in structured ways.

Everyday Examples

- Dot product: Bilinear form producing a scalar from two vectors.
- Matrix multiplication: Bilinear in its row and column inputs.
- Determinant: Multilinear in its columns (or rows), encoding volume.
- Cross product: Bilinear but antisymmetric, giving a vector orthogonal to two inputs.
- Neural networks: Convolutions and tensor contractions are deeply multilinear operations, reshaped for computation.

Linear algebra captures what happens when you act on one direction at a time. Multilinear algebra generalizes this, letting us measure, transform, and compute with several directions simultaneously. It is the leap from lines to volumes, from single-vector transformations to the rich geometry of many interacting vectors.

Exercises

1. Linearity Check: Let $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ be defined by $f(x, y) = (2x, 3y)$. Show that f is linear.
2. Bilinear Dot Product: Verify that the dot product $\langle u, v\rangle = u_1v_1 + u_2v_2 + u_3v_3$ is bilinear by checking linearity in each argument separately.
3. Area via Determinant: For vectors $u = (1, 0)$ and $v = (1, 2)$ in $\mathbb{R}^2$, compute the determinant of the 2×2 matrix with columns u and v. Interpret the result geometrically.
4. Volume via Scalar Triple Product: Compute the scalar triple product $u \cdot (v \times w)$ for $u = (1, 0, 0)$, $v = (0, 1, 0)$, and $w = (0, 0, 1)$. Explain why the result makes sense in terms of volume.
5. Generalization Thought Experiment: Imagine you have four independent vectors in $\mathbb{R}^4$. What kind of geometric quantity does a multilinear map on these four inputs measure? (Hint: think of the analogy with length, area, and volume.)

1.2 Three Faces of a Tensor: Array, Map, and Element of a Product Space

A tensor can feel slippery at first, because it wears different "faces" depending on how you meet it. Each face is valid and useful. Together they form the three standard viewpoints:

1. Array View: Numbers in a Box

At the simplest level, a tensor looks like a multidimensional array of numbers.

- A vector is a 1-dimensional array: $[v_i]$.
- A matrix is a 2-dimensional array: $[a_{ij}]$.
- A general tensor of order k is like a k-dimensional grid of numbers: $[t_{i_1 i_2 \ldots i_k}]$.

This viewpoint is intuitive for computation, storage, and indexing. For example, in machine learning an image is a 3rd-order tensor: height $\times$ width $\times$ color channels.

2. Map View: Multilinear Functions

Another way to see tensors is as multilinear maps.

- A bilinear form takes two vectors and outputs a number, e.g., the dot product.
- A trilinear form takes three vectors and outputs a number, e.g., the scalar triple product.
- In general, a tensor T of type $(0, k)$ is a map

$$T : V \times V \times \cdots \times V \to \mathbb{R},$$

  linear in each slot separately.

This viewpoint highlights *behavior*: how the tensor interacts with vectors. It is central in geometry and physics, where tensors encode measurable relationships.

3. Product Space View: Elements of $V \otimes W \otimes \cdots$

The third viewpoint places tensors as elements of tensor product spaces.

- Given vector spaces V and W, their tensor product $V \otimes W$ is a new space built to capture bilinear maps.
- A simple (decomposable) tensor looks like $v \otimes w$.
- General tensors are linear combinations of these simple pieces.

This is the most abstract but also the most powerful perspective. It makes precise statements about dimension, bases, and transformation laws. It also unifies the array and map viewpoints: the array entries are just the coordinates of a tensor element with respect to a basis, and the map behavior is encoded in how the element acts under evaluation.

Reconciling the Views

- Array: concrete for computation.
- Map: functional and geometric meaning.
- Product space: rigorous foundation.

A beginner should be comfortable switching between them: "the same object, three different languages."

Exercises

1. Array Identification: Write down a 3rd-order tensor T with entries t_{ijk} where $i, j, k \in \{1, 2\}$. How many numbers are needed to specify T?
2. Dot Product as a Tensor: Show that the dot product on $\mathbb{R}^3$ can be viewed both as a bilinear map and as an array (matrix).
3. Simple Tensor Construction: Given $u = (1, 2) \in \mathbb{R}^2$ and $v = (3, 4) \in \mathbb{R}^2$, form the tensor $u \otimes v$. Write its coordinates as a 2×2 array.
4. Switching Perspectives: Consider the determinant of a 2×2 matrix: $\det([a_{ij}]) = a_{11}a_{22} - a_{12}a_{21}$. Explain how this determinant can be seen as a bilinear map of the two column vectors of the matrix.
5. Thought Experiment: Suppose you store an RGB image of size 100×200.
    - In the array view, what is the order (number of indices) of the tensor representing the image?
    - In the product space view, which vector spaces might this tensor live in?

1.3 Why It Matters: Graphics, Physics, ML, Data Compression

Tensors may sound abstract, but they appear everywhere in modern science and technology. Understanding *why- they matter helps motivate the study of multilinear algebra.

In Computer Graphics

- Transformations: 3D graphics use matrices (2nd-order tensors) to rotate, scale, and project objects.
- Lighting Models: Many shading equations combine vectors of light, normal directions, and material properties in bilinear or trilinear ways.
- Animation: Deformations of 3D models often rely on multilinear blending of control parameters.

Tensors let us express geometry and transformations in compact formulas that computers can process efficiently.

In Physics and Engineering

- Stress and Strain: The stress tensor (2nd-order) relates internal forces to orientations in a material. The elasticity tensor (4th-order) relates stress and strain in solids.
- Electromagnetism: The electromagnetic field tensor encodes electric and magnetic fields in a relativistic framework.
- Inertia Tensor: Describes how an object resists rotational acceleration depending on its mass distribution.

Tensors naturally capture "laws that hold in every direction," which is why they dominate in mechanics and field theories.

In Machine Learning

- Neural Networks: Input data (images, videos, audio) are tensors of order 3, 4, or higher.
- Convolutions: The convolution kernel is a small tensor sliding across a larger tensor, producing a new one.
- Attention Mechanisms: Core operations in transformers are tensor contractions that combine multiple input arrays.

Understanding tensor decompositions helps compress models, speed up computation, and reveal hidden structures in data.

In Data Compression and Signal Processing

- PCA and SVD: Classic matrix decompositions are 2D tensor methods.
- Tensor Decompositions (CP, Tucker, TT): These extend compression ideas to multidimensional data, useful for video, hyperspectral imaging, and big-data analysis.
- Multiway Data Analysis: Tensors allow us to uncover patterns across several "modes" simultaneously-like user $\times$ time $\times$ product in recommendation systems.

The Big Picture

Linear algebra lets us describe single-direction transformations (lines). Multilinear algebra extends this to multiple interacting directions (areas, volumes, hyper-volumes). These structures appear whenever we handle multi-dimensional data or physical laws.

Learning to *think tensorially- is the key to navigating modern applied mathematics, science, and AI.

Exercises

1. Graphics Example: A 3D rotation matrix is a 2nd-order tensor. Explain why it is linear in its input vector but not multilinear.
2. Physics Example: The stress tensor σ maps a direction vector n to a force vector σn. Why is this operation linear in n?
3. Machine Learning Example: An RGB image of size 32×32 has 3 color channels.
    - What is the order of the tensor representing this image?
    - How many entries does it have in total?
4. Data Compression Example: PCA reduces a data matrix (2nd-order tensor) to a low-rank approximation. Suggest what a "low-rank" tensor decomposition might achieve for video data (3rd-order tensor: frame $\times$ width $\times$ height).
5. Thought Experiment: Suppose you could only use vectors and matrices, not higher-order tensors. Which of the following applications would be impossible or very awkward:
    - Representing the interaction of three forces at once.
    - Compressing a color video.
    - Encoding the stress-strain relationship in 3D materials. ### 1.4 A First Walk-Through: Color Images and 3-Way Arrays

Let's make tensors concrete with an everyday example: digital images.

From Grayscale to Color

- A grayscale image of size 100×200 can be seen as a matrix (2nd-order tensor). Each entry stores the brightness of a pixel.

- A color image has three channels: red, green, and blue (RGB). Now every pixel carries three values. This naturally forms a 3rd-order tensor:

$$I \in \mathbb{R}^{100 \times 200 \times 3}.$$

  The three indices correspond to row, column, color channel.

The Three Index Roles

1. Row (height): vertical position in the image.
2. Column (width): horizontal position in the image.
3. Channel: one of the RGB color intensities.

Together, (i, j, k) points to a single number: the intensity of color k at pixel (i, j).

Operations as Tensor Manipulations

- Flattening: We can reshape the 3D tensor into a 2D matrix, useful for feeding into algorithms that expect vectors or matrices.
- Contraction (summing over an index): If we sum over the color channel, we turn an RGB image into a grayscale image.
- Outer products: A colored checkerboard pattern can be constructed by taking tensor products of row and column vectors, then adding a color channel vector.

Why This Example Matters

- It shows how natural data can have more than two indices.
- It illustrates why matrices (2D tensors) are not enough for modern problems.
- It connects tensor operations with practical tasks: filtering, compression, feature extraction.

In fact, video data adds one more index: time. A video is a 4th-order tensor: frame $\times$ height $\times$ width $\times$ channel.

Exercises

1. Counting Entries: How many numbers are required to store a color image of size 64×64?
2. Slicing: For a color image tensor $I \in \mathbb{R}^{100\times200\times3}$, what is the shape of:
   - a single row across all columns and channels?
   - a single color channel across all pixels?
3. Flattening Practice: A $32 \times 32 \times 3$ image is flattened into a vector. What is the length of this vector?
4. Grayscale Conversion: Define a grayscale image $G(i, j)$ from a color image tensor $I(i, j, k)$ by averaging across channels:

$$G(i, j) = \frac{1}{3}\sum_{k=1}^{3} I(i, j, k).$$

   Why is this operation an example of contraction?
5. Video as Tensor: Suppose you have a 10-second video at 30 frames per second, each frame 128×128 with 3 color channels.
   - What is the order of the video tensor?
   - How many entries does it contain in total?

Chapter 2. Minimal Prerequisites

2.1 Vector Spaces, Bases, Dimension

Before diving deeper into multilinear algebra, we need a short refresher on the basic building blocks of linear algebra: vector spaces.

What is a Vector Space?

A vector space is a collection of objects (called *vectors*) that can be:

1. Added: $u + v$ is again a vector.
2. Scaled: av (where a is a scalar) is again a vector.

The rules of addition and scaling follow natural laws: associativity, commutativity, distributivity, and the existence of a zero vector.

Examples:

- $\mathbb{R}^n$: all n-tuples of real numbers.
- Polynomials of degree d.
- Continuous functions on an interval.

Bases and Coordinates

A basis of a vector space is a set of vectors that:

1. Are linearly independent (no one is a linear combination of the others).
2. Span the entire space (every vector can be expressed as a linear combination of them).

For $\mathbb{R}^3$, the standard basis is:

$$e_1 = (1, 0, 0), \quad e_2 = (0, 1, 0), \quad e_3 = (0, 0, 1).$$

Every vector $v \in \mathbb{R}^3$ can be uniquely written as:

$$v = xe_1 + ye_2 + ze_3,$$

with coordinates (x, y, z).

Dimension

The dimension of a vector space is the number of vectors in any basis.

- $\mathbb{R}^n$ has dimension n.
- Polynomials of degree d form a vector space of dimension $d + 1$.
- A trivial space $\{0\}$ has dimension 0.

Dimension gives the "number of independent directions" in the space.

Why This Matters for Multilinear Algebra

- Tensors live in spaces built from vector spaces (tensor products).
- Understanding bases and dimensions is crucial for counting entries of tensors.
- Coordinates provide the link between abstract definitions and concrete arrays.

Exercises

1. Checking Vector Spaces: Decide whether each of the following is a vector space over $\mathbb{R}$:
   - (a) All 2×2 real matrices.
   - (b) All positive real numbers.
   - (c) All polynomials with real coefficients.
2. Basis in $\mathbb{R}^2$: Show that $(1, 1)$ and $(1, -1)$ form a basis for $\mathbb{R}^2$. Express the vector $(3, 2)$ in this basis.
3. Counting Dimension: What is the dimension of the space of all real polynomials of degree 4? Suggest a natural basis.
4. Uniqueness of Representation: In $\mathbb{R}^3$, write $(2, 3, 5)$ as a combination of e_1, e_2, e_3. Why is this representation unique?
5. Application to Tensors: If $V = \mathbb{R}^2$ and $W = \mathbb{R}^3$, what is the dimension of the product space $V \otimes W$?

2.2 Linear Maps, Matrices, Change of Basis

Having reviewed vector spaces, we now turn to linear maps-the main actors in linear algebra.

Linear Maps

A linear map $T : V \to W$ between vector spaces satisfies:

$$T(av + bw) = aT(v) + bT(w),$$

for all scalars a, b and vectors $v, w \in V$.

Examples:

- Scaling: $T(x) = 3x$.
- Rotation: $T : \mathbb{R}^2 \to \mathbb{R}^2$ rotates vectors by 90°.
- Derivative: $D : P_3 \to P_2$ (maps a polynomial of degree 3 to its derivative).

Linear maps preserve the structure of vector spaces.

Matrices as Representations

Given bases of V and W, a linear map $T : V \to W$ can be represented by a matrix.

- Columns of the matrix are just the images of the basis vectors of V.
- If $T(e_i) = \sum_j a_{ji} f_j$, then the matrix entries are a_{ji}.

Thus, matrices are coordinate-based representations of abstract linear maps.

Composition and Matrix Multiplication

- Composing linear maps corresponds to multiplying their matrices.
- The identity map corresponds to the identity matrix.
- Inverse maps correspond to inverse matrices (when they exist).

This makes linear maps concrete and computable.

Change of Basis

Suppose we change basis in a vector space V:

- Old basis: $\{e_1, \dots, e_n\}$.
- New basis: $\{e'_1, \dots, e'_n\}$.

The change-of-basis matrix P expresses each new basis vector as a combination of the old ones.

For a linear operator $T : V \to V$:

$$[T]_{new} = P^{-1}[T]_{old}P.$$

This formula is fundamental for tensors, since tensors must transform consistently under basis changes.

Why This Matters for Multilinear Algebra

- Linear maps are 2nd-order tensors.
- Understanding how matrices change under new coordinates sets the stage for how higher-order tensors transform.
- The concept of basis change ensures that tensors encode *intrinsic- information, not just numbers in an array.

Exercises

1. Linear or Not? Decide whether each map is linear:

   - (a) $T(x, y) = (2x, 3y)$.
   - (b) $S(x, y) = (x^2, y)$.
   - (c) $R(x, y) = (y, x)$.

2. Matrix Representation: Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be defined by $T(x, y) = (x + 2y, 3x + y)$. Find the matrix of T with respect to the standard basis.

3. Composition Practice: If $A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, compute AB. Interpret the action of AB as a linear map.

4. Change of Basis: In $\mathbb{R}^2$, let the old basis be $e_1 = (1, 0), e_2 = (0, 1)$. The new basis is $e_1' = (1, 1), e_2' = (1, -1)$.

   - Find the change-of-basis matrix P.
   - Verify that P^{-1} transforms coordinates back to the old basis.

5. Tensor Connection: Explain why a linear map $T : V \to W$ can be viewed as an element of $V^- \otimes W$. (Hint: it eats a vector and produces another vector, which can be encoded by pairing with covectors.)

2.3 Inner Products, Dual Spaces, Adjoint

Linear maps and vector spaces give the structure. To measure *angles, lengths, and projections*, we need inner products. To generalize "coordinates" beyond a chosen basis, we need dual spaces. These two ideas connect directly in multilinear algebra.

Inner Products

An inner product on a real vector space V is a function

$$\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$$

satisfying:

1. Linearity in each slot: $\langle av + bw, u \rangle = a\langle v, u \rangle + b\langle w, u \rangle$.
2. Symmetry: $\langle v, w \rangle = \langle w, v \rangle$.
3. Positive definiteness: $\langle v, v \rangle \geq 0$, with equality only when $v = 0$.

This structure gives:

- Length: $\|v\| = \sqrt{\langle v, v \rangle}$.
- Angle: $\cos\theta = \frac{\langle v,w \rangle}{\|v\|\|w\|}$.
- Orthogonality: $\langle v, w \rangle = 0$.

Example: the dot product in $\mathbb{R}^n$.

Dual Spaces

The dual space V^* is the set of all linear functionals $f : V \to \mathbb{R}$.

- Elements of V^* are called covectors.
- If $V = \mathbb{R}^n$, then $V^- \cong \mathbb{R}^n$, but conceptually they are different:
  - Vectors: "arrows" in space.
  - Covectors: "measuring devices" that output numbers when fed a vector.

The dual basis:

- If $\{e_1, \dots, e_n\}$ is a basis of V, then there is a unique dual basis $\{e^1, \dots, e^n\}$ in V^* with

$$e^i(e_j) = \delta^i_j.$$

This duality underpins how tensor indices "live up or down" (contravariant vs. covariant).

Adjoint of a Linear Map

Given a linear map $T : V \to V$ on an inner product space, the adjoint T^* is defined by:

$$\langle Tv, w \rangle = \langle v, T^* w \rangle \quad \forall v, w \in V.$$

- If T is represented by a matrix A in an orthonormal basis, then T^* corresponds to the transpose $A^\top$.
- Adjoint maps generalize the idea of "transpose" to arbitrary inner product spaces.

Why This Matters for Multilinear Algebra

- Inner products allow us to raise or lower indices (switch between vectors and covectors).
- Dual spaces are essential for defining general tensors (mixing vectors and covectors).
- Adjoint operators appear everywhere in applications: projections, least squares, and symmetry in physical laws.

Exercises

1. Inner Product Verification: Show that $\langle (x_1, y_1), (x_2, y_2) \rangle = 2x_1x_2 + y_1y_2$ defines an inner product on $\mathbb{R}^2$.
2. Length and Angle: For $u = (1, 2, 2)$ and $v = (2, 0, 1)$ in $\mathbb{R}^3$, compute:
   - $\|u\|$, $\|v\|$.
   - The cosine of the angle between them.
3. Dual Basis: Let $V = \mathbb{R}^2$ with basis $e_1 = (1, 0), e_2 = (0, 1)$.
   - Write the dual basis e^1, e^2.
   - Compute $e^1(3, 4)$ and $e^2(3, 4)$.
4. Adjoint Map: Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ with matrix

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}.$$

   Find the matrix of T^* under the standard dot product.
5. Tensor Connection: Explain why an element of $V^- \otimes V$ can be interpreted as a matrix, and why adjointness naturally appears when working with inner products.

2.4 Bilinear Forms and Quadratic Forms

Now that we have inner products and dual spaces, we can introduce two important types of multilinear maps that already appear in basic linear algebra: bilinear forms and quadratic forms.

Bilinear Forms

A bilinear form on a vector space V is a function

$$B : V \times V \to \mathbb{R}$$

that is linear in each argument separately:

- $B(av_1 + bv_2, w) = aB(v_1, w) + bB(v_2, w)$,
- $B(v, aw_1 + bw_2) = aB(v, w_1) + bB(v, w_2)$.

Examples:

- Dot product: $\langle v, w \rangle$ is symmetric and bilinear.
- Matrix form: Any matrix A defines a bilinear form by

$$B(v, w) = v^\top A w.$$

Properties:

- Symmetric: if $B(v, w) = B(w, v)$.
- Skew-symmetric: if $B(v, w) = -B(w, v)$.

Quadratic Forms

A quadratic form is a special case obtained by feeding the *same- vector into both slots of a bilinear form:

$$Q(v) = B(v, v).$$

In coordinates, with a matrix A:

$$Q(v) = v^\top A v.$$

Examples:

- In $\mathbb{R}^2$, $Q(x, y) = x^2 + y^2$ corresponds to the identity matrix.
- In optimization, quadratic forms represent energy functions, error functions, or cost functions.

Geometric Meaning

- Quadratic forms define conic sections (ellipses, hyperbolas, parabolas) in 2D and quadric surfaces in higher dimensions.
- They also measure "curvature" locally in multivariable functions (via the Hessian matrix).

Why This Matters for Multilinear Algebra

- Bilinear forms are 2nd-order tensors: one covector for each input.
- Quadratic forms connect multilinearity with geometry (shapes, volumes, energy).
- The distinction between symmetric and skew-symmetric bilinear forms leads to exterior algebra and inner product structures later.

Exercises

1. Matrix Bilinear Form: Let

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}.$$

   Define $B(v, w) = v^\top A w$.

   - Compute $B((1, 0), (0, 1))$.
   - Compute $B((1, 2), (3, 4))$.

2. Symmetry Check: For the above B, show that $B(v, w) = B(w, v)$.

3. Quadratic Form: Compute $Q(x, y) = [x\ y]A[x\ y]^\top$ for the same matrix A. Write the explicit formula.

4. Geometric Interpretation: Consider $Q(x, y) = 4x^2 + y^2$. Sketch (or describe) the curve $Q(x, y) = 1$. What kind of conic section is it?

5. Tensor Connection: Explain why a bilinear form $B : V \times V \to \mathbb{R}$ can be seen as an element of $V^- \otimes V^*$. What does this mean in terms of indices (covariant slots)?

Part II. Tensors

Chapter 3. Tensors as Indexed Arrays

3.1 Order (Arity), Shape, and Indices

We now begin exploring tensors directly, starting with the array viewpoint. This perspective treats tensors as multi-dimensional generalizations of matrices.

Order (Arity) of a Tensor

The order (or arity) of a tensor is the number of indices needed to locate one of its entries.

- 0th-order: A scalar, e.g. 5.
- 1st-order: A vector v_i, with one index.
- 2nd-order: A matrix A_{ij}, with two indices.
- 3rd-order: A block of numbers T_{ijk}.
- kth-order: An array with k indices.

Thus, the order tells us how many "directions" or "modes" the tensor has.

Shape (Dimensions of Each Mode)

Each index ranges over some set of values, defining the shape of the tensor.

- A vector in $\mathbb{R}^n$ has shape (n).
- A matrix of size $m \times n$ has shape (m, n).
- A color image of size 100×200 with 3 channels has shape $(100, 200, 3)$.

The shape is just the list of sizes of each mode.

Indices and Notation

Indices label positions in the tensor:

$$T_{i_1 i_2 \ldots i_k}.$$

Example: A 3rd-order tensor T_{ijk} with shape $(2, 3, 4)$ has:

- $i \in \{1, 2\}$,
- $j \in \{1, 2, 3\}$,
- $k \in \{1, 2, 3, 4\}$.

So it contains $2 \times 3 \times 4 = 24$ entries.

Visual Intuition

- Scalars are points.
- Vectors are arrows (1D arrays).
- Matrices are grids (2D arrays).
- Higher-order tensors are cubes, hypercubes, or higher-dimensional arrays, which we can't fully draw but can still manipulate symbolically.

Why This Matters

- The array view is the most concrete: you can store tensors in memory, index them, and manipulate them in code.
- It provides the language of shapes that data science, ML, and physics use constantly.
- Later, we will see that these indices correspond to *slots- in multilinear maps and tensor product spaces.

Exercises

1. Counting Entries: How many entries does a tensor of shape $(3, 4, 5)$ have?
2. Order Identification: Identify the order and shape of each object:
   - (a) A grayscale image 64×64.
   - (b) A video: 30 frames, each 128×128 RGB.
   - (c) A dataset with 1000 samples, each a 20×20 grayscale image.
3. Index Practice: For a tensor T_{ijk} with shape $(2, 2, 2)$, list explicitly all the index triples (i, j, k).

4. Shape Transformation: Flatten a tensor with shape $(5, 4, 3)$ into a matrix. What two possible shapes could the matrix have (depending on how you group the indices)?
5. Thought Experiment: Why is a scalar sometimes called a "0th-order tensor"? How does this viewpoint help unify the hierarchy of scalars, vectors, matrices, and higher-order tensors?

3.2 Covariant vs. Contravariant Indices

So far, we've treated indices as simple "positions in an array." But in multilinear algebra, indices carry roles. Some belong to vectors (contravariant), others to covectors (covariant). Understanding this distinction is crucial for how tensors behave under change of basis.

Vectors vs. Covectors

- Vectors are elements of a space V. They transform with the basis.
- Covectors (linear functionals) are elements of the dual space V^*. They transform with the *inverse transpose- of the basis change.

This leads to two kinds of indices:

- Contravariant indices (upper): v^i, coordinates of a vector.
- Covariant indices (lower): ω_j, coordinates of a covector.

Tensors Mixing Both

A tensor may have both types of indices:

$$T^{i_1 i_2 \cdots i_p}_{j_1 j_2 \cdots j_q},$$

which means it accepts q vectors and p covectors as inputs (or outputs, depending on interpretation).

Examples:

- A vector $v^i \to$ contravariant (upper index).
- A covector $\omega_j \to$ covariant (lower index).
- A bilinear form $B_{ij} \to$ two covariant indices.
- A linear map $A^i{}_j \to$ one up and one down (it eats a vector, gives back a vector).

Why Two Types?

This distinction is not cosmetic:

- When we change basis, vectors and covectors transform in "opposite" ways.
- Having both ensures that tensor equations describe intrinsic relationships, independent of coordinates.

Example: Inner product

$$\langle v, w \rangle = g_{ij} v^i w^j.$$

Here g_{ij} is a metric tensor (covariant), combining two contravariant vectors into a scalar.

Pictures Before Symbols

- Contravariant: arrows pointing "outward" (directions in space).
- Covariant: measuring devices pointing "inward" (hyperplanes that assign numbers to arrows).
- Tensors: diagrams with arrows in and out, representing how they connect inputs to outputs.

This picture-based intuition helps prevent index mistakes when writing formulas.

Why This Matters

- Covariant vs. contravariant indices explain the geometry of tensors, not just their array form.
- It prepares us for raising and lowering indices with inner products.
- It ensures we can handle basis changes correctly (Chapter 12 will revisit this in detail).

Exercises

1. Identify Index Type: For each object, say whether its indices are covariant, contravariant, or mixed:
   - (a) v^i.
   - (b) ω_j.
   - (c) $A^i{}_j$.
   - (d) B_{ij}.
2. Dual Basis Practice: In $\mathbb{R}^2$ with basis e_1, e_2 and dual basis e^1, e^2:

- Write a vector $v = 3e_1 + 4e_2$ in coordinates v^i.
- Evaluate $\omega(v)$ for $\omega = 2e^1 - e^2$.

3. Basis Change Intuition: Suppose we scale the basis of $\mathbb{R}^2$ by 2: $e'_i = 2e_i$.

   - How do the contravariant coordinates v^i of a vector change?
   - How do the covariant coordinates ω_i of a covector change?

4. Mixed Tensor Example: Interpret the meaning of a tensor $T^i{}_j$ acting on a vector v^j. What kind of object is the result?

5. Thought Experiment: Why do we need both contravariant and covariant indices to describe something like the dot product? What would go wrong if we only allowed one type?

3.3 Change of Basis Rules in Coordinates

So far we have seen that indices can be contravariant (upper) or covariant (lower). The key difference appears when we change basis. Multilinear algebra is all about writing rules that remain valid regardless of coordinates, and basis transformations reveal why the distinction is essential.

Vectors Under Change of Basis

Let $V = \mathbb{R}^n$. Suppose we change from an old basis $\{e_i\}$ to a new basis $\{e'_i\}$:

$$e'_i = P^j{}_i e_j,$$

where P is the change-of-basis matrix.

- A vector v has coordinates v^i in the old basis and v'^i in the new basis.
- The relation is:

$$v'^i = (P^{-1})^i{}_j v^j.$$

Thus, contravariant components transform with the inverse of the basis change.

Covectors Under Change of Basis

Now consider a covector $\omega \in V^*$, expressed in the old basis as ω_i. Under the same change of basis:

$$\omega'_i = P^j{}_i \, \omega_j.$$

Thus, covariant components transform directly with the basis change matrix.

General Tensor Transformation

A tensor with both covariant and contravariant indices transforms by applying these rules to each index:

$$T'^{i_1 \ldots i_p}{}_{j_1 \ldots j_q} = (P^{-1})^{i_1}{}_{k_1} \cdots (P^{-1})^{i_p}{}_{k_p} \, P^{\ell_1}{}_{j_1} \cdots P^{\ell_q}{}_{j_q} \, T^{k_1 \ldots k_p}{}_{\ell_1 \ldots \ell_q}.$$

- Upper indices get P^{-1}.
- Lower indices get P.
- This ensures tensor equations remain coordinate-independent.

Simple Example: Linear Maps

A linear map $A : V \to V$ has components $A^i{}_j$. Under change of basis:

$$A'^i{}_j = (P^{-1})^i{}_k \, A^k{}_\ell \, P^\ell{}_j.$$

This is the familiar similarity transformation:

$$[A]_{new} = P^{-1}[A]_{old}P.$$

Why This Matters

- Basis changes test whether your formulas are intrinsic or just coordinate artifacts.
- The distinction between covariant and contravariant is precisely what makes tensor equations survive basis changes intact.
- In physics, this explains why laws (like Maxwell's equations or stress-strain relations) remain valid no matter what coordinates we choose.

Exercises

1. Vector Transformation: Let $P = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$.
   - If $v = (4, 6)$ in the old basis, what are its coordinates in the new basis?
2. Covector Transformation: With the same P, let $\omega = (1, 2)$ in the old basis. What are its coordinates in the new basis?
3. Matrix Transformation: Let

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

   Compute $A' = P^{-1}AP$.
4. Tensor Component Count: How many transformation matrices P and P^{-1} appear in the formula for a tensor of type $(2, 1)$?
5. Thought Experiment: Why would formulas break if we treated all indices as the same type (ignoring covariant vs. contravariant)? Consider the dot product as an example.

3.4 Einstein Summation and Index Hygiene

When working with tensors, writing every summation explicitly quickly becomes messy. To keep formulas clean, mathematicians and physicists use the Einstein summation convention and a set of informal "index hygiene" rules.

Einstein Summation Convention

The rule is simple: whenever an index appears once up and once down, you sum over it.

Example in $\mathbb{R}^3$:

$$y^i = A^i{}_j x^j$$

means

$$y^i = \sum_{j=1}^{3} A^i{}_j x^j.$$

This compact notation hides the summation symbol but makes multilinear expressions much easier to read.

Free vs. Dummy Indices

- Free indices: appear only once in a term; they label the components of the result.
- Dummy indices: appear exactly twice (once up, once down); they are summed over and can be renamed arbitrarily.

Example:

$$z^i = B^i{}_j x^j$$

Here i is free (labels components of z), and j is a dummy index (summed).

Index Hygiene Rules

1. Never use the same index more than twice in a term.
2. Never mix up free and dummy indices.
3. Rename dummy indices freely if it helps clarity.

Example of bad hygiene:

$$A^i{}_i \, x^i$$

This is ambiguous, because i appears three times. Correct it by renaming:

$$(A^i{}_i) \, x^j.$$

Examples of Einstein Notation

- Dot product: $\langle v, w \rangle = v^i w_i$.
- Matrix-vector product: $y^i = A^i{}_j x^j$.
- Bilinear form: $B(v, w) = B_{ij} v^i w^j$.
- Trace of a matrix: $\mathrm{tr}(A) = A^i{}_i$.

Why This Matters

- Einstein summation is the language of tensors: concise, unambiguous, and basis-independent.
- It allows formulas to be read structurally, focusing on how indices connect rather than on summation symbols.
- Practicing good index hygiene prevents mistakes when manipulating complicated expressions.

Exercises

1. Summation Practice: Expand the Einstein-summed expression

$$y^i = A^i{}_j x^j$$

   explicitly for $i = 1, 2$ when

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad x = (5, 6).$$

2. Dot Product Check: Show that $v^i w_i$ is invariant under a change of dummy index name (e.g. rewrite with j instead of i).

3. Trace Calculation: For

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -3 \end{bmatrix},$$

   compute $\mathrm{tr}(A)$ using Einstein notation.

4. Index Hygiene: Identify the mistake in the following expression and correct it:

$$C^i = A^i{}_j B^j{}_j x^j.$$

5. Thought Experiment: Why does Einstein notation require one index up and one down to imply summation? What would go wrong if we summed whenever indices simply repeated, regardless of position?

Chapter 4. Tensors as Multilinear Maps

4.1 Multilinearity and Currying

So far, we treated tensors as arrays of numbers. Now we switch to the map viewpoint: tensors as functions that are linear in each argument separately. This is the most natural way to see how tensors *act*.

What Does Multilinear Mean?

A map $T : V_1 \times V_2 \times \cdots \times V_k \to \mathbb{R}$ (or to another vector space) is multilinear if it is linear in each slot, while the others are fixed.

Example (bilinear):

$$B(av_1 + bv_2, w) = aB(v_1, w) + bB(v_2, w).$$

The same rule holds in each argument.

- Linear $\to$ 1 slot.
- Bilinear $\to$ 2 slots.
- Trilinear $\to$ 3 slots.
- k-linear $\to$ k slots.

Examples of Multilinear Maps

- Dot product: $\langle v, w \rangle$, bilinear.
- Matrix-vector action: $A(v)$, linear (1 slot).
- Determinant in $\mathbb{R}^2$: $\det(u, v) = u_1 v_2 - u_2 v_1$, bilinear.
- Scalar triple product: $u \cdot (v \times w)$, trilinear.

These all satisfy linearity in each argument separately.

Currying Viewpoint

Another way to think about multilinear maps is through currying:

- A bilinear map $B : V \times W \to \mathbb{R}$ can be seen as a function

$$B(v, -) : W \to \mathbb{R}, \quad w \mapsto B(v, w).$$

  So, fixing one input gives a linear map in the remaining argument.

- In general, a k-linear map can be seen as a nested sequence of linear maps, each taking one input at a time.

This perspective helps connect multilinear maps with ordinary linear maps, by viewing them as "linear maps into linear maps."

Why This Matters

- This viewpoint clarifies how tensors can be "evaluated" by plugging in vectors.
- It connects with functional programming ideas (currying and partial application).
- It bridges between arrays (coordinates) and abstract multilinear functionals.

Exercises

1. Bilinearity Check: Verify directly that the dot product $\langle (x_1, y_1), (x_2, y_2) \rangle = x_1 x_2 + y_1 y_2$ is bilinear.
2. Currying Example: Let $B(u, v) = u_1 v_1 + 2u_2 v_2$. For a fixed $u = (1, 2)$, write the resulting linear functional on v.
3. Determinant as Bilinear Form: Show that $\det(u, v) = u_1 v_2 - u_2 v_1$ is bilinear in $\mathbb{R}^2$.
4. Trilinear Example: Prove that the scalar triple product $u \cdot (v \times w)$ is trilinear by checking linearity in u.
5. Thought Experiment: Why might it be useful to think of a bilinear map as a linear map into the dual space, i.e. $B : V \to W^*$?

4.2 Evaluation with Vectors and Covectors

In the map viewpoint, tensors are best understood by how they act on inputs. Depending on their type (covariant or contravariant indices), tensors expect vectors, covectors, or both.

Feeding Vectors into Covariant Slots

A purely covariant tensor T_{ij} is a multilinear map

$$T : V \times V \to \mathbb{R}.$$

- You feed in two vectors $u, v \in V$.
- The output is a scalar: $T(u, v) = T_{ij} u^i v^j$.

Example:

- A bilinear form (like an inner product) is a covariant 2-tensor.

Feeding Covectors into Contravariant Slots

A purely contravariant tensor T^{ij} is a multilinear map

$$T : V^- \times V^- \to \mathbb{R}.$$

- You feed in two covectors $\alpha, \beta \in V^*$.
- The output is a scalar: $T(\alpha, \beta) = T^{ij} \alpha_i \beta_j$.

Mixed Tensors: Both Types of Inputs

A mixed tensor $T^i{}_j$ acts as a map:

$$T : V \times V^- \to \mathbb{R}.$$

But more naturally, it can be seen as:

- taking a vector and giving back a vector,
- or taking a covector and giving back a covector.

Example:

- A linear operator $A : V \to V$ has components $A^i{}_j$. Given v^j, it produces another vector $w^i = A^i{}_j v^j$.

Evaluating Step by Step

- Pick a tensor.
- Plug in the right kind of inputs (vector or covector) into the right slots.
- Contract the matching indices (up with down).
- The result is either a scalar, a vector, or another tensor (depending on how many free indices remain).

Why This Matters

- This clarifies the action of tensors, not just their coordinates.
- Evaluation explains why indices are placed up or down: they indicate what kind of input the tensor expects.
- It connects naturally with Einstein summation: every evaluation is just an index contraction.

Exercises

1. Evaluation of Bilinear Form: Let $B_{ij} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$. Compute $B(u, v)$ for $u = (1, 1), v = (2, 3)$.
2. Linear Operator Action: Let $A^i{}_j = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
   - Apply A to $v = (4, 5)$.
   - Interpret the result.
3. Covector Evaluation: If $\omega = (2, -1)$ and $v = (3, 4)$, compute $\omega(v)$.
4. Mixed Tensor Evaluation: For $T^i{}_j = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$, find the output vector when applied to $v = (1, 2)$.
5. Thought Experiment: Why does the distinction between covariant and contravariant slots matter when evaluating tensors? What would go wrong if we treated them the same?

4.3 From Maps to Arrays (and Back) via a Basis

So far, we've looked at tensors as multilinear maps (abstract) and as arrays of numbers (concrete). The bridge between these viewpoints is a choice of basis.

Step 1: Start with a Multilinear Map

Suppose $T : V \times W \to \mathbb{R}$ is bilinear.

- On its own, T is just a rule for combining vectors into a number.
- Example: in $\mathbb{R}^2$, $T((x_1, x_2), (y_1, y_2)) = x_1y_1 + 2x_2y_2$.

Step 2: Choose Bases

Let $\{e_i\}$ be a basis for V and $\{f_j\}$ a basis for W.

- We can evaluate $T(e_i, f_j)$ for each pair of basis vectors.
- These numbers form an array of components T_{ij}.

So in a basis, the multilinear map has a coordinate table.

Step 3: Using the Array to Reconstruct the Map

For general vectors $v = v^i e_i$ and $w = w^j f_j$:

$$T(v, w) = T_{ij} v^i w^j.$$

- Here, the coefficients v^i, w^j are the coordinates of v, w.
- The array entries T_{ij} tell us how the map acts on basis vectors.

Thus:

- Abstract definition: multilinear rule.
- Concrete representation: an array of numbers (depends on basis).

Step 4: General Tensors

For a tensor of type (p, q):

- Choose a basis for V.
- Evaluate the tensor on all combinations of q basis vectors and p dual basis covectors.
- The results form a multidimensional array $T^{i_1 \cdots i_p}{}_{j_1 \cdots j_q}$.

In coordinates:

$$T(v_1, \ldots, v_q, \omega^1, \ldots, \omega^p) = T^{i_1 \cdots i_p}{}_{j_1 \cdots j_q} \, v_1^{j_1} \cdots v_q^{j_q} \, \omega_{i_1}^1 \cdots \omega_{i_p}^p.$$

Why This Matters

- It explains why tensors can be stored as arrays of numbers: those numbers are just evaluations on basis elements.
- It shows why basis choice matters for components but not for the tensor itself.
- It unifies the map viewpoint and the array viewpoint into one consistent framework.

Exercises

1. Matrix from Bilinear Form: Let $T((x_1, x_2), (y_1, y_2)) = 3x_1y_1 + 2x_1y_2 + x_2y_1$.
   - Write down the matrix $[T_{ij}]$.
   - Compute $T((1, 2), (3, 4))$ using both the formula and the matrix.
2. Basis Choice Effect: In $\mathbb{R}^2$, let T be the dot product.
   - What is the matrix of T in the standard basis?
   - What is the matrix of T in the basis $(1, 1), (1, -1)$?
3. Coordinate Reconstruction: Let $T_{ij} = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$. Compute $T(v, w)$ for $v = (2, 1), w = (1, 4)$.
4. Higher Order Example: Suppose T is a 3rd-order tensor with components T_{ijk}. Write the formula for $T(u, v, w)$ in terms of T_{ijk}, u^i, v^j, and w^k.
5. Thought Experiment: Why can the same abstract tensor have different component arrays depending on the basis? What stays the same across all choices?

4.4 Universal Examples: Bilinear Forms, Trilinear Mixing

To make the map viewpoint concrete, let's look at a few universal examples. These are classic multilinear maps that keep reappearing in mathematics, physics, and data science.

Example 1: Bilinear Forms

A bilinear form is a covariant 2-tensor:

$$B : V \times V \to \mathbb{R}, \quad B(u, v) = B_{ij}u^i v^j.$$

- Dot product: $\langle u, v \rangle = \delta_{ij}u^i v^j$.
- General quadratic form: $Q(v) = v^\top Av$.
- Applications: measuring angles, energy, distance.

Example 2: Determinant (Alternating Multilinear Form)

The determinant in $\mathbb{R}^n$ is an n-linear map of the column vectors:

$$\det(v_1, \ldots, v_n).$$

It is:

- Multilinear: linear in each column separately.
- Alternating: if two inputs are the same, the determinant vanishes.
- Geometric meaning: volume of the parallelepiped spanned by the vectors.

Example 3: Scalar Triple Product (Trilinear)

In $\mathbb{R}^3$:

$$[u, v, w] = u \cdot (v \times w).$$

- Trilinear: linear in each of u, v, w.
- Geometric meaning: signed volume of the parallelepiped formed by u, v, w.
- Appears in mechanics (torques, volumes, orientation tests).

Example 4: Tensor Contraction in Applications

Suppose $T : V \times W \times X \to \mathbb{R}$ with components T_{ijk}.

- Fixing one input reduces T to a bilinear form.
- Fixing two inputs reduces T to a linear functional.

This is how general multilinear maps can be "partially evaluated."

Example 5: Mixing Signals (Trilinear Mixing)

In signal processing and ML, a trilinear map appears naturally:

$$M(u, v, w) = \sum_{i,j,k} T_{ijk} u^i v^j w^k.$$

- Example: a 3D convolution kernel.
- Applications: image processing, tensor regression, multiway data analysis.

Why These Examples Matter

- They show how multilinear maps generalize familiar ideas (dot product → determinant → triple product).
- They illustrate how multilinearity encodes geometry (lengths, areas, volumes).
- They connect abstract tensor notation to real applications (ML, physics, graphics).

Exercises

1. Dot Product as Bilinear Form: Show that the dot product $\langle u, v\rangle = u^i v^i$ is bilinear.
2. Determinant in 2D: For $u = (1, 0), v = (1, 2)$, compute $\det(u, v)$. Interpret geometrically.
3. Triple Product: Compute $[u, v, w]$ for $u = (1, 0, 0), v = (0, 1, 0), w = (0, 0, 1)$.
4. Signal Mixing Example: Let $T_{ijk} = 1$ if $i = j = k$, and 0 otherwise, for indices $1 \leq i, j, k \leq 2$. Compute $M(u, v, w)$ for $u = (1, 2), v = (3, 4), w = (5, 6)$.
5. Thought Experiment: Why do determinants and triple products count as multilinear maps, not just algebraic formulas?

Chapter 5. Tensors as Elements of Tensor Products

5.1 Constructing $V \otimes W$: Intuition and Goals

So far, we've looked at tensors as arrays and as multilinear maps. The third viewpoint places them as elements of tensor product spaces. This perspective is abstract, but it unifies everything: it explains *why- multilinear maps correspond to arrays and* how- they transform consistently.

Motivation

Suppose you want to encode a bilinear map $B : V \times W \to \mathbb{R}$.

- You could record all values of $B(e_i, f_j)$ on basis elements.
- You could write it as an array B_{ij}.
- Or you could find a new space where a single element encodes all the bilinear behavior.

That new space is the tensor product space $V \otimes W$.

Intuition: Building Blocks

- Given $v \in V$ and $w \in W$, we form a simple tensor $v \otimes w$.
- Think of $v \otimes w$ as a "formal symbol" that remembers both vectors at once.
- Then we allow linear combinations:

$$u = \sum_k v_k \otimes w_k.$$

- These combinations form a new vector space: $V \otimes W$.

Universal Property (informal)

The tensor product is defined so that:

- Every bilinear map $B : V \times W \to U$ factors uniquely through a linear map $\tilde{B} : V \otimes W \to U$.
- In plain words: "Bilinear maps are the same thing as linear maps out of a tensor product."

This property makes $V \otimes W$ the *right- space to house tensors.

Example

If $V = \mathbb{R}^2, W = \mathbb{R}^3$, then $V \otimes W$ has dimension $2 \times 3 = 6$.

- Basis: $e_i \otimes f_j$ with $i = 1, 2; j = 1, 2, 3$.
- A general element:

$$u = \sum_{i=1}^{2} \sum_{j=1}^{3} c_{ij}(e_i \otimes f_j).$$

- Coordinates c_{ij} are exactly the array components of a bilinear map.

Goals of This Viewpoint

- Provide a basis-independent foundation for tensors.
- Unify maps, arrays, and algebra into one framework.
- Generalize easily: $V_1 \otimes \cdots \otimes V_k$ encodes k-linear maps.

Exercises

1. Dimension Count: If $\dim V = 3$ and $\dim W = 4$, what is $\dim(V \otimes W)$?
2. Simple Tensor Example: In $\mathbb{R}^2 \otimes \mathbb{R}^2$, write the simple tensor $(1, 2) \otimes (3, 4)$ as a linear combination of basis elements $e_i \otimes e_j$.
3. Array to Tensor: For the bilinear form $B((x_1, x_2), (y_1, y_2)) = 2x_1y_1 + 3x_2y_2$, find the corresponding tensor in $\mathbb{R}^2 \otimes \mathbb{R}^2$.
4. Universal Property Check: Suppose $B : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ is given by $B(u, v) = u^\top v$.
   - Show how B factors through a linear map $\tilde{B} : \mathbb{R}^2 \otimes \mathbb{R}^2 \to \mathbb{R}$.
5. Thought Experiment: Why is it useful to replace "bilinear maps" with "linear maps from a bigger space"? How does this simplify reasoning and computation?

5.2 Simple vs. General Tensors

Now that we've seen how to build the tensor product space, we need to distinguish between its basic "atoms" and more complicated elements.

Simple (Decomposable) Tensors

A simple tensor (also called pure or decomposable) is one that can be written as a single tensor product:

$$u = v \otimes w.$$

Examples:

- In $\mathbb{R}^2 \otimes \mathbb{R}^3$, $(1, 2) \otimes (0, 1, 1)$ is simple.
- Its array of components is an outer product:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 2 & 2 \end{bmatrix}.$$

So simple tensors correspond to rank-one arrays (outer products).

General Tensors

A general tensor in $V \otimes W$ is a linear combination of simple tensors:

$$T = \sum_{k=1}^{m} v_k \otimes w_k.$$

- Some tensors can be expressed as a single simple tensor.
- Most cannot; they require a sum of several simple tensors.
- The minimal number of terms in such a sum is called the tensor rank (analogous to matrix rank for order-2 tensors).

Matrix Analogy

- Simple tensors rank-one matrices (outer products of a column and a row).
- General tensors arbitrary matrices (sums of rank-one pieces).

For higher-order tensors, the situation is the same:

- Simple tensor = one outer product of vectors.
- General tensor = sum of several such products.

Why This Matters

- The distinction underlies tensor decomposition methods (CP, Tucker, TT).
- Simple tensors capture the "building blocks" of tensor spaces.
- General tensors encode richer structure, but often we approximate them by a small number of simple ones.

Exercises

1. Simple or Not? In $\mathbb{R}^2 \otimes \mathbb{R}^2$, determine whether

   $$T = e_1 \otimes e_1 + e_2 \otimes e_2$$

   is a simple tensor.

2. Outer Product Calculation: Compute the outer product of $u = (1, 2)$ and $v = (3, 4, 5)$. Write the resulting 2×3 matrix.

3. Rank-One Check: Given the matrix

$$M = \begin{bmatrix} 2 & 4 \\ 3 & 6 \end{bmatrix},$$

show that M is a simple tensor (rank one) by writing it as $u \otimes v$.

4. General Tensor Example: Express

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

as a sum of two simple tensors in $\mathbb{R}^2 \otimes \mathbb{R}^2$.

5. Thought Experiment: Why are most tensors not simple? What does this imply about the usefulness of tensor decompositions in applications like machine learning?

5.3 Dimension and Bases of Tensor Products

Having distinguished simple tensors from general ones, let's now describe the structure of a tensor product space: its dimension and basis.

Dimension Formula

If V and W are finite-dimensional vector spaces, then:

$$\dim(V \otimes W) = \dim(V) \cdot \dim(W).$$

More generally, for $V_1, V_2, \ldots, V_k$:

$$\dim(V_1 \otimes V_2 \otimes \cdots \otimes V_k) = \prod_{i=1}^{k} \dim(V_i).$$

This matches our intuition that each index multiplies the number of "slots" in the array representation.

Basis of a Tensor Product

Suppose $\{e_i\}$ is a basis for V and $\{f_j\}$ is a basis for W.

- Then the set

$$\{e_i \otimes f_j : 1 \leq i \leq \dim V,\ 1 \leq j \leq \dim W\}$$

  forms a basis for $V \otimes W$.

Example:

- If $\dim V = 2, \dim W = 3$: $\{e_1 \otimes f_1, e_1 \otimes f_2, e_1 \otimes f_3, e_2 \otimes f_1, e_2 \otimes f_2, e_2 \otimes f_3\}$ is a basis (6 elements).

Coordinates in the Basis

A general tensor can be expressed as:

$$T = \sum_{i,j} c_{ij} \, (e_i \otimes f_j),$$

where the coefficients c_{ij} form the familiar matrix (array) representation.

For higher-order tensor products, the same logic applies:

$$T = \sum_{i,j,k} c_{ijk} \, (e_i \otimes f_j \otimes g_k).$$

Analogy with Arrays

- Basis elements correspond to array positions.
- Coefficients are the entries.
- The dimension formula tells us the total number of independent entries.

Thus, the product-space viewpoint and the array viewpoint are perfectly aligned.

Why This Matters

- The basis description explains why tensor product spaces have the same "shape" as arrays.
- It provides a rigorous foundation for the component representation of tensors.
- It prepares us to reconcile all three viewpoints (array, map, product space).

Exercises

1. Dimension Count: Compute $\dim(\mathbb{R}^2 \otimes \mathbb{R}^3)$. List a basis explicitly.
2. Basis Expansion: Express $T = (1, 2) \otimes (3, 4, 5)$ in terms of the standard basis $e_i \otimes f_j$.
3. Higher-Order Example: If $\dim U = 2, \dim V = 2, \dim W = 2$, what is $\dim(U \otimes V \otimes W)$?
4. Coefficient Identification: Let $T = e_1 \otimes f_1 + 2e_2 \otimes f_3$. What are the nonzero coefficients c_{ij}?
5. Thought Experiment: Why does the formula $\dim(V \otimes W) = \dim V \cdot \dim W$ make sense if you think of arrays? How does this match the earlier idea of "shape"?

5.4 Three Viewpoints Reconciled

We now have three different ways to understand tensors: as arrays, as multilinear maps, and as elements of tensor product spaces. Each looks different, but they are all equivalent perspectives on the same object.

1. Array View (Concrete)

- A tensor is a multi-dimensional array of numbers.
- Its entries depend on a choice of basis.
- Operations are done by indexing and summing (Einstein notation).
- Example: a $2 \times 3 \times 4$ tensor is just a 3D block of 24 numbers.

Strengths: intuitive, easy to compute. Limitations: depends heavily on coordinates.

2. Map View (Functional)

- A tensor is a multilinear function that takes vectors and covectors as inputs and produces scalars or other tensors.
- Example:
  - Dot product: bilinear map.
  - Determinant: alternating multilinear map.
  - Linear operators: mixed tensors.

Strengths: coordinate-free, emphasizes action. Limitations: abstract, less computational.

3. Product Space View (Foundational)

- A tensor is an element of a tensor product space $V^{\otimes p} \otimes (V^*)^{\otimes q}$.
- Basis choice identifies this element with an array of components.
- Universal property: multilinear maps from vectors linear maps out of tensor products.

Strengths: rigorous, unifying, handles all cases. Limitations: abstract at first, harder to visualize.

Reconciling Them

- Array = product-space + basis.
- Map = product-space + evaluation on inputs.
- Product-space = abstract "home" that makes the other two consistent.

So:

- If you want computation, use arrays.
- If you want geometry or physics intuition, use maps.
- If you want rigor and generality, use tensor products.

They are not rivals but complementary languages describing the same object.

Why This Matters

Understanding all three viewpoints and moving fluidly between them is the hallmark of real mastery. It's like being fluent in three languages: you choose the one that fits the context, but you know they describe the same reality.

Exercises

1. Array Map: Let $T_{ij} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

   - Write $T(u, v) = T_{ij} u^i v^j$.
   - Evaluate for $u = (1, 0), v = (0, 1)$.

2. Map Product-Space: Show that the bilinear form $B(u, v) = u_1 v_1 + 2u_2 v_2$ corresponds to an element of $\mathbb{R}^2 \otimes \mathbb{R}^2$.

3. Array Product-Space: If $T = 3e_1 \otimes f_2 + 5e_2 \otimes f_3$, what are the nonzero array components T_{ij}?

4. Three Languages: Write the dot product of $u, v \in \mathbb{R}^2$ in each of the three viewpoints:

   - Array form
   - Multilinear map
   - Tensor product element

5. Thought Experiment: Why is it valuable to know all three viewpoints instead of just one? Give an application where each viewpoint is the most natural.

Part III. Core Operations on Tensors

Chapter 6. Building Blocks

6.1 Tensor (Outer) Product

Now that we've reconciled the three viewpoints, we can explore the core operations on tensors. The first and most fundamental is the tensor product itself, also known as the outer product in the array viewpoint.

Definition

Given two vectors $u \in V$ and $v \in W$, their tensor product is

$$u \otimes v \in V \otimes W.$$

- It is bilinear in u, v.
- In a basis, it looks like an array of rank one (outer product).

Example with Vectors

If $u = (1, 2)$ and $v = (3, 4, 5)$:

$$u \otimes v = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 3 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 5 \\ 6 & 8 & 10 \end{bmatrix}.$$

This is a 2×3 array - exactly the outer product.

Higher-Order Outer Products

We can extend this:

$$u \otimes v \otimes w,$$

produces a 3rd-order tensor, with entries

$$(u \otimes v \otimes w)_{ijk} = u_i v_j w_k.$$

In general, the outer product of k vectors is a simple tensor of order k.

Properties

- Bilinearity: $(au + bu') \otimes v = a(u \otimes v) + b(u' \otimes v)$.
- Noncommutativity: $u \otimes v \neq v \otimes u$ (unless dimensions align and we impose symmetry).
- Rank-One Structure: Outer products produce the simplest tensors, forming the building blocks of general tensors.

Why This Matters

- The tensor product (outer product) is the construction rule for higher-order tensors.
- It connects vector multiplication with multidimensional arrays.
- Many decompositions (CP, Tucker, tensor train) rely on sums of outer products.

Exercises

1. Basic Outer Product: Compute $(2, 1) \otimes (1, 3)$. Write it as a 2×2 matrix.
2. Higher Order: Compute $(1, 2) \otimes (0, 1) \otimes (3, 4)$. What is the shape of the resulting tensor, and how many entries does it have?
3. Linearity Check: Verify that $(u+u') \otimes v = u \otimes v + u' \otimes v$ for $u = (1, 0), u' = (0, 1), v = (2, 3)$.
4. Rank-One Matrix: Show that every outer product $u \otimes v$ is a rank-one matrix. Give an example.
5. Thought Experiment: Why does the outer product produce the "simplest" tensors? How does this idea generalize from matrices to higher-order arrays?

6.2 Contraction: Summing Paired Indices

If the tensor product (outer product) builds bigger tensors, then contraction is the operation that reduces them by "pairing up" an upper and a lower index and summing over it. This is the natural generalization of the dot product and the trace.

Definition

Given a tensor $T^i{}_j$, contracting on the index i and j means:

$$\text{contr}(T) = T^i{}_i.$$

- You match one upper and one lower index.
- You sum over that index.
- The result is a new tensor of lower order.

Familiar Examples

1. Dot Product: For vectors u^i and v_i:

   $$u^i v_i$$

   is a contraction, producing a scalar.

2. Matrix-Vector Multiplication:

   $$y^i = A^i{}_j x^j$$

   contracts on j, leaving a vector y^i.

3. Trace of a Matrix:

   $$\text{tr}(A) = A^i{}_i$$

   contracts on the row and column indices, producing a scalar.

General Contraction

If $T^{i_1 i_2 \cdots i_p}{}_{j_1 j_2 \cdots j_q}$ is a tensor, contracting on i_k and j_ℓ yields a tensor of type $(p-1, q-1)$.

This is how tensor calculus generalizes dot products, matrix multiplication, and traces into one operation.

Why This Matters

- Contraction is the dual operation to the tensor product.
- Outer product increases order, contraction decreases order.
- Together, they generate most tensor operations in mathematics, physics, and machine learning.

Exercises

1. Dot Product as Contraction: Show explicitly that $u \cdot v = u^i v_i$ is a contraction.
2. Matrix-Vector Multiplication: Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $x = (5, 6)$. Compute $y^i = A^i{}_j x^j$.
3. Trace Calculation: For $A = \begin{bmatrix} 2 & 1 \\ 0 & -3 \end{bmatrix}$, compute $\mathrm{tr}(A)$ via contraction.
4. Higher-Order Contraction: Suppose $T^{ij}{}_{k\ell}$ is a 4th-order tensor. What is the order and type of the tensor obtained by contracting on j and ℓ?
5. Thought Experiment: Why can contraction only happen between one upper and one lower index? What would break if you tried to contract two uppers or two lowers?

6.3 Permutations of Modes: Transpose, Unfold, Matricize

Tensors often need to be rearranged so that their structure fits the operation we want to perform. This section introduces three related operations: permutation of modes, transpose, and matricization (unfolding).

Permuting Modes

A tensor of order k has k indices:

$$T_{i_1 i_2 \ldots i_k}.$$

A permutation of modes reorders these indices.

- Example: if T is order 3, with entries T_{ijk}, then a permutation might yield T_{kij}.
- This doesn't change the data, just how we view the axes.

Transpose (Matrix Case)

For a 2nd-order tensor (matrix), permutation of the two indices corresponds to the familiar transpose:

$$A_{ij} \mapsto A_{ji}.$$

This is just a special case of permuting modes.

Matricization (Unfolding)

For higher-order tensors, it is often useful to flatten them into matrices.

- Choose one index (or a group of indices) to form the rows.
- The remaining indices form the columns.

Example: For a 3rd-order tensor T_{ijk}:

- Mode-1 unfolding: rows indexed by i, columns by (j, k).
- Mode-2 unfolding: rows indexed by j, columns by (i, k).
- Mode-3 unfolding: rows indexed by k, columns by (i, j).

This is widely used in numerical linear algebra and machine learning.

Why Permutations Matter

- They allow us to reinterpret tensors for algorithms (e.g., apply matrix methods to unfolded tensors).
- They help align indices correctly before contraction.
- They reveal symmetry or structure hidden in the raw array.

Exercises

1. Permutation Practice: If T_{ijk} has shape $(2, 3, 4)$, what is the shape of the permuted tensor T_{kij}?
2. Matrix Transpose: Write the transpose of

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

3. Mode-1 Unfolding: For a tensor T_{ijk} of shape $(2, 2, 2)$, what is the shape of its mode-1 unfolding?
4. Reversibility: Show that permuting modes twice with inverse permutations returns the original tensor.
5. Thought Experiment: Why might unfolding a tensor into a matrix help in data analysis (e.g., PCA, SVD)? What do we lose by unfolding?

6.4 Kronecker Product vs. Tensor Product

The tensor product and the Kronecker product are closely related, but they live in slightly different worlds. Beginners often confuse them because they both "blow up" dimensions in similar ways. Let's carefully separate them.

Tensor Product (Abstract)

- Given two vector spaces V, W, their tensor product $V \otimes W$ is a new vector space.
- If $\dim V = m$ and $\dim W = n$, then $\dim(V \otimes W) = mn$.
- Basis: $e_i \otimes f_j$.
- Purpose: encodes bilinear maps as linear maps.

Example: For $u = (u_1, u_2)$, $v = (v_1, v_2, v_3)$:

$$u \otimes v = \begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \end{bmatrix}.$$

Kronecker Product (Matrix Operation)

- Given two matrices $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, the Kronecker product is:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}.$$

- This produces a block matrix of size $(mp) \times (nq)$.
- Used heavily in linear algebra identities, signal processing, and ML.

Relation Between the Two

- The Kronecker product is the matrix representation of the tensor product once bases are chosen.
- In other words:
  - Tensor product = basis-independent, abstract.
  - Kronecker product = concrete array form.

Examples

1. Vector Tensor Product: $(1,2) \otimes (3,4) = \begin{bmatrix} 3 & 4 \\ 6 & 8 \end{bmatrix}$.
2. Matrix Kronecker Product:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1\cdot B & 2\cdot B \\ 3\cdot B & 4\cdot B \end{bmatrix}.$$

Why This Matters

- Tensor product explains the theory of multilinearity.
- Kronecker product is the computational tool, letting us work with actual arrays.
- Keeping them distinct prevents confusion between *concept- and* representation*.

Exercises

1. Basic Kronecker: Compute

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

2. Dimension Check: If A is 2×3 and B is 3×4, what is the size of $A \otimes B$?
3. Tensor vs. Kronecker: Show explicitly that for vectors u, v, the tensor product $u \otimes v$ corresponds to the Kronecker product of u and $v^\top$.
4. Outer Product Relation: Express the outer product $u \otimes v$ as a Kronecker product when u, v are vectors.
5. Thought Experiment: Why is it useful to separate the abstract tensor product from the concrete Kronecker product? What problems might arise if we conflated them?

Chapter 7. Symmetry and (Anti)Symmetry

7.1 Symmetrization Operators

One of the most important features of tensors is how they behave under permutations of indices. Some tensors stay the same when you swap indices (symmetric), others change sign (antisymmetric). To formalize this, we introduce symmetrization operators.

Symmetric Tensors

A tensor T_{ij} is symmetric if swapping indices does not change it:

$$T_{ij} = T_{ji}.$$

More generally, a k-tensor $T_{i_1 i_2 \ldots i_k}$ is symmetric if:

$$T_{i_{\pi(1)} i_{\pi(2)} \ldots i_{\pi(k)}} = T_{i_1 i_2 \ldots i_k}$$

for every permutation π.

Example:

- The Hessian matrix of a scalar function is symmetric.
- A covariance matrix is symmetric.

Symmetrization Operator

Given any tensor T, its symmetrized version is obtained by averaging over all permutations of indices:

$$\mathrm{Sym}(T)_{i_1 i_2 \ldots i_k} = \frac{1}{k!} \sum_{\pi \in S_k} T_{i_{\pi(1)} i_{\pi(2)} \ldots i_{\pi(k)}}.$$

- S_k is the symmetric group on k indices.
- This ensures the result is symmetric, even if the original tensor was not.

Example: Symmetrizing a 2nd-Order Tensor

Given a matrix A, its symmetrization is:

$$\mathrm{Sym}(A) = \frac{1}{2}(A + A^\top).$$

This extracts the symmetric part of a matrix.

Properties

- Symmetrization is a projection operator: applying it twice gives the same result.
- The space of symmetric tensors is a subspace of the full tensor space.
- Symmetric tensors are often much smaller in dimension than general tensors.

Why This Matters

- Symmetrization leads to symmetric tensor spaces, central in polynomial algebra and statistics.
- It prepares us for the study of alternating tensors (exterior algebra) in the next part.
- Many applications (covariance, stress, Hessians) naturally produce symmetric tensors.

Exercises

1. Matrix Symmetrization: Compute the symmetrized version of

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

2. Check Symmetry: Is the tensor T_{ij} with entries $T_{12} = 1, T_{21} = 2, T_{11} = 0, T_{22} = 3$ symmetric?
3. Symmetrizing a 3rd-Order Tensor: Write the formula for the symmetrized version of a 3rd-order tensor T_{ijk}.
4. Projection Property: Show that if T is already symmetric, then $\mathrm{Sym}(T) = T$.
5. Thought Experiment: Why might symmetrization be important in statistics (e.g., covariance estimation) or in physics (e.g., stress tensors)?

7.2 Alternation (Antisymmetrization)

If symmetrization enforces invariance under index swaps, alternation (or antisymmetrization) enforces the opposite: swapping two indices flips the sign. This construction is central to exterior algebra and geometric concepts like orientation and volume.

Antisymmetric Tensors

A tensor T_{ij} is antisymmetric if

$$T_{ij} = -T_{ji}.$$

- In particular, $T_{ii} = 0$.
- For higher orders:

$$T_{i_{\pi(1)} i_{\pi(2)} \dots i_{\pi(k)}} = \mathrm{sgn}(\pi)\, T_{i_1 i_2 \dots i_k},$$

  where $\mathrm{sgn}(\pi)$ is the sign of the permutation π.

Examples:

- The determinant is based on a fully antisymmetric tensor.
- The cross product in $\mathbb{R}^3$ can be expressed using an antisymmetric tensor.

Alternation Operator

Given any tensor T, its alternated version is obtained by summing with signs:

$$\mathrm{Alt}(T)_{i_1 i_2 \ldots i_k} = \frac{1}{k!} \sum_{\pi \in S_k} \mathrm{sgn}(\pi)\, T_{i_{\pi(1)} i_{\pi(2)} \cdots i_{\pi(k)}}.$$

- This forces antisymmetry.
- If T was already antisymmetric, $\mathrm{Alt}(T) = T$.

Example: Alternating a 2nd-Order Tensor

For a matrix A, alternation gives:

$$\mathrm{Alt}(A) = \frac{1}{2}(A - A^\top).$$

This extracts the skew-symmetric part of a matrix.

Properties

- Alternation is a projection operator: $\mathrm{Alt}(\mathrm{Alt}(T)) = \mathrm{Alt}(T)$.
- The space of antisymmetric k-tensors is called $\Lambda^k(V)$, the exterior power.
- Its dimension is $\binom{n}{k}$, much smaller than the full tensor space.

Why This Matters

- Alternating tensors represent oriented areas, volumes, and higher-dimensional analogues.
- They are the foundation of differential forms and exterior algebra.
- Physics applications: angular momentum, electromagnetism, determinants.

Exercises

1. Skew-Symmetric Matrix: Compute the alternated version of

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

2. Check Antisymmetry: Is the tensor T_{ij} with $T_{12} = 1, T_{21} = -1, T_{11} = 0, T_{22} = 0$ antisymmetric?

3. Alternating a 3rd-Order Tensor: Write the formula for $\mathrm{Alt}(T)_{ijk}$ explicitly in terms of the six permutations of (i, j, k).
4. Zero Property: Show that if two indices are equal in an antisymmetric tensor, the component must vanish.
5. Thought Experiment: Why is antisymmetry essential for defining oriented volume (via determinants) and exterior algebra?

7.3 Decompositions by Symmetry Type

So far we've seen how to symmetrize and antisymmetrize tensors. The next step is to notice that *any- tensor can be decomposed into symmetric and antisymmetric parts. This is very similar to how every matrix can be written as the sum of a symmetric and a skew-symmetric matrix.

The 2nd-Order Case (Matrices)

Given a matrix A:

$$A = \tfrac{1}{2}(A + A^\top) + \tfrac{1}{2}(A - A^\top).$$

- The first term is symmetric.
- The second term is antisymmetric.
- Together, they reconstruct the original matrix.

This is the simplest case of decomposition by symmetry type.

Higher-Order Generalization

For a tensor $T_{i_1 \dots i_k}$:

- Apply the symmetrization operator to get the symmetric part.
- Apply the alternation operator to get the antisymmetric part.

But for $k > 2$, there are many possible symmetry types (not just symmetric vs. antisymmetric).

Young Diagrams and Symmetry Types (Preview)

In general, the ways indices can be symmetrized/antisymmetrized correspond to representations of the symmetric group.

- Fully symmetric: indices invariant under any swap.
- Fully antisymmetric: indices change sign under swaps.
- Mixed types: some indices symmetric among themselves, others antisymmetric.

These patterns can be visualized with Young diagrams (a deeper subject in representation theory).

Why Decomposition Matters

- Symmetry often reduces the effective dimension of tensor spaces.
- Many natural tensors fall into specific symmetry classes (e.g., metric tensors are symmetric, differential forms are antisymmetric).
- In applications, splitting into symmetric/antisymmetric parts clarifies the geometric meaning:
  - Symmetric part → stretching, scaling (like stress/strain).
  - Antisymmetric part → rotation, orientation (like angular momentum).

Exercises

1. Matrix Decomposition: Decompose

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

   into symmetric and antisymmetric parts.

2. Sym + Anti Check: Verify that every 2×2 matrix can be written uniquely as $S + K$, with S symmetric and K antisymmetric.
3. Symmetric Projection: For T_{ijk}, write the formula for its symmetric part $\mathrm{Sym}(T)_{ijk}$.
4. Antisymmetric Projection: For T_{ijk}, write the formula for its antisymmetric part $\mathrm{Alt}(T)_{ijk}$.
5. Thought Experiment: Why does decomposing tensors by symmetry type help in physics and engineering (e.g., stress tensors, electromagnetic fields)? ### 7.4 Applications: Determinants and Oriented Volumes

Now we put the ideas of symmetry and antisymmetry into practice. Some of the most important geometric and physical quantities are built from alternating tensors - especially the determinant and oriented volumes.

Determinant as an Antisymmetric Multilinear Map

The determinant of an $n \times n$ matrix can be seen as an alternating n-linear form:

$$\det(v_1, v_2, \dots, v_n),$$

where v_i are the column vectors of the matrix.

- Multilinear: linear in each column.
- Alternating: if two columns are equal, the determinant vanishes.
- Geometric meaning: signed volume of the parallelepiped spanned by the columns.

Oriented Areas in $\mathbb{R}^2$

Given two vectors $u, v \in \mathbb{R}^2$, the area of the parallelogram they span is:

$$\text{Area}(u, v) = |\det(u, v)|.$$

The sign of $\det(u, v)$ indicates orientation (clockwise vs. counterclockwise).

Oriented Volumes in $\mathbb{R}^3$

Given three vectors $u, v, w \in \mathbb{R}^3$, the scalar triple product

$$[u, v, w] = u \cdot (v \times w)$$

is the determinant of the 3×3 matrix with columns u, v, w.

- Magnitude: volume of the parallelepiped.
- Sign: orientation (right-handed vs. left-handed system).

Exterior Powers and General Volumes

The antisymmetric space $\Lambda^k(V)$ encodes oriented k-dimensional volumes:

- For $k = 2$: oriented areas (parallelograms).
- For $k = 3$: oriented volumes (parallelepipeds).
- For general k: higher-dimensional "hyper-volumes."

Thus, alternating tensors generalize determinants and orientation to any dimension.

Why This Matters

- Determinants, areas, and volumes are not just numbers - they are built from antisymmetric tensors.
- This viewpoint makes clear why determinants vanish when vectors are linearly dependent: antisymmetry forces collapse.
- Orientation is encoded algebraically, which is crucial in physics (torques, flux) and geometry (orientation of manifolds).

Exercises

1. 2D Determinant: Compute $\det\big((1,2),(3,4)\big)$. Interpret the result geometrically.
2. Area from Determinant: Find the area of the parallelogram spanned by $u = (2,0)$ and $v = (1,3)$ using determinants.
3. Triple Product: Compute $[u,v,w]$ for $u = (1,0,0)$, $v = (0,2,0)$, $w = (0,0,3)$. What volume does this represent?
4. Linear Dependence: Show that if v_1, v_2, v_3 are linearly dependent in $\mathbb{R}^3$, then $[v_1, v_2, v_3] = 0$.
5. Thought Experiment: Why does antisymmetry naturally encode the geometric idea of orientation? How does this link back to the sign of a determinant?

Part IV. Exterior Algebra

Chapter 8. Wedge Products and k-Vectors

8.1 The Wedge Product and Geometric Meaning

We now begin Part IV: Exterior Algebra, which formalizes antisymmetric tensors into a powerful algebraic system. The key operation here is the wedge product ($\wedge$).

Definition of the Wedge Product

Given two vectors $u, v \in V$, their wedge product is:

$$u \wedge v = u \otimes v - v \otimes u.$$

- It is bilinear: linear in each input.
- It is antisymmetric: $u \wedge v = -v \wedge u$.
- It vanishes if u and v are linearly dependent.

Geometric Meaning

- $u \wedge v$ represents the oriented parallelogram spanned by u and v.
- The magnitude corresponds to the area:

  $$\|u \wedge v\| = \|u\| \, \|v\| \sin\theta,$$

  where θ is the angle between u and v.
- The sign encodes orientation.

Higher Wedge Products

For k vectors $u_1, u_2, \ldots, u_k$:

$$u_1 \wedge u_2 \wedge \cdots \wedge u_k$$

represents the oriented k-dimensional volume element (parallelepiped).

- Antisymmetry ensures the wedge vanishes if the vectors are linearly dependent.
- Thus, wedge products capture the idea of dimension of span:
  - Two vectors $\rightarrow$ area.
  - Three vectors $\rightarrow$ volume.
  - More vectors $\rightarrow$ higher-dimensional volume.

Algebraic Properties

1. Anticommutativity: $u \wedge v = -v \wedge u$.
2. Associativity (up to sign): $(u \wedge v) \wedge w = u \wedge (v \wedge w)$.
3. Alternating Property: $u \wedge u = 0$.
4. Distributivity: $(u + v) \wedge w = u \wedge w + v \wedge w$.

Together, these rules define the exterior algebra $\Lambda(V)$.

Why This Matters

- The wedge product is the algebraic foundation of determinants, areas, and volumes.
- It provides a coordinate-free way to work with oriented geometry.
- In advanced topics, wedge products lead directly to differential forms and integration on manifolds.

Exercises

1. Basic Wedge: Compute $(1, 0) \wedge (0, 1)$ in $\mathbb{R}^2$. What does it represent?
2. Antisymmetry Check: Show that $(1, 2, 3) \wedge (4, 5, 6) = -(4, 5, 6) \wedge (1, 2, 3)$.
3. Area via Wedge: Use wedge products to find the area of the parallelogram spanned by $u = (2, 0)$ and $v = (1, 3)$.
4. Zero Wedge: Show that $u \wedge v = 0$ when $u = (1, 2)$, $v = (2, 4)$. Interpret geometrically.

5. Thought Experiment: Why does antisymmetry make wedge products vanish for linearly dependent vectors? How does this encode the idea of dimension reduction?

8.2 Areas, Volumes, Orientation, Determinant

Now that we've introduced the wedge product, we can connect it directly to geometry: areas, volumes, and orientation. The wedge product is the algebraic language for these concepts.

2D: Areas from the Wedge Product

For vectors $u, v \in \mathbb{R}^2$, the wedge product $u \wedge v$ represents the oriented area of the parallelogram they span.

- Algebraically:

$$u \wedge v = \det \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix}.$$

- Geometric meaning: $|u \wedge v|$ is the area, the sign encodes clockwise vs. counterclockwise orientation.

3D: Volumes via Triple Wedge

For $u, v, w \in \mathbb{R}^3$:

$$u \wedge v \wedge w$$

represents the oriented volume of the parallelepiped spanned by them.

- Algebraically:

$$u \wedge v \wedge w = \det \begin{bmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{bmatrix}.$$

- Geometric meaning: $|u \wedge v \wedge w|$ is the volume, the sign encodes right- vs. left-handed orientation.

General n-Dimensional Case

For n vectors $v_1, \ldots, v_n \in \mathbb{R}^n$:

$$v_1 \wedge v_2 \wedge \cdots \wedge v_n = \det(v_1, v_2, \ldots, v_n),$$

the determinant of the matrix with those vectors as columns.

- Magnitude: n-dimensional volume of the parallelepiped.
- Sign: orientation (positive if vectors preserve orientation, negative if they reverse it).

Orientation

- Orientation is the choice of "handedness" (right-hand vs. left-hand rule).
- The wedge product encodes orientation automatically via antisymmetry.
- Swapping two vectors flips the sign, meaning the orientation is reversed.

Why This Matters

- Wedge products generalize determinants to all dimensions.
- They provide a uniform way to compute areas, volumes, and higher-dimensional volumes.
- Orientation, crucial in geometry and physics, is naturally handled through antisymmetry.

Exercises

1. Area in 2D: Compute the oriented area of the parallelogram spanned by $u = (1, 2)$, $v = (3, 4)$ using $u \wedge v$.
2. Volume in 3D: Compute $u \wedge v \wedge w$ for $u = (1, 0, 0), v = (0, 2, 0), w = (0, 0, 3)$. Interpret geometrically.
3. Orientation Flip: Show that $u \wedge v = -v \wedge u$ changes the orientation of the area.
4. Determinant Link: Verify that $u \wedge v \wedge w$ in $\mathbb{R}^3$ equals $\det[u, v, w]$.
5. Thought Experiment: Why does the wedge product vanish if the set of vectors is linearly dependent? What does this mean geometrically for areas/volumes?

8.3 Basis, Dimension, Grassmann Algebra

We've seen that wedge products describe oriented areas, volumes, and higher-dimensional analogues. To make this systematic, we construct exterior powers of a vector space, also known as Grassmann algebras.

Exterior Powers $\Lambda^k(V)$

For a vector space V:

- $\Lambda^k(V)$ is the space of all antisymmetric k-tensors (wedge products of k vectors).
- $\Lambda^0(V) \cong \mathbb{R}$ (scalars).
- $\Lambda^1(V) \cong V$ (vectors).
- $\Lambda^2(V)$: oriented areas.
- $\Lambda^3(V)$: oriented volumes.
- In general, $\Lambda^k(V)$ encodes oriented k-dimensional "volume elements."

Basis of $\Lambda^k(V)$

Suppose V has dimension n with basis $e_1, e_2, \dots, e_n$.

- A basis for $\Lambda^k(V)$ is given by wedge products:

$$e_{i_1} \wedge e_{i_2} \wedge \cdots \wedge e_{i_k}, \quad i_1 < i_2 < \cdots < i_k.$$

- These are linearly independent and span all of $\Lambda^k(V)$.

Dimension Formula

The dimension of $\Lambda^k(V)$ is:

$$\dim \Lambda^k(V) = \binom{n}{k}.$$

- Example: If $n = 4$, then $\dim \Lambda^2(V) = \binom{4}{2} = 6$.
- This matches the number of independent ways to choose k basis vectors from n.

Grassmann (Exterior) Algebra

The direct sum of all exterior powers:

$$\Lambda(V) = \Lambda^0(V) \oplus \Lambda^1(V) \oplus \cdots \oplus \Lambda^n(V),$$

is called the exterior algebra (or Grassmann algebra).

- Multiplication in this algebra is given by the wedge product.
- Anticommutativity ensures $u \wedge u = 0$.
- This makes $\Lambda(V)$ into a graded algebra, organized by degree.

Why This Matters

- Exterior algebra gives a systematic framework for working with antisymmetric tensors.
- The dimension formula explains why areas, volumes, etc. fit neatly into subspaces.
- Grassmann algebra underlies modern geometry, topology, and physics (differential forms, integration on manifolds).

Exercises

1. Basis in Λ^2: For $V = \mathbb{R}^3$ with basis (e_1, e_2, e_3), list a basis of $\Lambda^2(V)$.
2. Dimension Count: If $\dim V = 5$, compute $\dim \Lambda^2(V)$, $\dim \Lambda^3(V)$, and $\dim \Lambda^4(V)$.
3. Simple Wedge Expansion: In $\mathbb{R}^3$, expand $(e_1 + e_2) \wedge e_3$.
4. Grassmann Algebra Structure: Identify which degrees (k) of $\Lambda^k(V)$ correspond to:
   - scalars,
   - vectors,
   - oriented areas,
   - oriented volumes.
5. Thought Experiment: Why does the formula $\dim \Lambda^k(V) = \binom{n}{k}$ match the combinatorial idea of "choosing k directions from n"?

8.4 Differential Forms (Gentle Preview)

We close Part IV with a glimpse of how wedge products extend beyond linear algebra into calculus on manifolds. This leads to differential forms, the language of modern geometry and physics.

Differential 1-Forms

- A 1-form is a covector field: at each point, it eats a vector and gives a number.
- Example in $\mathbb{R}^2$:

$$\omega = x\,dy - y\,dx,$$

  where dx, dy are basis 1-forms.

Higher-Degree Forms

- A k-form is an antisymmetric multilinear map that takes k vectors (directions) and outputs a scalar.
- Built using wedge products of 1-forms:
  - $dx \wedge dy$ is a 2-form (measures oriented area in the xy-plane).
  - $dx \wedge dy \wedge dz$ is a 3-form (measures oriented volume in $\mathbb{R}^3$).

Exterior Derivative d

Differential forms come with a derivative operator d:

- Takes a k-form to a $(k+1)$-form.
- Generalizes gradient, curl, and divergence in one unified framework.
- Example:
  - For $f(x, y)$ a scalar function (0-form):

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy,$$

    which is the gradient written as a 1-form.

Integration of Forms

- Differential forms can be integrated over curves, surfaces, and higher-dimensional manifolds.
- Example: integrating a 1-form along a curve gives the work done by a force field.
- Integrating a 2-form over a surface gives the flux through the surface.

Why This Matters

- Differential forms unify multivariable calculus (gradient, curl, divergence) into one elegant theory.
- They are the foundation of Stokes' theorem, which generalizes all the fundamental theorems of calculus.
- Physics: electromagnetism, fluid dynamics, and general relativity all use differential forms.

Exercises

1. 1-Form Action: Let $\omega = 3dx + 2dy$. Evaluate $\omega(v)$ for $v = (1, 4)$.
2. Wedge of 1-Forms: Compute $dx \wedge dy(v, w)$ for $v = (1, 0), w = (0, 2)$.
3. Exterior Derivative: For $f(x, y) = x^2 y$, compute df.
4. Integration Example: Interpret $\int_\gamma (y\,dx)$ along a path γ in $\mathbb{R}^2$. What does it represent physically?
5. Thought Experiment: How does the wedge product in linear algebra naturally lead into differential forms in calculus?

Chapter 9. Hodge Dual and Metrics (Optional but Useful)

9.1 Inner Products on Exterior Powers

To use wedge products effectively, we need a way to measure their size and compare them. This is where inner products on exterior powers come in. They extend the familiar dot product on vectors to areas, volumes, and higher-dimensional elements.

Induced Inner Product

Suppose V has an inner product $\langle \cdot, \cdot \rangle$. We can extend it to $\Lambda^k(V)$ by declaring wedge products of basis vectors to be orthonormal:

$$\langle e_{i_1} \wedge \cdots \wedge e_{i_k},\, e_{j_1} \wedge \cdots \wedge e_{j_k} \rangle = \begin{cases} \det(\delta_{i_a j_b}) & \text{if sets are equal,} \\ 0 & \text{otherwise.} \end{cases}$$

In simpler words:

- Take all ordered k-tuples of basis vectors.
- The wedge products of these are orthonormal if the underlying sets match.

Example in $\mathbb{R}^2$

Let e_1, e_2 be an orthonormal basis. Then

$$\langle e_1 \wedge e_2, e_1 \wedge e_2 \rangle = 1.$$

So $e_1 \wedge e_2$ has unit length, representing a unit square area.

Example in $\mathbb{R}^3$

Let e_1, e_2, e_3 be orthonormal. Then

- $e_1 \wedge e_2, e_1 \wedge e_3, e_2 \wedge e_3$ form an orthonormal basis of $\Lambda^2(\mathbb{R}^3)$.
- $e_1 \wedge e_2 \wedge e_3$ is the unit element of $\Lambda^3(\mathbb{R}^3)$, representing a unit cube volume.

Norms of Wedge Products

For $u, v \in V$:

$$\|u \wedge v\|^2 = \|u\|^2 \|v\|^2 - \langle u, v \rangle^2.$$

This is the formula for the area of a parallelogram spanned by u, v.

- If u, v are orthogonal: area = product of lengths.
- If u, v are parallel: area = 0.

For higher k, the squared norm of $u_1 \wedge \cdots \wedge u_k$ equals the determinant of the Gram matrix $[\langle u_i, u_j \rangle]$.

Why This Matters

- Inner products on exterior powers connect wedge products to geometry quantitatively (areas, volumes).
- They unify length, area, and volume measurement into one consistent framework.
- This lays the groundwork for the Hodge star operator, which depends on having an inner product.

Exercises

1. Area Formula: Compute $\|u \wedge v\|$ for $u = (1, 0), v = (1, 1)$ in $\mathbb{R}^2$.
2. Gram Determinant: Show that $\|u \wedge v\|^2 = \det \begin{bmatrix} \langle u, u \rangle & \langle u, v \rangle \\ \langle v, u \rangle & \langle v, v \rangle \end{bmatrix}$.
3. 3D Basis Check: Verify that $e_1 \wedge e_2, e_1 \wedge e_3, e_2 \wedge e_3$ are orthonormal in $\Lambda^2(\mathbb{R}^3)$.
4. Volume Norm: Compute the norm of $e_1 \wedge e_2 \wedge (e_1 + e_3)$. Interpret geometrically.
5. Thought Experiment: Why does the Gram determinant naturally appear in wedge product norms? How does this connect to the idea of measuring volume via linear independence?

9.2 Hodge Star, Pseudo-Vectors, Cross Products

With an inner product defined on exterior powers, we can now introduce the Hodge star operator ($\star$), which creates a bridge between k-forms and $(n-k)$-forms. This operator encodes geometric duality, and in 3D it gives rise to the familiar cross product.

The Hodge Star Operator

Let V be an n-dimensional inner product space with an orientation. The Hodge star is a linear map:

$$\star : \Lambda^k(V) \to \Lambda^{n-k}(V).$$

It is defined so that for all $\alpha, \beta \in \Lambda^k(V)$:

$$\alpha \wedge \star\beta = \langle \alpha, \beta \rangle \, \text{vol},$$

where vol is the unit volume element in $\Lambda^n(V)$.

Examples in $\mathbb{R}^3$

With orthonormal basis e_1, e_2, e_3:

- $\star e_1 = e_2 \wedge e_3$.
- $\star(e_1 \wedge e_2) = e_3$.
- $\star(e_1 \wedge e_2 \wedge e_3) = 1$.

So the star turns 1-vectors into 2-forms, 2-forms into 1-vectors, and the 3-form into a scalar.

Cross Product as Hodge Dual

In $\mathbb{R}^3$, the cross product $u \times v$ can be expressed using the Hodge star:

$$u \times v = \star(u \wedge v).$$

- $u \wedge v$ represents the oriented area spanned by u, v.
- The Hodge star converts that 2-form into the unique vector perpendicular to the plane, with magnitude equal to the area.

Thus, the cross product is not a primitive concept - it is the Hodge star of a wedge product.

Pseudo-Vectors

- Objects like angular momentum or magnetic field transform like vectors, but with an orientation twist.
- These are pseudo-vectors, naturally arising from antisymmetric 2-forms in $\mathbb{R}^3$.
- The Hodge star provides the link between antisymmetric 2-forms and pseudo-vectors.

Why This Matters

- The Hodge star encodes geometric duality between subspaces and their complements.
- It explains the origin of the cross product in 3D and why it does not generalize to higher dimensions.
- It sets the stage for physics applications: electromagnetism, fluid dynamics, and general relativity all use the Hodge dual.

Exercises

1. Basic Star: In $\mathbb{R}^3$, compute $\star(e_1)$, $\star(e_2)$, and $\star(e_3)$.
2. Star on 2-Forms: Verify that $\star(e_1 \wedge e_2) = e_3$ in $\mathbb{R}^3$.
3. Cross Product via Star: Compute $u \times v$ using $u \times v = \star(u \wedge v)$ for $u = (1, 0, 0), v = (0, 1, 0)$.
4. Star Squared: Show that in $\mathbb{R}^3$, applying the star twice gives $\star \star \alpha = \alpha$ for 1-forms.
5. Thought Experiment: Why does the cross product only exist in $\mathbb{R}^3$ (and in a special sense in $\mathbb{R}^7$)? How does the Hodge star viewpoint clarify this?

9.3 Volume Forms and Integration Glimpses

We now tie together the Hodge star, wedge products, and inner products to introduce the concept of volume forms - the mathematical tools that let us integrate over spaces of any dimension.

Volume Form

In an n-dimensional oriented inner product space V, the volume form is the unit n-form:

$$\text{vol} = e_1 \wedge e_2 \wedge \cdots \wedge e_n,$$

where $\{e_i\}$ is an orthonormal positively oriented basis.

- It represents the "unit n-dimensional volume element."
- Any n vectors $v_1, \ldots, v_n$ give

  $$\text{vol}(v_1, \ldots, v_n) = \det[v_1 \ \cdots \ v_n],$$

  i.e. their oriented volume.

Volume via Hodge Star

For a k-form α, its dual $\star\alpha$ is an $(n-k)$-form.

- The wedge $\alpha \wedge \star\alpha$ is proportional to vol.
- This expresses how much "volume" α contributes in n-dimensions.

Example in $\mathbb{R}^3$:

- If $\alpha = u \wedge v$, then

  $$\alpha \wedge \star\alpha = \|u \wedge v\|^2 \,\text{vol},$$

  encoding the squared area of the parallelogram spanned by u, v.

Integration of Differential Forms (Glimpse)

The volume form allows us to integrate over manifolds:

- In $\mathbb{R}^n$, integrating vol over a region gives its volume.
- A k-form can be integrated over a k-dimensional surface (curve, area, etc.).
- Example:
  - In $\mathbb{R}^2$, $\text{vol} = dx \wedge dy$.
  - The integral $\int_\Omega dx \wedge dy$ is the area of region Ω.

This is the seed of Stokes' theorem, which unifies the fundamental theorem of calculus, Green's theorem, and the divergence theorem.

Why This Matters

- Volume forms make precise the link between algebra (determinants) and geometry (volume).
- They are the foundation for integration on curved spaces (manifolds).
- In physics, volume forms appear in conservation laws, flux integrals, and field theories.

Exercises

1. 2D Volume Form: Show that for $u = (1, 0), v = (0, 2)$ in $\mathbb{R}^2$: $\text{vol}(u, v) = \det \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.
2. 3D Example: Compute $\text{vol}(u, v, w)$ for $u = (1, 0, 0), v = (0, 1, 0), w = (0, 0, 2)$.
3. Star Relation: In $\mathbb{R}^3$, verify that $(u \wedge v) \wedge \star(u \wedge v) = \|u \wedge v\|^2 \,\text{vol}$.
4. Integration Preview: Evaluate $\int_{[0,1]\times[0,1]} dx \wedge dy$. Interpret the result.
5. Thought Experiment: Why is orientation essential when defining a volume form? What would go wrong in integration if we ignored orientation?

Part V. Symmetric Tensors and Polynomial View

Chapter 10. Symmetric Powers and Homogeneous Polynomials

10.1 Symmetric Tensors as Polynomial Coefficients

So far, we have studied antisymmetric tensors and exterior algebra. Now we turn to the opposite case: symmetric tensors. These are equally important, and they connect directly to polynomials.

Symmetric Tensors

A symmetric tensor of order k satisfies:

$$T_{i_1 i_2 \dots i_k} = T_{i_{\pi(1)} i_{\pi(2)} \dots i_{\pi(k)}}$$

for any permutation π.

Example:

- A quadratic form $Q(x) = x^\top A x$ comes from a symmetric 2-tensor.
- Higher-order symmetric tensors generalize this to cubic, quartic, etc.

From Symmetric Tensors to Polynomials

Every symmetric k-tensor corresponds to a homogeneous polynomial of degree k.

- Given a symmetric tensor $T_{i_1 \dots i_k}$:

$$p(x) = T_{i_1 \dots i_k} x^{i_1} \cdots x^{i_k}.$$

- Example: In $\mathbb{R}^2$, let $T_{11} = 1, T_{12} = T_{21} = 2, T_{22} = 3$. Then

$$p(x, y) = x^2 + 4xy + 3y^2.$$

This is a homogeneous polynomial of degree 2.

From Polynomials to Symmetric Tensors

Conversely, every homogeneous polynomial defines a symmetric tensor:

- Example: $p(x, y, z) = 2x^2z + 3yz^2$.
- Its coefficients directly correspond to entries of a symmetric 3-tensor.

Thus, symmetric tensors and homogeneous polynomials are two sides of the same coin.

Why This Matters

- This correspondence makes symmetric tensors central in algebraic geometry and statistics.
- Moments of probability distributions are symmetric tensors.
- Polynomial optimization problems can be rephrased in terms of symmetric tensors.

Exercises

1. Quadratic Form: Show that the polynomial $p(x, y) = 3x^2 + 2xy + y^2$ corresponds to a symmetric 2-tensor. Write its matrix.
2. Cubic Example: Write the symmetric 3-tensor for $p(x, y) = x^3 + 3x^2y + 2y^3$.
3. Backwards: Given the symmetric tensor with entries $T_{111} = 1, T_{112} = 2, T_{122} = 3, T_{222} = 4$, write the polynomial in two variables.
4. Homogeneity Check: Explain why the polynomial $x^2 + xy + y^2$ is homogeneous of degree 2, but $x^2 + y + 1$ is not.
5. Thought Experiment: Why might it be useful to switch between tensor and polynomial viewpoints in machine learning or physics? ### 10.2 Polarization Identities

We saw that symmetric tensors and homogeneous polynomials are two sides of the same coin. But how exactly do we go from one to the other in a precise way? The tool for this is the polarization identity, which allows us to reconstruct a symmetric multilinear map (or tensor) from its associated polynomial.

From Symmetric Tensor to Polynomial

If T is a symmetric k-linear form on V, we define the polynomial

$$p(x) = T(x, x, \dots, x).$$

This is a homogeneous polynomial of degree k.

Example:

- If $T(x, y) = \langle x, y \rangle$, then $p(x) = \langle x, x \rangle = \|x\|^2$.

From Polynomial to Symmetric Tensor

Given a homogeneous polynomial $p(x)$ of degree k, we want to recover the symmetric tensor T.

The polarization identity says:

$$T(x_1, \dots, x_k) = \frac{1}{k!\, 2^k} \sum_{\epsilon_1, \dots, \epsilon_k = \pm 1} \epsilon_1 \cdots \epsilon_k \, p(\epsilon_1 x_1 + \cdots + \epsilon_k x_k).$$

This formula extracts the multilinear coefficients hidden inside the polynomial.

Example: Quadratic Forms

For $p(x) = \|x\|^2$:

- Using polarization, we recover the bilinear form:

$$T(x, y) = \tfrac{1}{2}(\|x + y\|^2 - \|x\|^2 - \|y\|^2) = \langle x, y \rangle.$$

This is the classical polarization identity for inner products.

Why This Matters

- The polarization identity ensures that the correspondence between symmetric tensors and homogeneous polynomials is exact and invertible.
- It provides a way to compute multilinear structure from polynomial data.
- In applications:
  - In physics, energy polynomials give rise to symmetric stress tensors.
  - In statistics, cumulants and moments correspond to symmetric tensors recovered from generating functions.

Exercises

1. Inner Product Recovery: Use the quadratic polarization identity to show that

$$\langle x, y\rangle = \tfrac{1}{4}(\|x+y\|^2 - \|x-y\|^2).$$

2. Cubic Case: Let $p(x) = x_1^3$ in $\mathbb{R}^2$. Use the polarization formula to compute $T(e_1, e_1, e_1)$ and $T(e_1, e_1, e_2)$.
3. Verification: For $p(x, y) = x^2 + y^2$, verify that the polarization identity recovers the symmetric bilinear form $T(x, y) = x_1y_1 + x_2y_2$.
4. Homogeneity Check: Show why polarization fails if $p(x)$ is not homogeneous.
5. Thought Experiment: Why is it important that the tensor be symmetric in order for the polynomial tensor correspondence to work smoothly?

10.2 Polarization Identities

We saw that symmetric tensors and homogeneous polynomials are two sides of the same coin. But how exactly do we go from one to the other in a precise way? The tool for this is the polarization identity, which allows us to reconstruct a symmetric multilinear map (or tensor) from its associated polynomial.

From Symmetric Tensor to Polynomial

If T is a symmetric k-linear form on V, we define the polynomial

$$p(x) = T(x, x, \dots, x).$$

This is a homogeneous polynomial of degree k.

Example:

- If $T(x, y) = \langle x, y \rangle$, then $p(x) = \langle x, x \rangle = \|x\|^2$.

From Polynomial to Symmetric Tensor

Given a homogeneous polynomial $p(x)$ of degree k, we want to recover the symmetric tensor T.

The polarization identity says:

$$T(x_1, \dots, x_k) = \frac{1}{k!\, 2^k} \sum_{\epsilon_1, \dots, \epsilon_k = \pm 1} \epsilon_1 \cdots \epsilon_k \, p(\epsilon_1 x_1 + \cdots + \epsilon_k x_k).$$

This formula extracts the multilinear coefficients hidden inside the polynomial.

Example: Quadratic Forms

For $p(x) = \|x\|^2$:

- Using polarization, we recover the bilinear form:

$$T(x, y) = \tfrac{1}{2}(\|x + y\|^2 - \|x\|^2 - \|y\|^2) = \langle x, y \rangle.$$

This is the classical polarization identity for inner products.

Why This Matters

- The polarization identity ensures that the correspondence between symmetric tensors and homogeneous polynomials is exact and invertible.
- It provides a way to compute multilinear structure from polynomial data.
- In applications:
  - In physics, energy polynomials give rise to symmetric stress tensors.
  - In statistics, cumulants and moments correspond to symmetric tensors recovered from generating functions.

Exercises

1. Inner Product Recovery: Use the quadratic polarization identity to show that

$$\langle x, y\rangle = \tfrac{1}{4}(\|x+y\|^2 - \|x-y\|^2).$$

2. Cubic Case: Let $p(x) = x_1^3$ in $\mathbb{R}^2$. Use the polarization formula to compute $T(e_1, e_1, e_1)$ and $T(e_1, e_1, e_2)$.
3. Verification: For $p(x, y) = x^2 + y^2$, verify that the polarization identity recovers the symmetric bilinear form $T(x, y) = x_1y_1 + x_2y_2$.
4. Homogeneity Check: Show why polarization fails if $p(x)$ is not homogeneous.
5. Thought Experiment: Why is it important that the tensor be symmetric in order for the polynomial tensor correspondence to work smoothly?

10.3 Moments and Cumulants

Symmetric tensors are not just abstract objects - they naturally appear in probability and statistics. Two key concepts, moments and cumulants, can be represented as symmetric tensors, providing a multilinear view of random variables and their dependencies.

Moments as Symmetric Tensors

For a random vector $X \in \mathbb{R}^n$, the k-th moment tensor is:

$$M^{(k)} = \mathbb{E}[X^{\otimes k}],$$

where

$$X^{\otimes k} = X \otimes X \otimes \cdots \otimes X \quad (k \text{ times}).$$

- Entries:

$$(M^{(k)})_{i_1 i_2 \ldots i_k} = \mathbb{E}[X_{i_1} X_{i_2} \cdots X_{i_k}].$$

- Since multiplication is commutative, $M^{(k)}$ is a symmetric tensor.

Examples:

- $M^{(1)}$ = mean vector.
- $M^{(2)}$ = second moment matrix (related to covariance).
- Higher-order moments capture skewness, kurtosis, etc.

Cumulants as Symmetric Tensors

Cumulants refine moments by removing redundancy:

- Defined via the log of the moment-generating function.
- The k-th cumulant tensor $C^{(k)}$ is also symmetric.
- Example:
  - $C^{(1)}$ = mean.
  - $C^{(2)}$ = covariance matrix.
  - $C^{(3)}$ = skewness tensor.
  - $C^{(4)}$ = kurtosis tensor.

Why Tensors?

- Moments and cumulants encode multi-way dependencies between variables.
- As tensors, they can be studied with linear algebra and decompositions.
- Applications:
  - Signal processing: blind source separation via cumulant tensors.
  - Statistics: identifying distributions via moment tensors.
  - Machine learning: feature extraction from higher-order statistics.

Connection to Symmetric Polynomials

- Moments correspond to coefficients of homogeneous polynomials (moment generating functions).
- Polarization identities let us move between the polynomial representation and the tensor form.

Exercises

1. Second Moment: Let $X = (X_1, X_2)$ with $\mathbb{E}[X_1^2] = 2, \mathbb{E}[X_1X_2] = 1, \mathbb{E}[X_2^2] = 3$. Write the 2nd moment tensor $M^{(2)}$.
2. Symmetry Check: Show that $(M^{(3)})_{ijk} = \mathbb{E}[X_iX_jX_k]$ is symmetric in all indices.
3. Covariance as Cumulant: Explain why the covariance matrix is the 2nd cumulant tensor $C^{(2)}$.
4. Higher Cumulants: What does $C^{(3)}$ measure that is not captured by $C^{(2)}$?
5. Thought Experiment: Why might cumulants be more useful than moments in separating independent signals or detecting non-Gaussianity?

10.4 Low-Rank Symmetric Decompositions

Symmetric tensors, like general tensors, can be decomposed into simpler parts. In the symmetric case, this often means writing a symmetric tensor as a sum of outer products of vectors with themselves. These low-rank symmetric decompositions play a central role in data science, signal processing, and machine learning.

Symmetric Rank

For a symmetric tensor $T \in \mathrm{Sym}^k(V)$, the symmetric rank is the smallest r such that

$$T = \sum_{i=1}^{r} \lambda_i \, v_i^{\otimes k},$$

where $v_i \in V$ and $\lambda_i \in \mathbb{R}$.

- Each term $v_i^{\otimes k} = v_i \otimes v_i \otimes \cdots \otimes v_i$ (k times) is a simple symmetric tensor.
- This is the symmetric analogue of the rank-one decomposition for matrices.

Example: Quadratic Forms

A quadratic form (symmetric 2-tensor) can be decomposed as:

$$Q(x) = \sum_{i=1}^{r} \lambda_i (v_i^\top x)^2.$$

This is the spectral decomposition of a symmetric matrix.

Higher-Order Case

For cubic or quartic symmetric tensors:

- Example in 3rd order:

$$T(x, y, z) = \sum_{i=1}^{r} \lambda_i (v_i^\top x)(v_i^\top y)(v_i^\top z).$$

- This is the CP decomposition restricted to the symmetric case.

Applications

- Statistics: cumulant tensors often admit low-rank symmetric decompositions, useful in blind source separation.
- Machine learning: compressing polynomial kernels, tensor regression, deep learning model compression.
- Algebraic geometry: the study of Waring decompositions of polynomials.

Why This Matters

- Low-rank symmetric decompositions reduce complexity, making huge tensors manageable.
- They connect tensor algebra with classical spectral theory and polynomial factorization.
- They explain why symmetric tensors are a natural language for representing data with latent low-dimensional structure.

Exercises

1. Matrix Case: Decompose

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

   as a sum of symmetric rank-one matrices.

2. Simple Symmetric Tensor: Write explicitly $v^{\otimes 3}$ for $v = (1, 2)$.

3. Symmetric Decomposition: Express the quadratic form $Q(x, y) = 4x^2 + 2xy + y^2$ as a sum of rank-one terms $\lambda_i(a_i x + b_i y)^2$.

4. Interpretation: For a 3rd-order symmetric tensor in $\mathbb{R}^2$, how many coefficients does it have? Compare this with how many are needed to specify a rank-1 symmetric decomposition.

5. Thought Experiment: Why might low-rank symmetric decompositions be especially valuable in machine learning (hint: think compression and interpretability)?

Part VI. Linear Maps Between Tensor Spaces

Chapter 11. Reshaping, Vectorization, and Commutation

11.1 Mode-n Unfolding and Matricization

When working with higher-order tensors, it is often useful to "flatten" them into matrices so that standard linear algebra tools can be applied. This process is called unfolding or matricization.

Definition: Mode-n Unfolding

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the mode-n unfolding arranges all entries into a matrix where:

- The row index corresponds to the n-th mode.
- The column index corresponds to all other modes combined.

Formally:

$$T_{i_1 i_2 \ldots i_N} \quad \mapsto \quad T^{(n)}_{i_n, j},$$

where j encodes the multi-index $(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$.

Example (3rd-Order Tensor)

If T has shape (I_1, I_2, I_3):

- Mode-1 unfolding: size $I_1 \times (I_2 I_3)$.
- Mode-2 unfolding: size $I_2 \times (I_1 I_3)$.
- Mode-3 unfolding: size $I_3 \times (I_1 I_2)$.

This reshaping preserves all entries, only changing their layout.

Why Unfold?

- Brings tensors into a familiar matrix framework.
- Enables the use of matrix decompositions (SVD, QR, eigenvalue methods).
- Used in algorithms for tensor decompositions (Tucker, CP, HOSVD).
- Useful in machine learning for feature extraction, low-rank approximations, and compression.

Connection to Linear Maps

Mode-n unfolding allows us to represent how a linear operator acts on the n-th mode of a tensor.

- For example, applying a matrix $A \in \mathbb{R}^{J \times I_n}$ to mode-n corresponds to left-multiplying the unfolded tensor:

$$(T \times_n A)^{(n)} = A\, T^{(n)}.$$

This makes tensor-matrix products consistent with matrix multiplication.

Why This Matters

- Mode-n unfolding bridges the gap between multilinear algebra and classical linear algebra.
- It provides the foundation for defining multilinear rank and tensor decompositions.
- Without unfolding, many computational algorithms for tensors would be infeasible.

Exercises

1. Shape Check: If T has dimensions $(2, 3, 4)$, what are the shapes of its mode-1, mode-2, and mode-3 unfoldings?
2. Matrix Form: Write out explicitly the mode-1 unfolding of a tensor $T \in \mathbb{R}^{2\times 2\times 2}$.
3. Operator Action: Let $T \in \mathbb{R}^{2\times 3\times 4}$. If $A \in \mathbb{R}^{5\times 2}$, what is the shape of $(T \times_1 A)$?
4. Reversibility: Explain why unfolding is reversible (i.e., why no information is lost).
5. Thought Experiment: Why is it advantageous to analyze tensors via unfoldings rather than directly in their multi-index form?

11.2 Vec Operator and Kronecker Identities

Once a tensor has been unfolded into a matrix, a common next step is to "flatten" that matrix into a vector. This operation is called vectorization, or the vec operator. It provides a bridge between multilinear algebra and standard linear algebra identities involving Kronecker products.

The Vec Operator

For a matrix $A \in \mathbb{R}^{m \times n}$:

$$\text{vec}(A) \in \mathbb{R}^{mn}$$

is obtained by stacking the columns of A into a single vector.

Example:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \text{vec}(A) = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix}.$$

For higher-order tensors, we apply vec after unfolding into a matrix.

Key Identity (Matrix Multiplication)

For matrices A, X, B of compatible sizes:

$$\text{vec}(AXB^\top) = (B \otimes A)\,\text{vec}(X).$$

- Here, $\otimes$ is the Kronecker product.
- This identity rewrites a two-sided matrix multiplication as a single linear transformation.

Tensor Extension

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, linear maps applied along each mode can be expressed in vec form using Kronecker products.

If $A_n \in \mathbb{R}^{J_n \times I_n}$, then

$$(T \times_1 A_1 \times_2 A_2 \cdots \times_N A_N)^{\text{vec}} = (A_N \otimes \cdots \otimes A_2 \otimes A_1)\,\text{vec}(T).$$

This is the multilinear analogue of the matrix vec identity.

Applications

- Efficient coding of multilinear transformations.
- Proofs of tensor decomposition algorithms.
- Numerical linear algebra (solving tensor equations).
- Machine learning: efficient representation of Kronecker-structured layers.

Why This Matters

- The vec operator turns multilinear problems into linear ones.
- Kronecker products capture structure in large transformations compactly.
- These tools make computation with tensors more systematic and efficient.

Exercises

1. Vec of a Matrix: Compute $\text{vec}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$.
2. Matrix Identity Check: Let $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Verify that $\text{vec}(AXB^\top) = (B \otimes A)\,\text{vec}(X)$.
3. Tensor Shape: If $T \in \mathbb{R}^{2 \times 3 \times 4}$, what is the dimension of $\text{vec}(T)$?
4. Kronecker Ordering: Explain why the order of factors in $(A_N \otimes \cdots \otimes A_1)$ matters.
5. Thought Experiment: Why does vectorization make multilinear algebra easier to connect with existing linear algebra tools?

11.3 Linear Operators Acting on Tensors

So far, we have unfolded tensors and vectorized them to connect with matrix operations. Now we look at tensors as natural domains and codomains for linear operators. This viewpoint is powerful because many tensor operations are simply linear maps on tensor product spaces.

Operators on Tensor Product Spaces

If $A : V \to V'$ and $B : W \to W'$ are linear maps, then their tensor product operator

$$A \otimes B : V \otimes W \to V' \otimes W'$$

is defined by

$$(A \otimes B)(v \otimes w) = (Av) \otimes (Bw).$$

This definition extends linearly to all of $V \otimes W$.

Example with Matrices

If $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times q}$, and $X \in \mathbb{R}^{n \times q}$:

$$(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^\top).$$

This links tensor product operators to Kronecker product identities.

Mode-n Multiplication as Operator Action

For a tensor $T \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, multiplying along the n-th mode by a matrix $A \in \mathbb{R}^{J \times I_n}$:

$$T' = T \times_n A,$$

is equivalent to applying the operator $I \otimes \cdots \otimes A \otimes \cdots \otimes I$ to $\mathrm{vec}(T)$.

Thus, tensor–matrix products are just specialized cases of linear operators on tensor spaces.

Higher-Order Operators

More generally, a linear operator can act on multiple modes simultaneously:

$$T' = T \times_1 A_1 \times_2 A_2 \cdots \times_N A_N.$$

This corresponds to the operator

$$A_N \otimes \cdots \otimes A_2 \otimes A_1$$

on the vectorized tensor.

Why This Matters

- Many "complicated" tensor operations are just linear maps in disguise.
- Operator language clarifies the connection between abstract multilinear algebra and concrete matrix computations.
- This sets the stage for defining tensor rank, decompositions, and algorithms.

Exercises

1. Operator on Product: Let $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Compute $(A \otimes B)(e_1 \otimes e_2)$.
2. Mode-1 Action: For $T \in \mathbb{R}^{2\times 3}$, let $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. Write explicitly how $T \times_1 A$ transforms rows.
3. Vec Form: Verify that $(A \otimes B)\,\mathrm{vec}(X) = \mathrm{vec}(BXA^\top)$ for $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = I$.
4. Composed Operators: Show that $(A_1 \otimes B_1)(A_2 \otimes B_2) = (A_1 A_2) \otimes (B_1 B_2)$.
5. Thought Experiment: Why is it useful to think of tensor–matrix multiplication as applying a linear operator on a tensor space, instead of just as array reshaping?

Chapter 12. Metrics, Forms, and Raising/Lowering Indices

12.1 Using Inner Products to Move Indices

In tensor calculus, indices can appear as upper (contravariant) or lower (covariant). Inner products give us a way to convert between them. This process is called raising and lowering indices.

Covariant vs. Contravariant Indices

- A vector v has components v^i (upper index).
- A covector (linear functional) ω has components ω_i (lower index).
- A tensor may mix both kinds, e.g. $T^i{}_j$.

Without an inner product, upper and lower indices live in different spaces (V and V^*).

Using the Metric Tensor

An inner product (or metric) g provides a natural way to link vectors and covectors.

- Metric tensor: $g_{ij} = \langle e_i, e_j \rangle$.
- Inverse metric: g^{ij} satisfies $g^{ij} g_{jk} = \delta^i_k$.

Lowering an Index

Given a vector v^i, define its covector by:

$$v_i = g_{ij} v^j.$$

This maps $V \to V^*$.

Raising an Index

Given a covector ω_i, define its vector by:

$$\omega^i = g^{ij} \omega_j.$$

This maps $V^- \to V$.

Example in $\mathbb{R}^2$ with Euclidean Metric

Let $g = I$ (the identity matrix). Then raising and lowering indices does nothing:

- $v_i = v^i$.
- $\omega^i = \omega_i$.

But in a non-Euclidean metric (e.g., relativity), raising/lowering changes components significantly.

Why This Matters

- Index manipulation allows us to move seamlessly between vector and covector viewpoints.
- In geometry and physics, metrics define lengths, angles, and duality.
- In relativity, raising and lowering indices with the Minkowski metric distinguishes between time and space components.

Exercises

1. Lowering Indices: In $\mathbb{R}^2$ with metric

$$g = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix},$$

   compute the covector v_i for $v^j = (1, 4)$.

2. Raising Indices: Using the same metric, compute ω^i from $\omega_j = (2, 6)$.

3. Check Consistency: Verify that raising and then lowering an index returns the original vector.

4. Physics Example: In special relativity, the Minkowski metric is

$$g = \text{diag}(-1, 1, 1, 1).$$

   Show how lowering the time component of a 4-vector changes its sign.

5. Thought Experiment: Why is an inner product (metric) essential for connecting vectors and covectors? What would break if we tried to raise/lower indices without one? ### 12.2 Dualizations and Adjoint Operations

Raising and lowering indices with a metric doesn't just apply to vectors and covectors - it also extends naturally to linear maps and tensors. This leads to the notions of duals and adjoints, which are central in multilinear algebra, physics, and functional analysis.

Dual of a Linear Map

Given a linear map $A : V \to W$, its dual map is

$$A^* : W^- \to V^*$$

defined by

$$(A^- \omega)(v) = \omega(Av),$$

for all $v \in V$, $\omega \in W^*$.

- In matrix form: If A has matrix M, then A^* has matrix $M^\top$.

Adjoint of a Linear Map

If V and W have inner products, we can define the adjoint $A^\dagger : W \to V$ by the property:

$$\langle Av, w \rangle_W = \langle v, A^\dagger w \rangle_V.$$

- In Euclidean space with the standard dot product, $A^\dagger = A^\top$.
- With a general metric tensor g, the adjoint depends on raising/lowering indices:

$$(A^\dagger)^i{}_j = g^{ik} A^l{}_k g_{lj}.$$

Extension to Tensors

- For tensors with mixed indices, dualization flips upper lower indices.
- Example: For $T^i{}_j$, the dual acts as

$$(T^*)_i{}^j = g_{ik} g^{jl} T^k{}_l.$$

This systematically moves indices while preserving linear relationships.

Why This Matters

- Dualization is purely algebraic (map between spaces and their duals).
- Adjoint operations involve the metric and carry geometric meaning (orthogonality, length).
- In physics, adjoints appear in energy conservation laws, quantum mechanics (Hermitian operators), and relativity.

Exercises

1. Dual Map: Let $A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$. Compute its dual A^*.
2. Adjoint in Euclidean Space: Show that in $\mathbb{R}^2$ with the standard dot product, the adjoint of A is just $A^\top$.
3. Adjoint with Metric: In $\mathbb{R}^2$ with metric

$$g = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix},$$

   compute the adjoint of $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
4. Tensor Dualization: For a tensor $T^i{}_j = \delta^i_j$ (the identity), compute its dual under the Euclidean metric.
5. Thought Experiment: Why does the adjoint depend on the choice of inner product, while the dual does not? What does this tell us about algebra vs. geometry? ### 12.3 Coordinate Rules Made Simple

So far, we've seen raising/lowering of indices, dual maps, and adjoints. These can look abstract, but in practice they reduce to straightforward coordinate rules once a basis and metric are fixed. This section summarizes the "index gymnastics" in a beginner-friendly way.

Raising and Lowering

- Lowering:

$$v_i = g_{ij} v^j$$

- Raising:

$$\omega^i = g^{ij} \omega_j$$

Here g_{ij} is the metric, g^{ij} its inverse.

Tip: Think of lowering as "applying g" and raising as "applying g^{-1}."

Dual Maps

For a matrix A representing a linear map:

$$(A^*)_{ij} = A_{ji}.$$

So the dual corresponds to a transpose in coordinates.

Adjoint Maps

With a general metric g, the adjoint of a matrix A is:

$$A^\dagger = g^{-1} A^\top g.$$

- If $g = I$ (Euclidean metric), then $A^\dagger = A^\top$.
- This formula generalizes to any inner product space.

Tensors with Mixed Indices

For a tensor $T^i{}_j$, indices can be moved by contracting with g_{ij} or g^{ij}:

- To lower an upper index:

$$T_{ij} = g_{ik} T^k{}_j.$$

- To raise a lower index:

$$T^{ij} = g^{jk} T^i{}_k.$$

This bookkeeping ensures correct transformations under basis changes.

Why This Matters

- These coordinate rules are the practical toolkit for working with metrics and adjoints.
- In relativity, this is exactly how time and space components are manipulated.
- In machine learning and physics, these rules underlie how gradients, covariances, and bilinear forms are expressed.

Exercises

1. Lower an Index: In $\mathbb{R}^2$ with metric $g = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$, compute v_i for $v^j = (1, 2)$.
2. Raise an Index: With the same metric, compute ω^i for $\omega_j = (4, 6)$.
3. Adjoint Check: For $A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$, compute $A^\dagger$ under $g = I$.
4. Mixed Tensor: Given $T^i{}_j = \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}$, compute T_{ij} using $g = I$.
5. Thought Experiment: Why do physicists insist on keeping track of upper vs. lower indices, while engineers often ignore the distinction in Euclidean spaces?

Part VII. Tensor Ranks and Decompositions

Chapter 13. Ranks for Tensors

13.1 Matrix Rank vs. Tensor Rank

Before diving into advanced tensor decompositions, we need to understand what rank means for tensors. Unlike matrices, where rank is simple and unique, tensors have several different notions of rank. This section starts with the familiar matrix rank and then introduces tensor rank.

Matrix Rank (Review)

For a matrix $A \in \mathbb{R}^{m \times n}$:

- The rank is the dimension of its column space (or row space).
- Equivalently, the smallest r such that

$$A = \sum_{i=1}^{r} u_i v_i^\top,$$

  where $u_i \in \mathbb{R}^m, v_i \in \mathbb{R}^n$.
- Each term is a rank-one matrix.

Example:

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix},$$

so its rank is 1.

Tensor Rank

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the tensor rank (sometimes called CP rank) is the smallest r such that

$$T = \sum_{i=1}^{r} u_i^{(1)} \otimes u_i^{(2)} \otimes \cdots \otimes u_i^{(N)},$$

where each $u_i^{(k)} \in \mathbb{R}^{I_k}$.

- Each term is a rank-one tensor (outer product of vectors).
- Rank measures how many simple pieces are needed to build the tensor.

Key Differences: Matrix vs. Tensor

- Uniqueness: Matrix rank is uniquely defined; tensor rank can vary depending on the notion (CP rank, Tucker rank, etc.).
- Computation: Matrix rank is easy to compute (via SVD); tensor rank is NP-hard to compute in general.
- Upper Bound: For a matrix $m \times n$, rank min(m,n). For a tensor, the maximum possible rank is often not obvious.

Example: 3-Way Tensor

Let

$$T_{ijk} = 1 \quad \text{if } i = j = k, \quad 0 \text{ otherwise.}$$

This "diagonal" tensor has tensor rank = dimension n.

Why This Matters

- Tensor rank generalizes the concept of complexity from matrices to higher-order data.
- Low-rank structure is crucial in applications: compression, latent factor models, signal separation.
- Understanding tensor rank lays the foundation for CP, Tucker, and TT decompositions.

Exercises

1. Matrix Rank: Compute the rank of

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}.$$

2. Tensor Rank-1: Show that the tensor $T_{ijk} = a_i b_j c_k$ has rank 1.
3. Decomposition Practice: Express

$$T_{ij} = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$$

   as a sum of rank-one matrices.
4. Diagonal Tensor: For T_{ijk} with $T_{111} = 1, T_{222} = 1$, all others 0, determine its rank.
5. Thought Experiment: Why might tensor rank be harder to compute and less well-behaved than matrix rank?

13.2 Multilinear (Tucker) Rank and Mode Ranks

Besides CP rank, tensors have another important notion of rank: the multilinear rank, also called the Tucker rank. This definition is based on the ranks of mode-n unfoldings and gives a more stable and computable measure of complexity.

Mode-n Rank

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$:

- The mode-n rank is the matrix rank of its mode-n unfolding $T^{(n)}$.
- Denote it by $\text{rank}_n(T)$.

Multilinear (Tucker) Rank

The multilinear rank of T is the tuple

$$(\text{rank}_1(T), \text{rank}_2(T), \ldots, \text{rank}_N(T)).$$

This captures how much independent variation the tensor has along each mode.

Example (3rd-Order Tensor)

Suppose $T \in \mathbb{R}^{3\times4\times5}$.

- Mode-1 unfolding rank = 2.
- Mode-2 unfolding rank = 3.
- Mode-3 unfolding rank = 4.

Then the multilinear rank of T is $(2, 3, 4)$.

Comparison with CP Rank

- CP rank: minimal number of rank-1 outer products.
- Tucker rank: ranks of unfoldings (tuple of integers).
- CP rank is a single number, often hard to compute.
- Tucker rank is a multi-dimensional profile, easier to compute via SVD of unfoldings.

Tucker Decomposition

The multilinear rank is closely tied to the Tucker decomposition:

$$T = G \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)},$$

where

- G = core tensor,
- $U^{(n)}$ = basis matrices capturing each mode's subspace.

The dimensions of G are exactly the multilinear rank.

Why This Matters

- Multilinear rank provides a practical, computable measure of tensor complexity.
- It is robust under noise and approximations, unlike CP rank.
- Widely used in tensor compression (Tucker, HOSVD) and machine learning.

Exercises

1. Matrix Case: Show that for a matrix (2nd-order tensor), the Tucker rank reduces to the usual matrix rank.
2. Rank Profile: Suppose $T \in \mathbb{R}^{2\times3\times4}$ has mode ranks (2,2,1). What does this tell you about its structure?
3. Unfolding Rank: For a tensor $T_{ijk} = u_i v_j$ (independent of k), compute its Tucker rank.
4. Comparison: Give an example of a tensor with low Tucker rank but high CP rank.
5. Thought Experiment: Why might Tucker rank be preferred over CP rank in applications like compression or noise-robust signal processing? ### 13.3 Identifiability and Uniqueness

When we decompose a tensor into simpler components, a natural question arises: is the decomposition unique? For matrices, the SVD gives a stable, essentially unique factorization. For tensors, things are more subtle. This section explores identifiability - the conditions under which tensor decompositions are unique.

Identifiability in CP Rank

For a CP (CANDECOMP/PARAFAC) decomposition

$$T = \sum_{i=1}^{r} u_i^{(1)} \otimes u_i^{(2)} \otimes \cdots \otimes u_i^{(N)},$$

the decomposition is said to be identifiable if no other decomposition with the same rank r exists (up to trivial scaling and permutation).

- Trivial indeterminacies:
  - Scaling: multiplying one factor by α and another by $1/\alpha$.
  - Permutation: reordering the components.

Kruskal's Condition (for Uniqueness)

A famous result: If each factor matrix $U^{(n)}$ has Kruskal rank k_n (maximum number of columns that are linearly independent), and

$$k_1 + k_2 + \cdots + k_N \geq 2r + (N-1),$$

then the CP decomposition is unique (up to scaling and permutation).

Identifiability in Tucker Rank

For Tucker decomposition

$$T = G \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)},$$

uniqueness is generally weaker:

- The core tensor G is not unique.
- However, the subspaces spanned by the factor matrices $U^{(n)}$ are unique (this is the essence of HOSVD).

Why Uniqueness Matters

- In applications like chemometrics, signal separation, and latent factor models, unique decomposition means interpretable components.
- Without identifiability, decompositions may still compress data but lose meaning.
- CP rank uniqueness is one reason tensors can capture latent structure more faithfully than matrices.

Challenges

- Checking uniqueness conditions is not always easy.
- In practice, algorithms may converge to non-unique solutions.
- Regularization and domain knowledge are often used to guide towards interpretable decompositions.

Exercises

1. Matrix vs. Tensor: Why is the SVD of a matrix unique (up to signs), but CP decomposition of a tensor not always unique?
2. Scaling Ambiguity: Show explicitly how scaling one factor vector and compensating in another keeps the CP decomposition unchanged.
3. Permutation Ambiguity: Given two equivalent CP decompositions, illustrate how permuting rank-1 components yields the same tensor.
4. Kruskal's Condition: Suppose a 3rd-order tensor has factor matrices with Kruskal ranks (3,3,3). For rank $r = 3$, does Kruskal's condition guarantee uniqueness?
5. Thought Experiment: Why might uniqueness of tensor decompositions be more valuable in data analysis than uniqueness of matrix decompositions?

Chapter 14. Cannonical Decompositions

14.1 CP (CANDECOMP/PARAFAC)

The Canonical Polyadic (CP) decomposition - also known as CANDECOMP/PARAFAC - is one of the most fundamental ways to factorize a tensor. It generalizes the idea of expressing a matrix as a sum of rank-one outer products to higher-order tensors.

Definition

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, a CP decomposition of rank r is:

$$T = \sum_{i=1}^{r} u_i^{(1)} \otimes u_i^{(2)} \otimes \cdots \otimes u_i^{(N)},$$

where each $u_i^{(n)} \in \mathbb{R}^{I_n}$.

- Each term is a rank-one tensor.
- The smallest such r is the CP rank of T.

Example (Matrix Case)

For a matrix A, the CP decomposition reduces to the familiar rank factorization:

$$A = \sum_{i=1}^{r} u_i v_i^\top.$$

Example (3rd-Order Tensor)

For $T \in \mathbb{R}^{I \times J \times K}$, the CP decomposition is:

$$T_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr},$$

where $A = [a_{ir}], B = [b_{jr}], C = [c_{kr}]$ are called factor matrices.

Properties

- CP decomposition is unique under mild conditions (contrast with matrix factorization).
- Provides a compact representation: instead of storing IJK entries, we store only factor matrices of size $(I + J + K)R$.
- Often interpretable in applications (each component corresponds to a latent factor).

Applications

- Psychometrics: original use of PARAFAC for analyzing survey data.
- Chemometrics: spectral analysis of chemical mixtures.
- Signal processing: blind source separation.
- Machine learning: latent factor models, recommender systems, neural net compression.

Why This Matters

- CP is the most direct generalization of matrix rank factorization.
- Unlike SVD, CP works without orthogonality constraints, making it more flexible.
- Its uniqueness is a key reason for its wide use in data analysis.

Exercises

1. Matrix Analogy: Show that the CP decomposition for a 2×2 matrix is the same as its rank decomposition.
2. 3rd-Order Example: Suppose $T_{ijk} = u_i v_j w_k$. Show that T has CP rank 1.
3. Factor Matrices: Write explicitly the factor matrices A, B, C for

$$T_{ijk} = \delta_{ij}\delta_{jk}.$$

4. Compression: Estimate storage cost for a tensor of size $50 \times 40 \times 30$ if represented directly vs. CP decomposition with rank $r = 10$.
5. Thought Experiment: Why might CP decomposition be more interpretable than Tucker decomposition in some applications?

14.2 Tucker and HOSVD

The Tucker decomposition is another foundational way to factorize tensors. It generalizes the matrix SVD to higher dimensions and is closely tied to the concept of multilinear rank. When computed with orthogonal factors, it is often called the Higher-Order SVD (HOSVD).

Tucker Decomposition

For a tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the Tucker decomposition is:

$$T = G \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)},$$

where:

- $G \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ is the core tensor,
- $U^{(n)} \in \mathbb{R}^{I_n \times R_n}$ are the factor matrices,
- $(R_1, R_2, \ldots, R_N)$ is the multilinear rank of T.

Comparison with CP

- CP: sum of rank-one components, no core tensor.
- Tucker: includes a core tensor that mixes components across modes.
- CP rank: single number, often hard to compute.
- Tucker rank: tuple $(R_1, \ldots, R_N)$, easier to compute (via SVD of unfoldings).

Higher-Order SVD (HOSVD)

The HOSVD is a special Tucker decomposition where:

- Each factor matrix $U^{(n)}$ has orthonormal columns (from the SVD of the mode-n unfolding).
- The core tensor G has certain orthogonality properties.

HOSVD is not unique but provides a stable, interpretable decomposition.

Example (3rd-Order Tensor)

For $T \in \mathbb{R}^{I \times J \times K}$:

$$T = \sum_{p=1}^{R_1} \sum_{q=1}^{R_2} \sum_{r=1}^{R_3} g_{pqr}\, u_p^{(1)} \otimes u_q^{(2)} \otimes u_r^{(3)}.$$

Here, the core entries g_{pqr} show how basis vectors across different modes interact.

Applications

- Compression: reduce each dimension by truncating factor matrices.
- Signal processing: spatiotemporal data analysis.
- Machine learning: feature extraction and dimensionality reduction.
- Neuroscience: multi-subject brain activity analysis.

Why This Matters

- Tucker/HOSVD generalize the SVD to tensors, making them intuitive for those familiar with matrices.
- Tucker rank is easier to compute than CP rank, and truncation gives practical low-rank approximations.
- Provides a balance between interpretability (factor matrices) and flexibility (core tensor).

Exercises

1. Matrix Analogy: Show that Tucker decomposition reduces to the usual SVD when $N = 2$.
2. Mode Ranks: For $T \in \mathbb{R}^{3 \times 4 \times 5}$, suppose its multilinear rank is $(2, 3, 2)$. What is the size of the core tensor?
3. Factor Matrix Construction: Explain how to construct the mode-1 factor matrix $U^{(1)}$ from the SVD of the mode-1 unfolding.
4. Compression: Estimate storage cost for a tensor of size $30 \times 40 \times 50$ with Tucker rank $(5, 5, 5)$. Compare to full storage.
5. Thought Experiment: Why might Tucker/HOSVD be better than CP for approximation, even if CP is more interpretable?

14.3 Tensor Trains (TT) and Hierarchical Formats

When tensors become very large (high dimensions or many modes), storing and computing with them directly becomes impossible. Tensor Train (TT) decomposition and related hierarchical formats provide scalable ways to represent such tensors with drastically reduced storage.

Tensor Train (TT) Decomposition

A tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is represented as a product of smaller 3-way tensors (called cores):

$$T_{i_1 i_2 \cdots i_N} = G^{(1)}_{i_1} G^{(2)}_{i_2} \cdots G^{(N)}_{i_N},$$

where:

- Each $G^{(k)}_{i_k}$ is an $r_{k-1} \times r_k$ matrix,
- $(r_0, r_1, \ldots, r_N)$ are the TT ranks, with $r_0 = r_N = 1$.

Thus, the tensor entry is computed by multiplying a chain of matrices.

Example (Order-3 Tensor)

For $T \in \mathbb{R}^{I_1 \times I_2 \times I_3}$:

$$T_{i_1 i_2 i_3} = \sum_{a=1}^{r_1} \sum_{b=1}^{r_2} G^{(1)}_{i_1, a} \, G^{(2)}_{a, i_2, b} \, G^{(3)}_{b, i_3}.$$

Storage Cost

- Full tensor: $I_1 I_2 \cdots I_N$ entries.
- TT decomposition: about $\sum_{k=1}^{N} I_k r_{k-1} r_k$.
- For moderate TT ranks, this is exponentially smaller.

Hierarchical Formats (HT, H-Tucker)

- Hierarchical Tucker (HT) generalizes TT with a tree structure.
- Both TT and HT exploit low-rank structure in different unfolding schemes.
- Widely used in scientific computing, quantum physics, and deep learning.

Applications

- Scientific computing: solving PDEs with high-dimensional discretizations.
- Quantum physics: matrix product states (MPS) in quantum many-body systems.
- Machine learning: compressing large models, representing structured kernels.

Why This Matters

- TT and HT make computations feasible in dimensions where naive methods fail ("curse of dimensionality").
- They connect tensor methods with physics (MPS) and numerical mathematics (low-rank solvers).
- Provide scalable building blocks for high-dimensional learning and simulation.

Exercises

1. Storage Comparison: Compute storage size for a tensor of size $10 \times 10 \times 10 \times 10$ vs. TT decomposition with TT ranks all equal to 5.
2. Chain Structure: For $T \in \mathbb{R}^{4\times4\times4}$, sketch the TT structure (cores and ranks).
3. Rank-1 Case: Show that if all TT ranks are 1, the TT decomposition reduces to a rank-one tensor.
4. Quantum Link: Explain how TT decomposition corresponds to Matrix Product States (MPS) in physics.
5. Thought Experiment: Why does the chain-like structure of TT decomposition scale better than CP or Tucker decompositions in very high dimensions?

14.4 Connections to SVD and PCA

Tensor decompositions are natural generalizations of matrix factorizations such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). Understanding these connections helps link classical linear algebra intuition with modern multilinear methods.

SVD Recap (Matrix Case)

For a matrix $A \in \mathbb{R}^{m \times n}$:

$$A = U\Sigma V^\top,$$

where

- U and V are orthogonal,
- Σ is diagonal with singular values.

This factorization is unique (up to signs) and provides:

- Rank,
- Best low-rank approximation,
- Geometric interpretation (rotations and scalings).

PCA Recap

PCA applies SVD to a data matrix:

- Rows = observations, columns = features.
- Extracts orthogonal directions (principal components) that maximize variance.

CP and SVD

- CP decomposition generalizes matrix rank factorization to higher orders.
- A matrix is just a 2-way tensor: CP = rank decomposition = SVD without orthogonality.
- For $N \geq 3$, CP decomposition does not correspond to orthogonal factors, but it still reveals latent components.

Tucker/HOSVD and SVD

- Tucker decomposition generalizes SVD to higher-order tensors.
- HOSVD is literally the higher-order SVD:
  - Compute SVD of each mode unfolding.
  - Factor matrices = left singular vectors.
  - Core tensor = interaction of components across modes.
- Truncating singular vectors gives best low-rank approximations in a multilinear sense.

PCA vs. Multilinear PCA

- PCA = best low-rank approximation of a data matrix.
- Tucker decomposition = multilinear PCA, reducing dimensionality along each mode simultaneously.
- Applications: face recognition, video compression, multi-way data analysis.

Why This Matters

- CP matrix rank factorization.
- Tucker/HOSVD SVD/PCA.
- These links provide intuition: tensor decompositions are not arbitrary; they extend familiar tools to multi-way data.

Exercises

1. SVD Analogy: Show how SVD of a 3×3 matrix can be viewed as a Tucker decomposition with a diagonal core.
2. CP vs. SVD: Explain why CP decomposition of a matrix reduces to its rank factorization, but not to SVD.
3. HOSVD Steps: Outline the steps of HOSVD for a 3rd-order tensor $T \in \mathbb{R}^{4\times5\times6}$.
4. PCA Analogy: Suppose we have a video dataset stored as a tensor (frames$\times$height$\times$width). Explain how Tucker decomposition acts as a multilinear PCA.
5. Thought Experiment: Why is orthogonality central in SVD/PCA, but not in CP decomposition? What are the trade-offs?

Part VIII. Computation and Numerical Practice

Chapter 15. Working with Tensors in Code

15.1 Efficient Indexing and Memory Layout

Working with tensors in practice requires careful attention to how they are stored and accessed in memory. Poor indexing can make even simple operations slow. This section explains how indexing works under the hood and how to optimize memory layout for performance.

Linearization of Multi-Indices

A tensor $T \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is stored in linear memory.

- Each entry $T_{i_1, i_2, \ldots, i_N}$ is mapped to a single offset in memory.
- Formula (row-major / C-style layout):

$$\text{offset}(i_1, \ldots, i_N) = i_1(I_2 I_3 \cdots I_N) + i_2(I_3 \cdots I_N) + \cdots + i_N.$$

- Formula (column-major / Fortran, MATLAB):

$$\text{offset}(i_1, \ldots, i_N) = i_1 + I_1(i_2 + I_2(i_3 + \cdots + I_{N-1} i_N)).$$

Strides

- A stride tells how far (in memory) you move when an index increases by 1.
- Example: in row-major order, stride along the last index = 1.
- Efficient access requires stepping through memory with small, contiguous strides.

Cache Efficiency

- CPUs fetch memory in blocks (cache lines).
- Accessing elements in a contiguous block is fast.
- Jumping across large strides leads to cache misses and slow performance.

Practical Implications

- Loop ordering matters: iterate over the innermost (stride-1) dimension in the inner loop.
- Slicing tensors may produce views with non-contiguous strides (NumPy, PyTorch). Copying may be needed for efficiency.
- Tensor libraries often allow explicit control of memory layout (row-major vs. column-major).

Why This Matters

- Many tensor operations are memory-bound, not compute-bound.
- Efficient indexing and layout can make the difference between minutes and milliseconds.
- A good understanding of strides helps when debugging tensor code in NumPy, PyTorch, JAX, etc.

Exercises

1. Offset Calculation (Row-Major): For $T \in \mathbb{R}^{2\times3\times4}$, what is the memory offset of $T_{1,2,3}$ (0-based indexing)?
2. Offset Calculation (Column-Major): For the same tensor, compute the offset of $T_{1,2,3}$ in column-major order.
3. Stride Check: In row-major order for $T \in \mathbb{R}^{3\times4\times5}$, what are the strides for each dimension?
4. Cache-Efficient Loop: Write pseudocode for iterating over all entries of a 3D tensor in row-major order.
5. Thought Experiment: Why might a deep learning library internally reorder tensor layouts depending on the hardware (CPU vs. GPU)?

15.2 BLAS, Einsum, Performance Patterns

After understanding memory layout, the next step in efficient tensor computation is using optimized libraries and abstractions. This section covers BLAS, the einsum notation, and common performance patterns that make tensor operations practical at scale.

BLAS: Basic Linear Algebra Subprograms

- BLAS is the standard library for high-performance vector and matrix operations.
- Many tensor operations reduce to BLAS calls:
  - Matrix multiplication (GEMM) is the backbone of most tensor contractions.
  - Level-1 BLAS: vector ops ($y \leftarrow ax + y$).
  - Level-2 BLAS: matrix-vector ops.
  - Level-3 BLAS: matrix-matrix ops (highest efficiency).

Why this matters: Efficient tensor libraries (NumPy, PyTorch, TensorFlow, JAX) rely heavily on BLAS under the hood.

Einsum Notation

The Einstein summation convention ("einsum") expresses tensor contractions concisely.

Example:

- Matrix multiplication:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

  In einsum: `einsum('ik,kj->ij', A, B)`.

- Inner product:

$$\langle u, v \rangle = \sum_i u_i v_i$$

  In einsum: `einsum('i,i->', u, v)`.

- Outer product:

$$T_{ij} = u_i v_j$$

  In einsum: `einsum('i,j->ij', u, v)`.

Advantages:

- Expresses contractions clearly without reshaping.
- Often compiles to highly efficient BLAS/GPU kernels.

Performance Patterns

1. Batching: Group operations across multiple tensors (e.g., batched GEMM in GPUs).
2. Blocking / Tiling: Break large tensors into cache-sized blocks to improve locality.
3. Fusing Operations: Combine multiple small operations into one kernel (important in GPU computing).
4. Avoiding Copies: Use views/strides instead of reshaping whenever possible.
5. Automatic Differentiation: Frameworks like PyTorch and JAX integrate einsum with backpropagation efficiently.

Why This Matters

- BLAS-level performance is critical for large-scale tensor applications.
- Einsum notation unifies different tensor operations under one compact framework.
- Recognizing performance patterns makes code scale across CPUs, GPUs, and accelerators.

Exercises

1. Einsum Practice: Write the einsum expression for computing

$$C_{ij} = \sum_{k,l} A_{ik} B_{kl} D_{lj}.$$

2. Outer Product: Using einsum, compute the 3rd-order tensor $T_{ijk} = u_i v_j w_k$.
3. BLAS Levels: Classify the following into BLAS level-1, 2, or 3:
   - Dot product,
   - Matrix-vector product,
   - Matrix-matrix product.
4. Batching Example: If you have 100 matrices of size 50×50, why is a batched GEMM faster than 100 separate GEMM calls?
5. Thought Experiment: Why might einsum be more maintainable than manually reshaping and transposing arrays for contractions?

15.3 Stability, Conditioning, Scaling Tricks

Efficient computation is not enough - tensor operations must also be numerically stable. Large-scale problems often involve ill-conditioned matrices or tensors, and small floating-point errors can accumulate dramatically. This section introduces stability concerns and practical tricks for keeping computations reliable.

Conditioning and Stability

- Condition number: For a matrix A,

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

  measures sensitivity of solutions to perturbations.

- In tensors, contractions can amplify errors, especially when factors are nearly linearly dependent.
- High tensor ranks often worsen conditioning.

Rule of thumb: Poorly conditioned problems cannot be solved accurately, no matter the algorithm.

Common Stability Issues in Tensors

1. Overflows/underflows: multiplying many large/small entries.
2. Loss of orthogonality: iterative algorithms drift from true subspaces.
3. Cancellation errors: subtracting nearly equal numbers.
4. Exploding/vanishing gradients: in automatic differentiation with deep tensor networks.

Scaling Tricks

- Normalization: Rescale vectors and factor matrices to keep entries in a safe range.
- Orthogonalization: Regularly re-orthogonalize factor matrices in decompositions (QR or SVD steps).
- Log-domain computations: Replace products with sums of logarithms to prevent overflow (e.g., in probabilistic models).
- Balanced scaling: In CP decompositions, distribute scale evenly across modes to avoid extreme values.

Regularization

- Add small perturbations (like λI) to stabilize inversions ("Tikhonov regularization").
- In optimization, add penalties to discourage ill-conditioned solutions.
- Helps avoid overfitting in statistical tensor models.

Why This Matters

- Numerical stability is essential for trustworthy tensor computations.
- Scaling and orthogonalization tricks are used in nearly every practical algorithm.
- Without them, decompositions may diverge, optimizations may fail, and results may become meaningless.

Exercises

1. Condition Number: Compute the condition number of

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 10^{-6} \end{bmatrix}.$$

   What does it tell you about stability?

2. Overflow Example: Suppose we compute the product of 100 numbers all equal to 1.01. Estimate the result. Why might floating-point overflow occur?

3. Scaling in CP: Explain why rescaling one factor by 10^6 and another by 10^{-6} in a CP decomposition gives the same tensor but may cause instability.

4. Orthogonalization: Describe how QR factorization can help maintain numerical stability in HOSVD computations.

5. Thought Experiment: Why might log-domain computation be essential in probabilistic models with tensors (e.g., hidden Markov models, Bayesian networks)?

Chapter 16. Automatic Differentiation and Gradients

16.1 Jacobians/Hessians as Tensors

In calculus, derivatives of multivariable functions are naturally represented as tensors. Recognizing this viewpoint helps connect analysis with multilinear algebra, and explains why tensors appear in optimization, machine learning, and physics.

Jacobian as a Matrix (2nd-Order Tensor)

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian matrix is

$$J_{ij} = \frac{\partial f_i}{\partial x_j}.$$

- It describes how small changes in input x produce changes in output $f(x)$.
- In tensor terms: $J \in \mathbb{R}^{m \times n}$.

Example:

$$f(x, y) = (x^2, xy), \quad J = \begin{bmatrix} 2x & 0 \\ y & x \end{bmatrix}.$$

Hessian as a 2nd-Order Derivative Tensor

For a scalar function $g : \mathbb{R}^n \to \mathbb{R}$, the Hessian is

$$H_{ij} = \frac{\partial^2 g}{\partial x_i \partial x_j}.$$

- It is a symmetric matrix ($H_{ij} = H_{ji}$).
- Encodes curvature: quadratic approximation of $g(x)$.
- In optimization, eigenvalues of H indicate convexity.

Higher-Order Derivatives as Tensors

- Third derivatives form a 3rd-order tensor:

$$T_{ijk} = \frac{\partial^3 g}{\partial x_i \partial x_j \partial x_k}.$$

- In general, the k-th derivative of g is a symmetric k-tensor.
- These appear in Taylor expansions, perturbation analysis, and physics (nonlinear elasticity, quantum chemistry).

Automatic Differentiation (AD) Perspective

- AD frameworks (PyTorch, JAX, TensorFlow) compute Jacobians, Hessians, and higher derivatives automatically.
- Under the hood, they construct tensors of partial derivatives and contract them efficiently.
- Tensor viewpoint clarifies why gradients, Jacobians, and Hessians are different "levels" of the same structure.

Why This Matters

- Viewing derivatives as tensors unifies multivariable calculus and multilinear algebra.
- Explains the role of Jacobians in transformations, Hessians in optimization, and higher derivatives in scientific modeling.
- Essential foundation for backpropagation and deep learning.

Exercises

1. Jacobian Practice: Compute the Jacobian of

$$f(x, y, z) = (xy, yz, xz).$$

2. Hessian Example: For $g(x, y) = x^2y + y^3$, compute the Hessian matrix.
3. Symmetry Check: Show explicitly that mixed partials of $g(x, y)$ are equal: $\frac{\partial^2 g}{\partial x \partial y} = \frac{\partial^2 g}{\partial y \partial x}$.
4. Third Derivative Tensor: Write down all nonzero entries of the 3rd derivative tensor of $g(x) = x^4$ (1D case).
5. Thought Experiment: Why is it natural that higher derivatives are symmetric tensors? What would break if they weren't?

16.2 Backprop as Structured Contractions

Backpropagation, the core algorithm behind training neural networks, is fundamentally a sequence of tensor contractions guided by the chain rule. Multilinear algebra provides a clean way to see why backprop works and why it is efficient.

Chain Rule in Tensor Form

For functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^p$:

$$J_{g \circ f}(x) = J_g(f(x))\, J_f(x),$$

where J denotes the Jacobian.

- Composition of functions = multiplication (contraction) of Jacobians.
- Backpropagation efficiently evaluates this chain without materializing huge Jacobians.

Forward vs. Reverse Mode

- Forward mode AD: propagate derivatives forward (good when inputs are few).
- Reverse mode AD (backprop): propagate sensitivities backward (good when outputs are few, e.g. scalar loss).

In reverse mode:

$$\frac{\partial L}{\partial x} = \left(\frac{\partial y}{\partial x}\right)^{\top} \frac{\partial L}{\partial y}.$$

This is a tensor contraction: contract gradient vector with a Jacobian.

Layer-by-Layer in Neural Networks

Each layer is a function:

$$h^{(l+1)} = \sigma(W^{(l)} h^{(l)} + b^{(l)}).$$

Backprop proceeds by:

1. Compute forward activations.
2. For loss L, compute gradient wrt output: $\frac{\partial L}{\partial h^{(L)}}$.
3. Contract backwards through each layer:
    - Jacobian of linear part: $W^{(l)}$.
    - Jacobian of nonlinearity: diagonal tensor of $\sigma'(z)$.

Example (Two Layers)

For $L(f(x))$ with $f(x) = \sigma(Wx)$:

$$\frac{\partial L}{\partial x} = W^\top(\sigma'(Wx) \odot \frac{\partial L}{\partial f}).$$

Here:

- $\odot$ = elementwise product,
- Contraction with $W^\top$ propagates gradient backward.

Tensor Viewpoint

- Jacobians are tensors.
- Backprop avoids forming full Jacobians (which would be huge) by contracting only along needed directions.
- Each step is a structured contraction of the gradient with the local Jacobian.

Why This Matters

- Explains efficiency: backprop runs in time proportional to the forward pass.
- Shows the unity of AD, calculus, and multilinear algebra.
- Clarifies why backprop generalizes beyond neural nets (any differentiable computational graph).

Exercises

1. Chain Rule Contraction: Let $f(x, y) = (x + y, xy)$, $g(u, v) = u^2 + v$. Write the backprop step explicitly using contractions.
2. Linear Layer: For $h = Wx$, show that $\frac{\partial L}{\partial x} = W^\top \frac{\partial L}{\partial h}$.
3. Nonlinear Layer: For $h = \tanh(z)$, derive the contraction rule for backpropagation.
4. Efficiency Check: Estimate the cost of explicitly forming the Jacobian of a fully connected layer with m outputs and n inputs, versus the cost of backprop.
5. Thought Experiment: Why is reverse-mode AD (backprop) much more efficient than forward-mode AD for training neural networks?

16.3 Practical Tips for PyTorch/JAX/NumPy

Automatic differentiation (AD) frameworks like PyTorch, JAX, and NumPy (with autograd extensions) make tensor calculus practical. But efficiency and clarity depend on how you structure code. This section gives concrete tips to avoid pitfalls and exploit the strengths of these libraries.

Tip 1. Use Vectorization, Not Loops

- Replace Python loops with tensorized operations.
- Example (inefficient):

```
y = torch.zeros(n)
for i in range(n):
    y[i] = a[i] - b[i]
```

- Example (efficient):

```
y = a - b
```

Tip 2. Exploit Broadcasting

- Broadcasting avoids unnecessary reshaping and repetition.
- Example:

```
# Add bias vector to each row
Y = X + b   # automatic broadcasting
```

- Broadcasting keeps memory use low and code clean.

Tip 3. Prefer `einsum` for Complex Contractions

- `einsum` is expressive and optimized.
- Example: matrix multiplication:

```
C = torch.einsum('ik,kj->ij', A, B)
```

- Works the same in NumPy and JAX.

Tip 4. Control Gradient Flow

- In PyTorch: `x.detach()` to stop gradients.
- In JAX: use `jax.lax.stop_gradient(x)`.
- Important for stabilizing training and avoiding accidental memory blowups.

Tip 5. Check Shapes with Assertions

- Many AD errors come from shape mismatches.
- Insert sanity checks:

```
assert X.shape == (batch, features)
```

- Shape discipline avoids subtle bugs in backprop.

Tip 6. Monitor Numerical Stability

- Use functions like `torch.nn.functional.log_softmax` instead of naive `softmax` to avoid overflow.
- Add small epsilons in denominators: `x / (y + 1e-8)`.
- Use mixed precision cautiously (FP16 vs. FP32).

Tip 7. Benchmark with Profilers

- PyTorch: `torch.profiler`.
- JAX: `jax.profiler.trace`.
- NumPy: `%timeit` (Jupyter).
- Helps identify bottlenecks in contractions and data movement.

Why This Matters

- Writing tensor code is easy; writing fast, stable, and scalable tensor code requires discipline.
- Following these practices prevents performance cliffs and silent gradient bugs.
- Bridges theory (multilinear algebra) with implementation (real training pipelines).

Exercises

1. Vectorization: Rewrite a loop-based dot product in PyTorch using vectorized syntax.
2. Broadcasting: Given $X \in \mathbb{R}^{100\times 50}$ and $b \in \mathbb{R}^{50}$, add b to each row using broadcasting.
3. Einsum Practice: Write the einsum expression for batched matrix multiplication $Y_b = A_b B_b$, with batch dimension b.
4. Gradient Stop: In PyTorch, why might we use `x.detach()` inside a training loop? Give an example.
5. Thought Experiment: Why is `log_softmax` numerically safer than `exp(x)/sum(exp(x))`?

Part IX. Applications you can touch

Chapter 17. Data Science and Signal Processing

17.1 Multilinear Regression

Regression is one of the most basic tools in data science: fitting a model that predicts an output from input data. When the data is naturally multi-way (tensor-structured) instead of flat vectors or matrices, multilinear regression becomes a natural extension.

Ordinary Regression (Review)

For data pairs (x_i, y_i):

$$y \approx Wx + b.$$

Here, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $W \in \mathbb{R}^{m \times n}$.

This assumes vector inputs.

Multilinear Regression Model

Suppose input data is a tensor $X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Instead of flattening X into a vector, we preserve its structure by using a multilinear map:

$$\hat{y} = X \times_1 W^{(1)} \times_2 W^{(2)} \cdots \times_N W^{(N)} + b,$$

where each $W^{(n)}$ acts along mode-n.

- This reduces parameter count dramatically compared to a full vectorized regression.
- Preserves interpretability along each mode (e.g., time, space, frequency).

Example

- Input: video clip $X \in \mathbb{R}^{\text{frames} \times \text{height} \times \text{width}}$.
- Flattened regression would need millions of parameters.
- Multilinear regression uses three factor matrices $W^{(\text{frames})}, W^{(\text{height})}, W^{(\text{width})}$, drastically reducing parameters.

Training

- Solve by least squares or regularized optimization.
- Often implemented via alternating minimization (update one $W^{(n)}$ at a time).
- Regularization (e.g., low-rank constraints) prevents overfitting.

Applications

- Neuroscience: predict brain activity from multi-way stimulus data.
- Chemometrics: regression on spectral cubes.
- Time-series analysis: structured prediction across modes (time, channels, features).

Why This Matters

- Exploits multi-way structure instead of destroying it by flattening.
- Reduces model complexity while keeping interpretability.
- Lays the groundwork for tensor methods in supervised learning.

Exercises

1. Flattened vs. Multilinear: For input $X \in \mathbb{R}^{10 \times 20}$ and output scalar, how many parameters does flattened regression need? How many parameters if we use multilinear regression with $W^{(1)} \in \mathbb{R}^{10 \times 3}, W^{(2)} \in \mathbb{R}^{20 \times 3}$?
2. Mode Multiplication: Write explicitly how $X \times_1 W^{(1)} \times_2 W^{(2)}$ works for a 2D input (matrix).
3. Interpretability: Explain why multilinear regression can separate effects of time and space in spatiotemporal data.
4. Optimization: Why is alternating minimization a natural algorithm for training multilinear regression models?
5. Thought Experiment: In what situations would flattening be acceptable, and when is multilinear regression clearly superior?

17.2 Spatiotemporal Data and Video Tensors

Many real-world datasets are not flat vectors or simple matrices but inherently multi-way arrays. A prime example is spatiotemporal data - measurements varying across both space and time. Video is a natural case: each frame is a 2D image, and the sequence of frames adds a temporal dimension, giving a 3rd-order tensor.

Video as a Tensor

A grayscale video with F frames, height H, and width W is naturally represented as:

$$X \in \mathbb{R}^{F \times H \times W}.$$

For color video, an additional channel dimension is added:

$$X \in \mathbb{R}^{F \times H \times W \times 3}.$$

Flattening this into a matrix or vector loses structure and explodes parameter count.

Tensor Decomposition for Spatiotemporal Data

1. Tucker decomposition:
    - Separates time, spatial rows, and spatial columns into low-rank factors.
    - Compresses video efficiently while preserving essential dynamics.
2. CP decomposition:
    - Represents data as a sum of rank-one spatiotemporal components.
    - Each component factors into (time profile) $\times$ (spatial pattern).
3. Tensor Train (TT):
    - Handles very long video sequences by chaining local factors.

Applications

- Compression: reduce storage while keeping perceptual quality.
- Background modeling: separate foreground objects from static background (via low-rank + sparse decomposition).
- Forecasting: use multilinear regression on decomposed factors to predict future frames.
- Pattern discovery: extract temporal modes (e.g., daily cycles) and spatial modes (e.g., recurring structures).

Beyond Video: General Spatiotemporal Data

- Climate data: temperature, humidity, pressure (time $\times$ latitude $\times$ longitude $\times$ altitude).
- Neuroscience: brain activity measured over time across sensors.
- Traffic flows: time $\times$ location $\times$ type of vehicle.

All benefit from multilinear analysis.

Why This Matters

- Treating spatiotemporal data as tensors respects its inherent multi-way structure.
- Leads to compact models, better interpretability, and efficient computation.
- Bridges data science, machine learning, and physics-based modeling.

Exercises

1. Video Dimensions: A color video with 100 frames of size 64×64. What is its tensor shape?
2. Compression: Estimate the storage size (in entries) of this video vs. a Tucker decomposition with ranks $(10, 10, 10, 3)$.
3. Foreground/Background: Explain how a low-rank + sparse model might separate background (low-rank) from moving objects (sparse).
4. Temporal Modes: If CP decomposition yields components of the form (time $\times$ space), how would you interpret a component with strong daily periodicity in the time factor?
5. Thought Experiment: Why might tensor decompositions uncover hidden structure in spatiotemporal data that PCA on flattened vectors would miss?

17.3 Blind Source Separation

Blind Source Separation (BSS) is the problem of extracting hidden signals (sources) from observed mixtures, without detailed knowledge of how they were mixed. Tensors provide powerful tools for solving BSS, often outperforming classical matrix-based methods.

The Mixing Problem

Suppose we observe signals $x(t) \in \mathbb{R}^m$ that are mixtures of n hidden sources $s(t) \in \mathbb{R}^n$:

$$x(t) = As(t),$$

where A is an unknown mixing matrix.

Goal: Recover $s(t)$ and A from only the observations $x(t)$.

Classical Approach: ICA (Independent Component Analysis)

- Assumes sources are statistically independent.
- Uses second- and higher-order statistics to separate signals.
- Works well for simple mixtures, but struggles with multi-way structure.

Tensor Approach to BSS

Moments and cumulants of observed signals are naturally represented as symmetric tensors:

- 2nd-order cumulant (covariance): matrix.
- 4th-order cumulant: 4th-order tensor.

By analyzing these higher-order tensors:

- Sources can be separated even when covariance is insufficient.
- CP decomposition of the cumulant tensor reveals source directions.

Example: Cocktail Party Problem

- Microphones record overlapping voices in a room.
- Covariance matrix cannot separate voices if they overlap in energy.
- 4th-order cumulant tensor factorization recovers independent voices.

Applications

- Audio processing: separating voices, music, or environmental sounds.
- Medical imaging: separating independent brain activity sources from EEG/fMRI data.
- Telecommunications: extracting signals from mixed channels.
- Finance: identifying independent factors driving market time series.

Why This Matters

- Tensor methods exploit multi-way statistical structure, not just pairwise correlations.
- CP decomposition guarantees identifiability in cases where matrix factorizations fail.
- This makes BSS one of the most successful real-world applications of multilinear algebra.

Exercises

1. Covariance Limitation: Why might two voices with similar pitch have indistinguishable covariance, but separable 4th-order statistics?
2. Tensor Rank: Explain why the CP rank of the 4th-order cumulant tensor corresponds to the number of independent sources.
3. Practical Example: Given three observed mixtures of two signals, sketch how CP decomposition could be used to separate them.
4. ICA vs. Tensor: Compare ICA (matrix-based) and tensor-based approaches for BSS. Which one uses more information about the data?
5. Thought Experiment: Why might tensors be especially effective for BSS when the number of sensors is close to the number of sources?

Chapter 18. Machine Learning and Deep Models

18.1 Convolutions as Multilinear Maps

Convolutions, a cornerstone of modern deep learning, are fundamentally multilinear operations. While often introduced algorithmically (sliding filters over data), they can be expressed neatly within the tensor algebra framework.

Convolution as a Tensor Contraction

Consider a 1D convolution of an input signal $x \in \mathbb{R}^n$ with a kernel $h \in \mathbb{R}^k$:

$$y_i = \sum_{j=1}^{k} h_j \, x_{i-j}.$$

This is a bilinear map: linear in both the input x and kernel h.

- In higher dimensions (2D, 3D), the same structure holds: convolution = tensor contraction between input data and kernel.

Convolution as a Multilinear Operator

For 2D convolution (images):

- Input: $X \in \mathbb{R}^{H \times W \times C}$ (height $\times$ width $\times$ channels).
- Kernel: $K \in \mathbb{R}^{r \times s \times C \times M}$ (filter height $\times$ filter width $\times$ channels $\times$ output channels).
- Output:

$$Y_{i,j,m} = \sum_{p,q,c} K_{p,q,c,m} \, X_{i+p,j+q,c}.$$

This is a 4-way contraction across indices p, q, c.

Tensor Perspective Benefits

1. Unification: Convolution is just a structured multilinear map.
2. Efficiency: Frameworks optimize convolution via tensor contractions and reshaping into matrix multiplications (im2col trick).
3. Generalization: Other operations (cross-correlation, attention) are tensor contractions of similar form.

Connection to Low-Rank Tensors

- Convolution kernels can be approximated by low-rank tensor decompositions (e.g., CP, Tucker).
- This reduces parameters and speeds up training in deep neural networks.
- Example: a 3D convolution kernel decomposed into separable 1D kernels.

Applications Beyond Deep Nets

- Signal processing: filtering, denoising, feature extraction.
- Physics: differential operators (Laplacian, wave equation) as convolutions.
- Graphics: image blurring, sharpening, edge detection.

Why This Matters

- Shows that convolutions are not "magic," but structured tensor contractions.
- Provides a natural bridge between deep learning and multilinear algebra.
- Explains why tensor decompositions are effective for convolutional networks.

Exercises

1. 1D Convolution: Write the convolution of $x = (1, 2, 3, 4)$ with $h = (1, -1)$ explicitly.
2. Tensor Formulation: For a grayscale image $X \in \mathbb{R}^{H \times W}$ and filter $K \in \mathbb{R}^{r \times s}$, express convolution as a tensor contraction.
3. Kernel Decomposition: Show how a separable 2D kernel (rank-1 matrix) can be written as outer product of two 1D kernels.
4. Low-Rank Compression: Estimate parameter savings if a $7 \times 7 \times 64 \times 128$ kernel is approximated by separable 7×1 and 1×7 filters.
5. Thought Experiment: Why might expressing convolutions as multilinear maps help in designing more efficient deep learning architectures?

18.2 Low-Rank Tensor Compression of Nets

Modern neural networks, especially convolutional and transformer-based models, contain millions (or even billions) of parameters. Many of these parameters are highly redundant. Low-rank tensor decompositions provide a principled way to compress networks without losing much accuracy.

Redundancy in Neural Nets

- Convolution kernels: $K \in \mathbb{R}^{r \times s \times C_{\text{in}} \times C_{\text{out}}}$.
- Fully connected layers: weight matrices $W \in \mathbb{R}^{m \times n}$.
- These often have effective rank much smaller than full dimensions.

CP Decomposition for Compression

A convolutional kernel K can be approximated as:

$$K \approx \sum_{i=1}^{R} a_i^{(1)} \otimes a_i^{(2)} \otimes a_i^{(3)} \otimes a_i^{(4)},$$

where R is small.

- Reduces parameters from $r \cdot s \cdot C_{\text{in}} \cdot C_{\text{out}}$ to about $R(r + s + C_{\text{in}} + C_{\text{out}})$.

Tucker Decomposition for Compression

Factorize kernel as:

$$K \approx G \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)},$$

where G is a small core tensor.

- Allows flexible rank choices along each mode.
- Often used for compressing fully connected layers.

Tensor Train (TT) for Compression

Large fully connected layers $W \in \mathbb{R}^{m \times n}$ can be reshaped into a high-order tensor and approximated in TT format.

- Parameters scale as $\mathcal{O}(dr^2 n)$ instead of $\mathcal{O}(mn)$.
- Enables deployment of large models on resource-limited devices.

Applications in Deep Learning

- CNNs: low-rank approximations of convolution filters.
- Transformers: compress attention matrices with tensor decomposition.
- Mobile AI: deploy compressed models on smartphones or edge devices.

Trade-offs

- Pros: fewer parameters, lower memory, faster inference.
- Cons: extra decomposition step, potential accuracy loss if ranks are too low, need for retraining/fine-tuning.

Why This Matters

- Low-rank tensor methods make deep learning more efficient and accessible.
- They link classical multilinear algebra directly with modern AI engineering.
- Provide theoretical tools to understand redundancy and overparameterization.

Exercises

1. Parameter Counting: Compare parameter count of a $7 \times 7 \times 64 \times 128$ convolution kernel vs. CP decomposition with rank $R = 20$.
2. Tucker Compression: Suppose a kernel has shape $10 \times 10 \times 32 \times 64$. If Tucker ranks are $(5, 5, 10, 10)$, how many parameters are needed (core + factors)?
3. TT Format: Explain how reshaping a 1024×1024 weight matrix into a 4th-order tensor enables TT compression.
4. Accuracy vs. Efficiency: Why might too aggressive a low-rank approximation harm model accuracy?
5. Thought Experiment: Could a neural net be trained directly in compressed tensor form, instead of compressing after training? What might be the advantages? ### 18.3 Attention as Tensor Contractions

The attention mechanism, central to transformer models, can be seen as a sequence of structured tensor contractions. Expressing attention in multilinear algebra terms clarifies both its efficiency and its flexibility.

Standard Attention Formula

Given queries $Q \in \mathbb{R}^{n \times d}$, keys $K \in \mathbb{R}^{m \times d}$, and values $V \in \mathbb{R}^{m \times d_v}$:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V.$$

- $QK^\top$: similarity scores between queries and keys.
- Softmax: normalizes across keys.

- Multiplication with V: aggregates values.

Tensor Contraction View

1. Similarity computation:

$$S_{ij} = \sum_k Q_{ik} K_{jk},$$

   a contraction over feature index k.

2. Weighted aggregation:

$$O_{i\ell} = \sum_j \text{softmax}(S_{ij})\, V_{j\ell}.$$

Thus, attention = two contractions:

- Contract Q with K (dot product).
- Contract softmax weights with V.

Multi-Head Attention as Blocked Contractions

- Split Q, K, V into h heads (smaller feature dimensions).
- Perform attention contraction in parallel for each head.
- Concatenate results.

This is equivalent to block-structured tensor contractions.

Low-Rank and Tensorized Variants

- Low-rank attention: approximate $QK^\top$ with a low-rank factorization.
- Tensorized attention: represent weights in CP/Tucker/TT form for efficiency.
- Linear attention: replace full contraction with kernelized approximations.

Applications and Insights

- Shows attention is not "black magic" but structured multilinear algebra.
- Explains why tensor decompositions reduce attention cost.
- Connects attention with classical bilinear forms and projections.

Why This Matters

- Brings transformers into the same framework as convolutions and regression.
- Provides a language for designing efficient attention mechanisms.
- Helps bridge deep learning architectures with tensor theory.

Exercises

1. Dot-Product Attention: Express $S = QK^\top$ as an einsum contraction.
2. Aggregation Step: Show how multiplying softmax-normalized scores with V is another einsum contraction.
3. Multi-Head Splitting: If $d = 64$ and $h = 8$, what is the per-head dimension?
4. Low-Rank Trick: If $QK^\top$ is approximated by $Q(UV^\top)K^\top$ with rank r, how does this reduce complexity?
5. Thought Experiment: Why might thinking of attention as a tensor contraction help design new transformer variants?

Chapter 19. Physics, Graphics, and Beyond

19.1 Stress/Strain Tensors

In physics and engineering, stress and strain are key concepts for understanding how materials deform under forces. Both are naturally expressed as second-order tensors, making them a classic application of multilinear algebra.

Strain Tensor (Deformation)

When a material is deformed, each point moves by a displacement vector $u(x)$.

- The strain tensor measures local stretching, compression, and shear.

$$\varepsilon_{ij} = \tfrac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

- Symmetric: $\varepsilon_{ij} = \varepsilon_{ji}$.
- Diagonal entries: stretching along axes.
- Off-diagonal entries: shear distortions.

Stress Tensor (Internal Forces)

The stress tensor σ_{ij} describes internal forces per unit area inside a material.

- Defined so that force on a surface with normal n_j is

$$f_i = \sigma_{ij} n_j.$$

- Diagonal entries: normal stresses (compression/tension).
- Off-diagonal entries: shear stresses.

Hooke's Law (Linear Elasticity)

Stress and strain are related by a 4th-order elasticity tensor C:

$$\sigma_{ij} = \sum_{k,l} C_{ijkl}\, \varepsilon_{kl}.$$

- In isotropic materials, C depends only on two constants (Young's modulus and Poisson's ratio).
- This is a bilinear relation between strain and stress tensors.

Eigenvalues and Principal Axes

- Stress tensor σ can be diagonalized:
  - Eigenvalues = principal stresses.
  - Eigenvectors = principal directions.
- Interpretation: directions along which stress is purely compressive or tensile.

Applications

- Civil engineering: bridge and building safety.
- Mechanical engineering: design of engines, aircraft, machines.
- Geophysics: stress in Earth's crust, earthquakes.
- Medical imaging: elastography for tissue stiffness.

Why This Matters

- Stress and strain are everyday tensor applications in engineering.
- They demonstrate how multilinear algebra naturally describes geometry and physics.
- Provide a tangible connection between abstract tensors and real-world forces.

Exercises

1. Strain Calculation: For displacement field $u(x, y) = (x + y, y)$, compute the strain tensor ε.
2. Stress on a Plane: If

$$\sigma = \begin{bmatrix} 10 & 2 \\ 2 & 5 \end{bmatrix}, \quad n = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

   compute the force vector f.
3. Symmetry: Show that both stress and strain tensors are symmetric.
4. Principal Stresses: Find the eigenvalues of $\sigma = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$. Interpret them.
5. Thought Experiment: Why is it natural that stress and strain are tensors instead of just vectors?

19.2 Inertia Tensors and Principal Axes

In mechanics, the moment of inertia describes how mass distribution resists rotational motion. While for simple objects it is a scalar, in general it is a second-order tensor - the inertia tensor.

Definition of Inertia Tensor

For a rigid body with mass density $\rho(\mathbf{r})$, the inertia tensor is:

$$I_{ij} = \int \left(\|\mathbf{r}\|^2 \delta_{ij} - r_i r_j \right) \rho(\mathbf{r}) \, dV,$$

where $\mathbf{r} = (x, y, z)$ is the position vector relative to the chosen origin.

- I_{ij} encodes how difficult it is to rotate the body around axis i.
- Symmetric: $I_{ij} = I_{ji}$.

Angular Momentum and Kinetic Energy

For angular velocity ω:

- Angular momentum:

$$\mathbf{L} = I\,\omega.$$

- Rotational kinetic energy:

$$T = \tfrac{1}{2}\omega^{\top} I\omega.$$

Thus, I acts as the matrix linking angular velocity to angular momentum.

Principal Axes

- Inertia tensor can be diagonalized:

$$I = P\Lambda P^{\top},$$

  where Λ contains principal moments of inertia, and P gives principal axes.
- Rotations about principal axes are "decoupled" and simpler to analyze.

Examples

1. Solid sphere (mass M, radius R):

$$I = \tfrac{2}{5}MR^2 I_3.$$

   (isotropic: same inertia around all axes).
2. Thin rod (length L, axis through center):

$$I = \tfrac{1}{12}ML^2.$$

3. Rectangular box: inertia tensor has different diagonal entries depending on edge lengths.

Applications

- Mechanical engineering: robotics, aerospace, vehicle dynamics.
- Astronomy: rotation of planets, stability of satellites.
- Computer graphics: simulating rigid-body dynamics in physics engines.

Why This Matters

- Inertia tensors are a clear example of a physical system governed by symmetric tensors.
- Principal axes give both mathematical elegance and practical insight (e.g., why objects tumble).
- Connects linear algebra (eigenvalues) directly with physical motion.

Exercises

1. Rod Example: Compute the inertia tensor of a thin rod of length L and mass M lying along the x-axis.
2. Sphere Symmetry: Show that a solid sphere's inertia tensor is isotropic (same in all directions).
3. Principal Axes: Diagonalize

$$I = \begin{bmatrix} 5 & 1 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

   Interpret the eigenvalues.
4. Angular Momentum: For $\omega = (1, 0, 0)$ and $I = \text{diag}(2, 3, 4)$, compute $\mathbf{L}$.
5. Thought Experiment: Why are principal axes of inertia so useful in spacecraft design?

19.3 3D Graphics: Transforms and Shading

Computer graphics relies heavily on linear and multilinear algebra. Behind every rendered image are tensor operations that handle transformations, lighting, and shading.

Homogeneous Coordinates and Transforms

- A 3D point (x, y, z) is represented as a 4D vector $(x, y, z, 1)$.
- Transformations are represented as 4×4 matrices:
  - Translation, rotation, scaling, perspective projection.
- Composition of transformations = matrix multiplication (tensor contraction).

Example:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = TRS \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Lighting as a Tensor Operation

The Phong reflection model:

$$I = k_a I_a + k_d(\mathbf{L} \cdot \mathbf{N}) I_d + k_s(\mathbf{R} \cdot \mathbf{V})^n I_s,$$

where

- $\mathbf{L}$ = light direction,
- $\mathbf{N}$ = surface normal,
- $\mathbf{R}$ = reflection direction,
- $\mathbf{V}$ = viewer direction.

Each dot product is a tensor contraction between vectors.

Normals and Transformations

- Surface normals transform differently than points (using inverse transpose of transformation matrix).
- Preserves correct shading under scaling/shearing.
- Another case where raising/lowering indices (via metrics) appears in practice.

Shading as Multilinear Maps

- Shading combines:
  - Light properties (color, intensity).
  - Surface properties (material, texture).
  - Geometry (normals, tangents).
- The mapping from these multi-way inputs to final pixel intensity is a multilinear function.

Applications in Graphics Pipelines

- Vertex shaders: apply transformations (matrix multiplications).
- Fragment shaders: compute color via multilinear lighting models.
- Physics-based rendering: more advanced tensor models (BRDFs, radiance fields).

Why This Matters

- Brings tensor ideas into a domain familiar to many learners: 3D graphics.
- Shows that rendering engines are, at heart, optimized tensor pipelines.
- Builds intuition that tensor algebra is not abstract - it powers everyday technology (games, movies, VR).

Exercises

1. Homogeneous Transform: Write the homogeneous transformation matrix for rotating 90° about the z-axis and then translating by (2,3,0).
2. Dot Product Lighting: Given $\mathbf{L} = (0, 0, 1)$, $\mathbf{N} = (0, 0, 1)$, compute the diffuse term in the Phong model.
3. Normal Transformation: Explain why a non-uniform scaling requires using the inverse transpose matrix to transform normals.
4. Matrix Composition: If M_1 is a scaling matrix and M_2 is a rotation, what is the composite transformation?
5. Thought Experiment: How might tensor decompositions (CP, Tucker) be used to compress lighting models or neural radiance fields (NeRFs)?

19.4 Quantum States and Operators

Quantum mechanics is one of the most natural playgrounds for multilinear algebra: states, observables, and dynamics are all encoded as tensors.

Quantum States as Vectors

- A pure quantum state is a vector in a complex Hilbert space:

$$|\psi\rangle \in \mathbb{C}^n.$$

- Example: a single qubit is a vector in $\mathbb{C}^2$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1.$$

Operators as Matrices (2nd-Order Tensors)

- Observables and dynamics are represented as linear operators (Hermitian or unitary matrices).
- Measurement probabilities come from contractions:

$$p = \langle\psi|A|\psi\rangle.$$

Composite Systems and Tensor Products

- Multi-particle systems live in the tensor product of state spaces.
- Two qubits:

$$\mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^4.$$

- Example entangled state (Bell state):

$$|\Phi^+\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Entanglement is simply non-separability in tensor terms.

Density Matrices (Mixed States)

- General states described by density operator ρ, a positive semidefinite Hermitian matrix with trace 1.
- Expectation values:

$$\langle A \rangle = \mathrm{Tr}(\rho A).$$

Quantum Gates as Tensor Maps

- Single-qubit gates: 2×2 unitary matrices (e.g., Pauli matrices).
- Multi-qubit gates: act via Kronecker (tensor) products.
- Example: CNOT = a 4×4 matrix acting on two-qubit states.

Applications

- Quantum computing: algorithms rely on tensor contractions for simulating circuits.
- Quantum many-body physics: tensor networks (MPS, PEPS) compress exponential state spaces.
- Chemistry: molecular states represented as high-order tensors.

Why This Matters

- Shows how tensors form the mathematical backbone of quantum mechanics.
- Explains entanglement as a tensor phenomenon.
- Connects multilinear algebra directly with one of the most exciting modern sciences.

Exercises

1. Qubit State: Write the state vector for a qubit in superposition $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
2. Operator Expectation: For Pauli-Z operator $\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, compute $\langle\psi|\sigma_z|\psi\rangle$ for $|\psi\rangle$ above.
3. Tensor Product: Compute $|0\rangle \otimes |1\rangle$ explicitly as a vector in $\mathbb{C}^4$.
4. CNOT Gate: Write the 4×4 matrix representation of the CNOT gate.
5. Thought Experiment: Why are tensor decompositions (MPS, PEPS, TT) essential for simulating quantum many-body systems efficiently?

Part IX. Glimpses Beyond This Book

Chapter 20. Manifolds and Tensor Fields (Preview)

20.1 Tangent and Cotangent Bundles

So far, we treated tensors on vector spaces with fixed bases. In geometry and physics, tensors often live on manifolds, where each point has its own tangent space. The tangent and cotangent bundles provide the foundation for tensor fields.

Tangent Space at a Point

For a smooth manifold M and point $p \in M$:

- The tangent space T_pM is the vector space of all possible velocity vectors of curves through p.
- Dimension of T_pM = dimension of M.
- Example: On a sphere S^2, T_pS^2 is the plane tangent to the sphere at p.

Tangent Bundle

The tangent bundle is the union of all tangent spaces:

$$TM = \bigsqcup_{p \in M} T_pM.$$

- A smooth manifold itself (of dimension $2n$ if $\dim M = n$).
- A point in TM is a pair (p, v) with $v \in T_pM$.

Cotangent Space and Bundle

- The cotangent space T_p^*M is the dual space of T_pM: linear functionals on tangent vectors.
- Elements are called covectors (or differential 1-forms).
- The cotangent bundle $T^*M = \bigsqcup_{p \in M} T_p^*M$.

Local Coordinates

If M has coordinates $(x^1, \ldots, x^n)$:

- Basis of tangent space: $\{\partial/\partial x^i\}$.
- Basis of cotangent space: $\{dx^i\}$.
- Any tangent vector $v \in T_pM$:

$$v = \sum_i v^i \frac{\partial}{\partial x^i}.$$

- Any covector $\omega \in T_p^*M$:

$$\omega = \sum_i \omega_i dx^i.$$

Why This Matters

- Tangent and cotangent bundles generalize the vector/covector distinction from linear algebra to curved spaces.
- They form the stage on which tensor fields (next sections) live.
- Central in physics: tangent = velocities, cotangent = momenta.

Exercises

1. Tangent Space: On the circle S^1, describe the tangent space at the point $(1, 0)$.
2. Cotangent Basis: In $\mathbb{R}^2$ with coordinates (x, y), what are the basis vectors of the tangent and cotangent spaces?
3. Pairing: For $v = v^1\partial/\partial x + v^2\partial/\partial y$ and $\omega = \omega_1 dx + \omega_2 dy$, compute $\omega(v)$.
4. Bundle Structure: Explain why the tangent bundle of a 2D manifold has dimension 4.
5. Thought Experiment: Why is momentum naturally a covector (in T_p^*M) instead of a vector?

20.2 Tensor Fields and Coordinate Changes

So far we have defined tangent and cotangent spaces at a single point. To do geometry and physics, we need tensors that vary smoothly across a manifold. These are called tensor fields.

Tensor Fields

- A tensor field of type (r,s) assigns to each point $p \in M$ a tensor

$$T(p) \in (T_pM)^{\otimes r} \otimes (T_p^*M)^{\otimes s}.$$

- Examples:
  - Vector field = (1,0) tensor field (assigns a tangent vector at each point).
  - Covector field (1-form): (0,1) tensor field.
  - Metric: (0,2) symmetric tensor field.

Coordinate Expressions

If $(x^1, \dots, x^n)$ are local coordinates, then:

- A vector field:

$$X = \sum_i X^i(x) \frac{\partial}{\partial x^i}.$$

- A covector field:

$$\omega = \sum_i \omega_i(x) dx^i.$$

- A general (r,s) tensor field:

$$T = \sum T^{i_1 \cdots i_r}{}_{j_1 \cdots j_s}(x) \frac{\partial}{\partial x^{i_1}} \otimes \cdots \otimes \frac{\partial}{\partial x^{i_r}} \otimes dx^{j_1} \otimes \cdots \otimes dx^{j_s}.$$

Coordinate Transformations

If we change coordinates from x^i to $\tilde{x}^j$:

- Basis vectors transform as

$$\frac{\partial}{\partial \tilde{x}^j} = \sum_i \frac{\partial x^i}{\partial \tilde{x}^j} \frac{\partial}{\partial x^i}.$$

- Dual basis transforms oppositely:

$$d\tilde{x}^j = \sum_i \frac{\partial \tilde{x}^j}{\partial x^i} dx^i.$$

- Tensor components transform with a mix of both rules (contravariant and covariant).

Example: For a (1,1) tensor field $A^i{}_j$:

$$\tilde{A}^i{}_j = \frac{\partial \tilde{x}^i}{\partial x^p} \frac{\partial x^q}{\partial \tilde{x}^j} A^p{}_q.$$

Why This Matters

- Tensor fields generalize the coordinate-free viewpoint: the object is intrinsic, components adapt to coordinates.
- Physics laws are tensorial: their form is preserved under coordinate transformations.
- Explains why tensors are the "language of nature" in relativity and continuum mechanics.

Exercises

1. Vector Field: Write the vector field $X = x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}$ on $\mathbb{R}^2$.
2. 1-Form Field: Write the 1-form $\omega = x\,dx + y\,dy$. Evaluate $\omega(X)$ for the vector field above.
3. Transformation Rule: Show how a vector field X^i transforms under coordinate change $x^i \mapsto \tilde{x}^j$.
4. Mixed Tensor: Verify the transformation law for a (1,1) tensor $A^i{}_j$.
5. Thought Experiment: Why is it essential in physics that tensorial equations look the same in any coordinate system?

20.3 Covariant Derivatives and Curvature

On flat spaces like $\mathbb{R}^n$, derivatives of vector fields are straightforward. On curved manifolds, however, we cannot subtract vectors at different points directly because they belong to different tangent spaces. The covariant derivative solves this problem and leads naturally to curvature.

Covariant Derivative

- For a vector field X and another vector field Y, the covariant derivative $\nabla_X Y$ measures how Y changes along X.
- Unlike the usual derivative, $\nabla_X Y \in T_p M$, so it lives in the tangent space at the same point.

In coordinates (x^i):

$$\nabla_i Y^j = \frac{\partial Y^j}{\partial x^i} + \Gamma^j_{ik} Y^k,$$

where Γ^j_{ik} are the Christoffel symbols of the connection.

Parallel Transport

- Parallel transport moves a vector along a curve while keeping it "as constant as possible."
- Depends on the connection.
- In Euclidean space, this agrees with ordinary translation; on curved manifolds (like spheres), the result depends on the path.

Curvature Tensor

The failure of parallel transport to be path-independent is measured by the Riemann curvature tensor:

$$R^i{}_{jkl} = \partial_k \Gamma^i_{jl} - \partial_l \Gamma^i_{jk} + \Gamma^i_{km} \Gamma^m_{jl} - \Gamma^i_{lm} \Gamma^m_{jk}.$$

- If $R = 0$, the manifold is flat (locally Euclidean).
- Nonzero R encodes intrinsic curvature.

Ricci Tensor and Scalar Curvature

- Contracting indices of the Riemann tensor gives the Ricci tensor R_{ij}.
- Further contraction gives the scalar curvature R.
- Central in Einstein's field equations of general relativity:

$$G_{ij} = R_{ij} - \tfrac{1}{2} g_{ij} R.$$

Why This Matters

- Covariant derivative generalizes differentiation to curved spaces.
- Curvature is intrinsic: no embedding is needed to detect it.
- These ideas connect multilinear algebra to geometry, relativity, and modern physics.

Exercises

1. Flat Space: Show that in Euclidean coordinates, Christoffel symbols vanish and the covariant derivative reduces to the usual derivative.
2. Sphere Example: Explain why parallel transport around a closed loop on a sphere rotates a vector.
3. Riemann Tensor Symmetries: Verify that $R^i{}_{jkl} = -R^i{}_{jlk}$.
4. Ricci Contraction: Show how to obtain R_{ij} from $R^k{}_{ikj}$.
5. Thought Experiment: Why does general relativity require curvature, while Newtonian gravity does not?

Chapter 21. Representation Theory and Invariants (Preview)

21.1 Group Actions on Tensor Spaces

Symmetry plays a central role in mathematics and physics. Groups capture symmetry, and their actions on tensor spaces reveal invariants and structure. This section introduces how groups act on tensors and why this matters.

Group Actions

- A group action of G on a vector space V is a map

$$G \times V \to V, \quad (g, v) \mapsto g \cdot v,$$

  such that $e \cdot v = v$ (identity acts trivially) and $(gh) \cdot v = g \cdot (h \cdot v)$.
- Example: The rotation group $SO(3)$ acts on $\mathbb{R}^3$ by matrix multiplication.

Group Actions on Tensor Products

If a group G acts on V, then it acts naturally on tensor powers:

$$g \cdot (v_1 \otimes v_2 \otimes \cdots \otimes v_k) = (g \cdot v_1) \otimes (g \cdot v_2) \otimes \cdots \otimes (g \cdot v_k).$$

Thus, tensors inherit group actions from their underlying vector spaces.

Examples

1. Rotations on Vectors: In physics, tensors transform under coordinate rotations. Stress and strain tensors are invariant laws expressed under such actions.
2. Permutation Group: Acts on tensor indices by permuting them. Leads to symmetric and antisymmetric tensors.
3. General Linear Group $GL(n)$: Natural action on $\mathbb{R}^n$. Extends to higher-order tensors, explaining transformation rules under basis changes.

Why Group Actions Matter

- Provide the language for defining invariants: tensorial equations remain true under group actions.
- In physics: laws of nature are symmetric under rotations, Lorentz transformations, gauge groups.
- In data science: invariance to permutations, rotations, or scalings improves model generalization.

Tensor Invariants

- A tensor is invariant under a group if $g \cdot T = T$ for all $g \in G$.
- Example: the Euclidean inner product $\langle x, y \rangle$ is invariant under $SO(n)$.
- Determinants, traces, and volume forms are other classical invariants.

Exercises

1. Vector Rotation: Show that the Euclidean norm $\|x\|^2 = x_1^2 + x_2^2 + x_3^2$ is invariant under $SO(3)$.
2. Permutation Action: Describe how the permutation (12) acts on a tensor T_{ijk}.
3. GL(n) Action: For a (1,1) tensor $A^i{}_j$, write its transformation rule under $GL(n)$.
4. Invariant Tensor: Prove that the Kronecker delta δ_{ij} is invariant under orthogonal transformations.
5. Thought Experiment: Why is invariance under certain group actions a guiding principle in formulating physical laws?

21.2 Invariant Tensors and Symmetry

Invariant tensors capture quantities that remain unchanged under group actions. They are the backbone of conservation laws in physics, canonical forms in mathematics, and inductive biases in machine learning.

Definition

A tensor T is invariant under a group G if

$$g \cdot T = T \quad \text{for all } g \in G.$$

- Example: Under the rotation group $SO(n)$, the Kronecker delta δ_{ij} is invariant.
- Example: The Levi-Civita symbol ε_{ijk} is invariant under $SO(3)$ but changes sign under reflections.

Classical Invariant Tensors

1. Inner Product: $\langle x, y \rangle = \delta_{ij} x^i y^j$ is invariant under orthogonal transformations.
2. Determinant: Expressed with $\varepsilon_{i_1 i_2 \ldots i_n}$, invariant under $SL(n)$ (special linear group).
3. Volume Form: Orientation-preserving transformations preserve volume.
4. Metric Tensor g_{ij}: Fundamental invariant under coordinate changes, defining distances.

Invariant Theory

- Studies polynomial functions or tensors that remain unchanged under group actions.
- Example: Symmetric polynomials invariant under the permutation group S_n.
- Example: Trace and determinant are invariants under conjugation by $GL(n)$.

Physics Examples

- Conservation of energy and momentum invariance under time and space translations (Noether's theorem).
- Angular momentum invariance under rotations.
- Gauge invariants define observable quantities in quantum field theory.

Applications in Data Science & ML

- Designing models invariant to certain transformations (e.g., convolutional nets = translation invariance).
- Graph neural networks = invariance under node permutations.
- Tensor methods enforce symmetries directly in architecture.

Why This Matters

- Invariants identify essential quantities independent of coordinates or representation.
- Symmetry reduces complexity: from many possible features to a few invariant ones.
- This unifies physics, pure math, and modern AI design.

Exercises

1. Rotation Invariance: Show explicitly that δ_{ij} is unchanged under rotation matrices $R \in SO(3)$.
2. Levi-Civita: Verify that ε_{ijk} changes sign under reflection (determinant = –1).
3. Determinant: Why is the determinant invariant under $SL(n)$ but not under all of $GL(n)$?
4. Graph Example: Explain why node permutation invariance is crucial in graph neural networks.
5. Thought Experiment: Why do invariants often correspond to conserved physical quantities?

21.3 Why Invariants Matter in Algorithms

Invariants are not just elegant mathematical objects - they are practical tools that make algorithms more robust, efficient, and interpretable. When an algorithm respects the right invariants, it exploits symmetry instead of fighting it.

Invariants Reduce Redundancy

- Without invariants, algorithms may waste time learning the same thing in multiple coordinate systems.
- Example: PCA uses covariance, which is rotation-invariant, so it doesn't matter how data is oriented.
- Example: Graph algorithms often rely on permutation-invariant structures (degree, adjacency spectrum).

Invariants Improve Robustness

- If an algorithm outputs the same result under symmetry transformations, results are stable and reproducible.
- Example: CNNs are translation-invariant, making them robust to shifts in input images.
- Example: Physics simulations rely on energy invariants for numerical stability.

Invariants Simplify Computation

- Many invariants collapse high-dimensional data into simpler summaries.
- Example: Determinant summarizes a matrix's volume-scaling property.
- Example: Tensor contractions with invariant tensors (like δ_{ij} or ε_{ijk}) simplify calculations.

Algorithm Design via Invariants

- Signal processing: invariant features (moments, cumulants) used for blind source separation.
- Machine learning: group-equivariant networks encode invariances (rotations, permutations).
- Optimization: invariance-aware preconditioning improves convergence.

Why This Matters

- Invariants let us design algorithms that generalize better, by focusing only on structure that truly matters.
- They connect deep theory (group actions, tensor algebra) with practical implementations (CNNs, GNNs, physics engines).
- Understanding invariants is key to the next generation of geometry- and symmetry-aware AI systems.

Exercises

1. Rotation-Invariant Feature: For a set of 2D points, explain why pairwise distances are invariant under rotations and translations.
2. Permutation-Invariance: Show that the sum of node features in a graph is invariant under permutations of node labels.
3. Algorithm Stability: Why does enforcing conservation of energy in a simulation improve long-term stability?
4. Invariant Contraction: Use δ_{ij} to show that contracting $A_{ij}\delta_{ij}$ yields the trace of A, an invariant.
5. Thought Experiment: If a machine learning model ignores known invariances in the data, what are the risks?

End Matter

A. Symbols and Notation Cheatsheet

This appendix gathers the most common symbols used throughout the book. Use it as a quick reference when working through chapters and exercises.

Vector Spaces and Duals

- V, W, U - vector spaces
- $\dim V$ - dimension of V
- V^* - dual space (space of linear functionals on V)
- $v \in V$ - vector
- $\alpha \in V^*$ - covector (linear form)

Tensors

- $T^r_s(V)$ - space of tensors of type (r,s) over V
- $u \otimes v$ - tensor (outer) product of u and v
- $T^{j_1 \dots j_r}_{i_1 \dots i_s}$ - components of a (r,s) tensor
- Contraction - summing over one upper and one lower index
- Symmetric / antisymmetric tensors - invariant or alternating under index permutations

Indices and Summation

- i, j, k, l, m, n - generic indices
- Einstein summation convention - repeated upper/lower indices are summed over
- δ_{ij} - Kronecker delta, identity for index contraction
- ε_{ijk} - Levi-Civita symbol (totally antisymmetric, used for cross products, determinants)

Linear Maps and Operators

- $A : V \to W$ - linear map from V to W
- $A^i{}_j$ - components of a (1,1) tensor, i.e., a linear operator
- $\mathrm{tr}(A)$ - trace of operator A
- $\det(A)$ - determinant of A

Tensor Operations

- $\otimes$ - tensor product
- $\wedge$ - wedge product (alternating tensor product)
- $\times_n$ - mode-n product of a tensor with a matrix
- $\mathrm{vec}(\cdot)$ - vectorization operator (flatten tensor into a vector)
- $\mathrm{Tr}(\cdot)$ - matrix trace
- ∇ - gradient / covariant derivative

Special Objects

- g_{ij} - metric tensor
- $R^i{}_{jkl}$ - Riemann curvature tensor
- R_{ij} - Ricci tensor
- R - scalar curvature
- $|v|$ or $\|v\|$ - norm of a vector
- $\langle u, v \rangle$ - inner product of u and v

Appendix B. Proof Sketches of Core Theorems

Appendix B.1 Universal Property of the Tensor Product (Proof Sketch)

The tensor product $V \otimes W$ is not just a convenient construction: it is characterized uniquely by its universal property. This section gives an intuitive sketch of the proof.

Statement of the Universal Property

Given vector spaces V, W over a field $\mathbb{F}$:

- There exists a vector space $V \otimes W$ together with a bilinear map

$$\otimes : V \times W \to V \otimes W$$

  such that:

For any bilinear map $f : V \times W \to U$ into another vector space U, there exists a unique linear map

$$\tilde{f} : V \otimes W \to U$$

satisfying

$$f(v, w) = \tilde{f}(v \otimes w).$$

Idea of the Proof

1. Build a candidate space:
   - Start with the free vector space generated by pairs (v, w).
   - Impose relations to enforce bilinearity:
     - $(v_1 + v_2, w) \sim (v_1, w) + (v_2, w)$
     - $(v, w_1 + w_2) \sim (v, w_1) + (v, w_2)$
     - $(\alpha v, w) \sim \alpha(v, w)$, $(v, \alpha w) \sim \alpha(v, w)$.

   The quotient space is defined as $V \otimes W$.
2. Define the canonical bilinear map:

$$(v, w) \mapsto v \otimes w.$$

3. Factorization property:
   - For any bilinear map $f : V \times W \to U$, define $\tilde{f}$ by

$$\tilde{f}(v \otimes w) = f(v, w).$$

   - This is well-defined because the relations in step 1 match the bilinear properties of f.
4. Uniqueness:

- Any linear map $\tilde{f}$ satisfying the condition must agree on generators $v \otimes w$.
- Since these generate the whole space, $\tilde{f}$ is unique.

Why This Matters

- The tensor product is defined *not- by coordinates but by this universal property.
- It guarantees that tensors are the natural home for bilinear (and multilinear) maps.
- In applications, this explains why changing bases, reshaping, or contracting tensors always behaves consistently.

Exercises

1. Matrix Multiplication as a Bilinear Map: Show that the bilinear map $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $f(x, y) = x^\top A y$, factors through a linear map on $\mathbb{R}^n \otimes \mathbb{R}^m$.
2. Dimension Formula: Using a basis $\{e_i\}$ for V and $\{f_j\}$ for W, prove that $\{e_i \otimes f_j\}$ is a basis of $V \otimes W$.
3. Uniqueness Check: Why can't two different linear maps $\tilde{f}_1, \tilde{f}_2 : V \otimes W \to U$ both satisfy $f(v, w) = \tilde{f}(v \otimes w)$?
4. Free Vector Space Analogy: Compare the construction of $V \otimes W$ to the way free groups or free vector spaces are defined.
5. Thought Experiment: If the tensor product is defined via a universal property, why does this make it "canonical" (independent of choices)?

Appendix B.2 Dimension Formula for Tensor Products (Proof Sketch)

The tensor product has a clean relationship between dimensions:

$$\dim(V \otimes W) = \dim(V) \cdot \dim(W).$$

Here's a sketch of why this is true.

Step 1. Choose Bases

- Let $\{e_i\}_{i=1}^m$ be a basis for V.
- Let $\{f_j\}_{j=1}^n$ be a basis for W.

Step 2. Construct Tensor Basis

- Consider all simple tensors $e_i \otimes f_j$, with $1 \leq i \leq m$, $1 \leq j \leq n$.
- There are exactly mn such tensors.

Step 3. Spanning Argument

- Any element of $V \otimes W$ is a linear combination of simple tensors $v \otimes w$.
- Expanding $v = \sum a_i e_i$, $w = \sum b_j f_j$, we get

$$v \otimes w = \sum_{i,j} a_i b_j \, (e_i \otimes f_j).$$

- Therefore, all tensors are linear combinations of $\{e_i \otimes f_j\}$.

Step 4. Linear Independence

- Suppose $\sum_{i,j} c_{ij} \, (e_i \otimes f_j) = 0$.
- Apply the universal property with bilinear maps defined by coordinate projections.
- One shows all coefficients c_{ij} must vanish.
- Hence, $\{e_i \otimes f_j\}$ is linearly independent.

Conclusion

- $\{e_i \otimes f_j\}$ is a basis.
- Thus, $\dim(V \otimes W) = mn = \dim(V) \cdot \dim(W)$.

Why This Matters

- Dimension formula ensures that tensor products don't "create new degrees of freedom" beyond combinations of existing bases.
- Makes tensor products predictable: if $V = \mathbb{R}^m$ and $W = \mathbb{R}^n$, then $V \otimes W \cong \mathbb{R}^{mn}$.
- Explains why tensor reshaping between matrices, higher arrays, and Kronecker products is consistent.

Exercises

1. Basis Construction: For $V = \mathbb{R}^2$ with basis $\{e_1, e_2\}$ and $W = \mathbb{R}^3$ with basis $\{f_1, f_2, f_3\}$, explicitly list the basis of $V \otimes W$.
2. Counting Dimensions: If $\dim(V) = 4$ and $\dim(W) = 5$, what is $\dim(V \otimes W)$?
3. Matrix Analogy: Show that $V \otimes W$ with chosen bases is isomorphic to the space of $m \times n$ matrices.
4. Independence Check: Prove linear independence of $\{e_i \otimes f_j\}$ by applying a bilinear map $f(e_i, f_j) = \delta_{ii_0}\delta_{jj_0}$.
5. Thought Experiment: How does this dimension formula generalize to $V_1 \otimes V_2 \otimes \cdots \otimes V_k$?

Appendix B.3 Decomposition of Tensors into Symmetric and Antisymmetric Parts (Proof Sketch)

Any tensor with two indices can be decomposed uniquely into a symmetric and an antisymmetric component. This is a fundamental structural result and often the first place where symmetry in tensors becomes useful.

Statement

Let $T \in V \otimes V$ (a bilinear form, or a $(0, 2)$-tensor). Then:

$$T = \tfrac{1}{2}(T + T^\top) \; + \; \tfrac{1}{2}(T - T^\top),$$

where

- $S = \frac{1}{2}(T + T^\top)$ is symmetric,
- $A = \frac{1}{2}(T - T^\top)$ is antisymmetric, and the decomposition is unique.

Proof Sketch

1. Symmetrization and Antisymmetrization Operators
    - Define the symmetrization operator:

$$\mathrm{Sym}(T)(u, v) = \tfrac{1}{2}[T(u, v) + T(v, u)].$$

- Define the antisymmetrization operator:

$$\mathrm{Alt}(T)(u,v) = \tfrac{1}{2}[T(u,v) - T(v,u)].$$

2. Linearity and Decomposition
   - Both operators are linear.
   - For any u, v:

$$T(u,v) = \mathrm{Sym}(T)(u,v) + \mathrm{Alt}(T)(u,v).$$

3. Uniqueness
   - Suppose $T = S + A$ with S symmetric and A antisymmetric.
   - Then $S = \mathrm{Sym}(T)$ and $A = \mathrm{Alt}(T)$.
   - No other decomposition is possible, proving uniqueness.

Example

Matrix form:

$$T = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}.$$

- Symmetric part:

$$S = \tfrac{1}{2}\left(\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}\right) = \begin{bmatrix} 1 & 3.5 \\ 3.5 & 2 \end{bmatrix}.$$

- Antisymmetric part:

$$A = \tfrac{1}{2}\left(\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}\right) = \begin{bmatrix} 0 & -0.5 \\ 0.5 & 0 \end{bmatrix}.$$

So $T = S + A$.

Why This Matters

- Symmetric tensors capture "energy-like" or metric quantities.
- Antisymmetric tensors capture orientation, area, and rotational effects.
- In physics: stress tensor = symmetric, electromagnetic field tensor = antisymmetric.

Exercises

1. Compute the Decomposition: Decompose

$$T = \begin{bmatrix} 0 & 2 \\ -1 & 3 \end{bmatrix}$$

   into symmetric and antisymmetric parts.

2. Uniqueness Check: Why can't a nonzero symmetric matrix also be antisymmetric?

3. Dimension Argument: Show that in $\mathbb{R}^n$:

   - Dimension of symmetric 2-tensors = $\frac{n(n+1)}{2}$.
   - Dimension of antisymmetric 2-tensors = $\frac{n(n-1)}{2}$.
   - Total adds up to n^2.

4. Application in Physics: Which parts of the strain tensor (from mechanics) and electromagnetic field tensor (from relativity) correspond to symmetric and antisymmetric parts?

5. Thought Experiment: Can you imagine a scenario where the antisymmetric part of a tensor carries more physical information than the symmetric part? ### Appendix B.4 Rank Decomposition for Matrices and Extension to CP Rank (Proof Sketch)

The concept of rank begins with matrices and extends to higher-order tensors. This appendix sketches the reasoning behind matrix rank decomposition and how it generalizes to the Canonical Polyadic (CP) decomposition for tensors.

Matrix Rank Decomposition

Statement: Any matrix $A \in \mathbb{R}^{m \times n}$ of rank r can be written as a sum of r rank-one matrices.

Formally:

$$A = \sum_{k=1}^{r} u^{(k)} (v^{(k)})^\top,$$

with $u^{(k)} \in \mathbb{R}^m$, $v^{(k)} \in \mathbb{R}^n$.

Sketch of Proof:

1. Column Space Basis:

   - Since $\text{rank}(A) = r$, there exist r independent columns.

- Let $u^{(1)}, \ldots, u^{(r)}$ be these basis vectors.

2. Expansion of Columns:

   - Each column of A is a linear combination of $\{u^{(k)}\}$.
   - Thus, A can be written as a sum of outer products between $u^{(k)}$ and suitable coefficient vectors $v^{(k)}$.

3. Minimality:

   - No fewer than r terms suffice: otherwise, the column space dimension would drop below r.

This proves the decomposition exists and is minimal.

Extension to Tensors: CP Decomposition

For a 3rd-order tensor $T \in \mathbb{R}^{I \times J \times K}$:

$$T_{ijk} = \sum_{r=1}^{R} a_i^{(r)} \, b_j^{(r)} \, c_k^{(r)},$$

or equivalently,

$$T = \sum_{r=1}^{R} a^{(r)} \otimes b^{(r)} \otimes c^{(r)}.$$

- R = tensor rank (minimal number of rank-one tensors).
- Unlike matrices, computing R is much harder (NP-hard in general).

Key Differences Between Matrix Rank and Tensor Rank

- Matrices: rank $\leq \min(m, n)$.
- Tensors: rank can exceed dimensions, and uniqueness properties are more subtle.
- Matrix SVD: always gives orthogonal decomposition.
- Tensor CP: uniqueness requires conditions (e.g., Kruskal's condition).

Why This Matters

- Rank decomposition gives the conceptual backbone of low-rank approximation.
- CP decomposition underlies applications in data compression, chemometrics, neuroscience, and machine learning.
- Shows how a simple linear algebra property grows into a rich multilinear theory.

Exercises

1. Matrix Rank-One Decomposition: Decompose

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$$

   into rank-one matrices.

2. Minimality Check: Why can't the above A be written as a single rank-one matrix?
3. Tensor Example: Express the tensor $T_{ijk} = \delta_{ij}\delta_{jk}$ (the identity cube) as a CP decomposition.
4. Rank Bound: Show that for $T \in \mathbb{R}^{I \times J \times K}$, the rank is at most $\min(IJ, JK, IK)$.
5. Thought Experiment: Why might tensor rank decomposition be *harder- but also* more useful- than matrix rank decomposition in data science? ### Appendix B.5 Identifiability Conditions for CP Decomposition (Proof Sketch)

One of the most important questions in tensor decomposition is uniqueness: when is a CP (Canonical Polyadic) decomposition determined uniquely (up to permutation and scaling)? This property is called identifiability.

Statement (Informal)

A CP decomposition

$$T = \sum_{r=1}^{R} a^{(r)} \otimes b^{(r)} \otimes c^{(r)}$$

is essentially unique (unique up to permutation and scaling of terms) if the factor matrices $[a^{(1)} \ \ldots \ a^{(R)}]$, $[b^{(1)} \ \ldots \ b^{(R)}]$, $[c^{(1)} \ \ldots \ c^{(R)}]$ satisfy certain rank conditions.

Kruskal's Theorem (Key Result)

Let $A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, C \in \mathbb{R}^{K \times R}$. Define the Kruskal rank of a matrix M, written k_M, as the maximum number such that every subset of k_M columns is linearly independent.

Kruskal's condition: If

$$k_A + k_B + k_C \geq 2R + 2,$$

then the CP decomposition is unique (up to trivial indeterminacies).

Sketch of Why This Works

1. Overcompleteness and Mixing:
   - Without conditions, many different sets of factors can represent the same tensor.
2. Column Independence:
   - Kruskal rank ensures that subsets of columns are independent, ruling out hidden degeneracies.
3. Balancing the Inequality:
   - The inequality ensures enough "spread" across modes so that factors can't be swapped or recombined into different decompositions.
4. Result:
   - Any alternative decomposition must match the original one (up to permutation and scaling).

Intuition

- For matrices, SVD gives uniqueness (up to rotations) because of orthogonality.
- For tensors, uniqueness is subtler: CP decomposition is often more unique than matrix decompositions.
- This uniqueness makes tensors powerful in applications like blind source separation and latent-variable modeling.

Why This Matters

- Guarantees that discovered latent factors have meaning.
- Critical in chemometrics, neuroscience, psychometrics, and machine learning.
- Explains why tensor methods succeed where matrix methods fail: uniqueness despite underdetermined settings.

Exercises

1. Kruskal Rank: Compute the Kruskal rank of

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

2. Uniqueness Check: Why is CP decomposition with $R = 1$ always unique?
3. Matrix vs Tensor: Compare uniqueness of matrix rank decomposition with tensor CP decomposition. Which one is stricter?
4. Practical Implication: In blind source separation, why is uniqueness of CP decomposition essential?
5. Thought Experiment: Why might identifiability be a *blessing- in data analysis but a* curse- for numerical algorithms?

Appendix D. Identities & "Cookbook"

This appendix collects the most frequently used identities in multilinear algebra, matrix calculus, and tensor manipulation. They are written in a quick-lookup style - proofs and derivations appear in the main chapters.

1. Kronecker Product & Vec Identities

- Vectorization of matrix products:

$$\operatorname{vec}(AXB) = (B^\top \otimes A)\operatorname{vec}(X).$$

- Vec of outer product:

$$\operatorname{vec}(uv^\top) = v \otimes u.$$

- Mixed product property:

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD),$$

  if dimensions match.

2. Trace Tricks

- Cyclic property:

$$\operatorname{tr}(AB) = \operatorname{tr}(BA).$$

- Frobenius inner product:

$$\langle A, B \rangle = \operatorname{tr}(A^\top B).$$

- Trace with Kronecker:

$$\operatorname{tr}(A \otimes B) = \operatorname{tr}(A)\operatorname{tr}(B).$$

3. Tensor Contractions

- Inner product of tensors:

$$\langle T, U \rangle = \sum_{i_1,\ldots,i_k} T_{i_1\ldots i_k} U_{i_1\ldots i_k}.$$

- Contraction with δ_{ij}:

$$A_{ij}\,\delta_{ij} = \operatorname{tr}(A).$$

- Contraction with Levi-Civita (ε_{ijk}):

$$\varepsilon_{ijk}\,u_j v_k = (u \times v)_i.$$

4. Determinant & Volumes

- Determinant via Levi-Civita:

$$\det(A) = \sum_{i_1,\dots,i_n} \varepsilon_{i_1\dots i_n} A_{1,i_1} A_{2,i_2} \cdots A_{n,i_n}.$$

- Volume of parallelepiped (vectors $v_1, \dots, v_n$):

$$\text{Vol} = |\det([v_1 \ \dots \ v_n])|.$$

5. Differential & Gradient Identities

- Gradient of quadratic form:

$$\nabla_x (x^\top A x) = (A + A^\top)x.$$

- Matrix calculus rule:

$$\nabla_X \operatorname{tr}(A^\top X) = A.$$

- Log-det derivative:

$$\nabla_X \log\det(X) = (X^{-1})^\top.$$

6. Useful Einsum Patterns

- Matrix multiplication:

```
C = np.einsum('ik,kj->ij', A, B)
```

- Tensor contraction:

```
y = np.einsum('ijk,k->ij', T, x)
```

- Inner product:

```
s = np.einsum('i,i->', u, v)
```

7. Symmetrization / Antisymmetrization

- Symmetrization of 2-tensor:

$$S_{ij} = \tfrac{1}{2}(T_{ij} + T_{ji}).$$

- Antisymmetrization:

$$A_{ij} = \tfrac{1}{2}(T_{ij} - T_{ji}).$$

- General k:

$$\mathrm{Sym}(T) = \frac{1}{k!} \sum_{\pi \in S_k} T_{i_{\pi(1)} \cdots i_{\pi(k)}}.$$

Appendix E.1 Mini-Project: Implement CP Decomposition for Small 3-Way Tensors

Goal: Learn how to compute a Canonical Polyadic (CP) decomposition of a small 3-way tensor by implementing an algorithm step-by-step. This project combines theory (tensor rank, factorization) with practice (numerical methods).

1. Background

- Recall: a 3rd-order tensor $T \in \mathbb{R}^{I \times J \times K}$ has CP form

$$T \approx \sum_{r=1}^{R} a^{(r)} \otimes b^{(r)} \otimes c^{(r)},$$

  where R is the target rank and $a^{(r)} \in \mathbb{R}^I, b^{(r)} \in \mathbb{R}^J, c^{(r)} \in \mathbb{R}^K$.

- CP decomposition generalizes SVD to higher-order tensors.
- Unlike SVD, exact decomposition is not guaranteed for arbitrary tensors, so we usually solve an optimization problem.

2. Implementation Plan

Step 1: Generate a Synthetic Tensor

- Choose dimensions, e.g., $I = J = K = 4$.
- Pick random factor matrices $A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, C \in \mathbb{R}^{K \times R}$.
- Form a tensor:

$$T_{ijk} = \sum_{r=1}^{R} A_{ir} B_{jr} C_{kr}.$$

Step 2: Alternating Least Squares (ALS) Algorithm

- Initialize random factor matrices $\hat{A}, \hat{B}, \hat{C}$.
- Repeat until convergence:
  1. Fix $\hat{B}, \hat{C}$, update $\hat{A}$ by solving least squares.
  2. Fix $\hat{A}, \hat{C}$, update $\hat{B}$.
  3. Fix $\hat{A}, \hat{B}$, update $\hat{C}$.

Step 3: Evaluate Error

- Reconstruction error:

$$\text{Error} = \frac{\|T - \hat{T}\|_F}{\|T\|_F}.$$

- Stop when error falls below threshold (e.g., 10^{-6}) or max iterations reached.

3. Coding Hints

- Use NumPy (or PyTorch/JAX) for efficient tensor operations.
- Reshape and unfold tensors along modes for least squares updates.
- Use `numpy.linalg.lstsq` to solve linear systems.
- Normalize columns of factors periodically to avoid numerical instability.

4. Extensions (Optional)

- Compare convergence for different ranks R.
- Add Gaussian noise to the tensor and see how well CP decomposition recovers the factors.
- Test uniqueness: permute/scaled versions of factors should still reconstruct the tensor.
- Visualize factor matrices as heatmaps.

5. Deliverables

- Working implementation of CP-ALS for 3-way tensors.
- Plots of error vs. iterations.
- Short report (1–2 pages) discussing:
  - Accuracy of decomposition.
  - Effect of rank choice.
  - Challenges in convergence.

By finishing this project, you'll understand CP decomposition at both the conceptual (tensor rank, multilinearity) and practical (numerical algorithms, implementation) levels.

Appendix E.2 Mini-Project: Tucker Decomposition for Video Compression

Goal: Explore how Tucker decomposition can compress multi-way data efficiently, using a small grayscale video (or synthetic 3D tensor).

1. Background

- A video clip with F frames, each of size $H \times W$, is naturally represented as a 3-way tensor:

$$X \in \mathbb{R}^{F \times H \times W}.$$

- Tucker decomposition:

$$X \approx G \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)},$$

  where

  - G is a smaller core tensor,
  - $U^{(1)}, U^{(2)}, U^{(3)}$ are factor matrices (orthogonal bases along each mode).

- This is a higher-order analogue of SVD (HOSVD).

2. Implementation Plan

Step 1: Data Preparation

- Option A: Load a small grayscale video (e.g., $30 \times 64 \times 64$: 30 frames, 64×64 pixels).
- Option B: Generate synthetic video data (moving shapes, noise).

Step 2: Compute Tucker Decomposition (HOSVD)

1. Unfold X along each mode (frame, height, width).
2. Compute SVD of each unfolding.
3. Select leading r_1, r_2, r_3 singular vectors for factor matrices $U^{(1)}, U^{(2)}, U^{(3)}$.
4. Form core tensor:

$$G = X \times_1 (U^{(1)})^\top \times_2 (U^{(2)})^\top \times_3 (U^{(3)})^\top.$$

Step 3: Reconstruct Compressed Video

- Approximation:

$$\hat{X} = G \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}.$$

- Compare reconstruction with original.

Step 4: Evaluate Compression

- Compression ratio:

$$\text{CR} = \frac{\text{entries in original tensor}}{\text{entries in core + factors}}.$$

- Reconstruction error:

$$\text{Error} = \frac{\|X - \hat{X}\|_F}{\|X\|_F}.$$

3. Coding Hints

- Use NumPy or Tensorly (`tensorly.decomposition.tucker`) to avoid low-level SVD coding.
- Visualize reconstruction error by displaying frames before and after compression.
- Try varying rank choices (r_1, r_2, r_3).

4. Extensions (Optional)

- Compare Tucker compression with simple PCA on flattened frames.
- Add noise to video and check whether Tucker decomposition captures underlying structure better than PCA.
- Explore color video: treat as a 4-way tensor $F \times H \times W \times 3$.
- Implement randomized SVD for scalability.

5. Deliverables

- Code that performs Tucker decomposition on a small video dataset.
- Plots:
  - Error vs. compression ratio.
  - Example frame (original vs. compressed reconstruction).
- Short writeup explaining:
  - How compression works.
  - Trade-off between rank choice and accuracy.

By completing this project, you'll see how tensor decompositions reduce storage and preserve structure in real data, and why multilinear methods are superior to naive flattening.

Appendix E.3 Mini-Project: Strain Tensor for a Rotating Plate

Goal: Understand how the strain tensor captures deformation by computing it for a simple rotating square plate. This bridges mechanics (stress/strain) with tensor calculus.

1. Background

- In continuum mechanics, the strain tensor measures local deformation of a material:

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),$$

  where $u(x)$ is the displacement field.
- Pure rotation should produce no strain (only rigid-body motion).
- This project demonstrates that principle.

2. Setup: Rotating Plate

- Consider a 2D square plate centered at the origin.
- Apply a small rotation by angle θ.
- Displacement of a point (x, y):

$$u(x, y) = R(\theta)\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix},$$

  where $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$.

3. Compute Strain Tensor

1. Expand $u(x, y)$:

$$u_1(x, y) = (\cos\theta - 1)x - (\sin\theta)y,$$

$$u_2(x, y) = (\sin\theta)x + (\cos\theta - 1)y.$$

2. Compute partial derivatives:
   - $\frac{\partial u_1}{\partial x} = \cos\theta - 1$,
   - $\frac{\partial u_1}{\partial y} = -\sin\theta$,
   - $\frac{\partial u_2}{\partial x} = \sin\theta$,
   - $\frac{\partial u_2}{\partial y} = \cos\theta - 1$.

3. Build strain tensor:

$$\varepsilon = \tfrac{1}{2}\begin{bmatrix} 2(\cos\theta - 1) & (-\sin\theta + \sin\theta) \\ (\sin\theta - \sin\theta) & 2(\cos\theta - 1) \end{bmatrix}.$$

4. Simplify:

$$\varepsilon = (\cos\theta - 1)I,$$

which vanishes when θ is small (pure rotation no strain).

4. Interpretation

- For infinitesimal θ, expand $\cos\theta \approx 1 - \frac{1}{2}\theta^2$.
- So strain $\varepsilon \sim -\frac{1}{2}\theta^2 I$, i.e., negligible for small angles.
- Confirms physical intuition: rigid rotations produce no first-order strain.

5. Coding Hints

- Implement in Python/NumPy: define displacement field, compute gradients symbolically (SymPy) or numerically (finite differences).
- Visualize deformation arrows for a square grid of points.
- Plot strain tensor components as heatmaps.

6. Extensions (Optional)

- Apply non-uniform deformation (e.g., shear or stretching) and compute strain.
- Compare symmetric (strain) vs. antisymmetric (rotation) parts of displacement gradient.
- Extend to 3D cube rotation.

7. Deliverables

- Code computing strain tensor for rotating plate.
- Visualization of deformation vs. strain (showing nearly zero strain for pure rotation).
- Short explanation connecting math to physical interpretation.

This project illustrates how the strain tensor isolates real deformation, distinguishing it from rigid-body motion.

Appendix E.4 Mini-Project: Parallel Transport of a Vector on a Sphere

Goal: Visualize and compute parallel transport of a vector along a closed path on a sphere, showing how curvature affects vector orientation.

1. Background

- On curved manifolds, moving a vector along a path while keeping it "as constant as possible" is called parallel transport.
- On a flat plane: parallel transport around any loop returns the vector unchanged.
- On a sphere: transporting a vector around a loop can change its orientation, revealing curvature.

2. Setup: The Sphere S^2

- Sphere of radius $R = 1$, embedded in $\mathbb{R}^3$.
- Tangent space at a point $p \in S^2$: plane orthogonal to p.
- Path: for simplicity, use a triangle on the sphere (e.g., along the equator and a meridian).

3. Parallel Transport along Geodesics

1. Choose Starting Point:
   - $p_0 = (0, 0, 1)$ (north pole).
   - Tangent vector: $v_0 = (1, 0, 0)$.
2. Transport Path:
   - Move vector along a geodesic (great circle).
   - Keep it tangent at each point.
   - Algorithmically: project derivative back into tangent space.
3. Closed Loop Example:
   - Move from north pole down to equator (longitude 0).
   - Travel along equator by 90°.
   - Return to north pole.
4. Result:
   - Vector rotates relative to original orientation.
   - Rotation angle equals the spherical excess of the triangle (area on sphere).

4. Coding Hints

- Use Python + NumPy/Matplotlib.
- Represent path as discrete points on sphere.
- At each step:
  - Move point along geodesic.
  - Update vector by projecting back into tangent plane:

$$v \leftarrow v - (v \cdot p)p, \quad \text{then normalize.}$$

- Visualize trajectory of vector arrows along sphere using 3D plotting (`matplotlib.pyplot.quiver`).

5. Extensions (Optional)

- Experiment with different spherical triangles.
- Compute holonomy angle = enclosed area × curvature (for unit sphere, curvature = 1).
- Compare parallel transport on sphere vs. flat plane (no change).
- Extend to numerical geodesics on other manifolds (torus, hyperbolic surface).

6. Deliverables

- Code that simulates parallel transport on a sphere.
- 3D visualization of vector transport around loop.
- Measurement of net rotation angle (compare with theory).
- Short reflection: "What does this reveal about curvature?"

This project gives tangible insight into how curvature manifests in parallel transport, making an abstract tensorial concept geometrically vivid.

Appendix E.5 Mini-Project: PCA vs. Tucker Decomposition on Real Data

Goal: Compare Principal Component Analysis (PCA), which is matrix-based, with Tucker decomposition, which is tensor-based, on a real dataset. This project shows why multilinear approaches capture more structure than flattening data.

1. Background

- PCA: finds low-rank approximations of matrices (e.g., flatten images or videos into 2D).
- Tucker decomposition: generalizes PCA to tensors, preserving multi-way structure.
- Key question: Does Tucker give a better representation than PCA when applied to data with natural multi-dimensional structure (e.g., images, videos, EEG signals)?

2. Dataset Options

- Images: MNIST digits (28×28 grayscale images).
- Video: small grayscale clip (frames × height × width).
- Multichannel signals: EEG (time × channel × trial).

3. Methodology

Step 1: Prepare Data

- For PCA: flatten each sample into a long vector.
- For Tucker: keep data as tensor.

Step 2: Apply PCA

- Use `scikit-learn` PCA.
- Choose top k components.
- Reconstruct approximations of data.

Step 3: Apply Tucker Decomposition

- Use `tensorly.decomposition.tucker`.
- Choose rank tuple (r_1, r_2, r_3).
- Reconstruct approximations.

Step 4: Compare Results

- Compute reconstruction error:

$$\text{Error} = \frac{\|X - \hat{X}\|_F}{\|X\|_F}.$$

- Compare compression ratio (# of parameters stored).
- Visualize original vs reconstructed samples.

4. Coding Hints

- Libraries: `numpy`, `scikit-learn`, `tensorly`.
- To compare fairly: match number of parameters used by PCA and Tucker.
- For MNIST: display digits reconstructed with 5, 10, 20 components.
- For Tucker: vary ranks (e.g., (10,10) vs (5,10,5)).

5. Extensions (Optional)

- Add Gaussian noise to data and test denoising power of PCA vs Tucker.
- Compare runtime and scalability.
- Try CP decomposition instead of Tucker.
- Explore color images as 3-way tensors (height $\times$ width $\times$ channels).

6. Deliverables

- Code implementing both PCA and Tucker on chosen dataset.
- Plots:
  - Error vs. compression ratio.
  - Side-by-side reconstruction images (original, PCA, Tucker).
- Short writeup:
  - Which method captures structure better?
  - Does Tucker handle correlations across modes more effectively?

This project highlights the advantage of tensor methods over flattening approaches, making the case for multilinear models in data science.