Chapter 3: Geometry & Algebra of Tensors

Chapter two discussed linear subspace learning using projective and manifold learning methods using mapping functions. The main emphasis was to use linear algebra to project to a lower-dimensional space. The mapping function is mainly dependent on the eigendecomposition of the dataset. Other approaches learn a manifold that belongs to a lower dimension using a submatrix by defining neighbourhood or adjacency between data points. Projecting the data points to a higher dimensional space in which they will be linearly separable is another method to apply when data are not linearly separable in their current dimension. The Kernel trick enables the dot product to be performed in Hilbert space without doing the actual mapping.

This chapter starts with a reminder of the intuition of data analysis in higher dimensional spaces and their applications. Section two will continue the multilinear algebra that started in tensor operations in chapter one and by the manifold learning and Hilbert space mapping in chapter two. Multilinear algebra will be explained in two subsections. The first subsection introduces non-linear algebra as expressed in non-Euclidean spaces using hyperbolic and elliptic geometry, defining curves as conic sections, such as circles, ellipses, parabolas and hyperbola. The section continues to solve a system of equations employing these non-linear shapes. The second subsection will introduce Differential geometry on Manifolds using coordinate-free approaches. This subsection mainly focuses on the preliminaries to understand exterior derivatives using differential forms and their projections in many subspaces of any higher space $\mathbb{R}^n$. This chapter explains another definition of a subset of tensors: differential forms as skew-symmetric (or anti-symmetric) covariant tensors. This leads to the third subsection in which tensors on Manifolds are explained. Then Multi-linear subspace learning (MSL) algorithms as a generalisation of the LSL in chapter two will be presented in the third section.

The chapter might look full of equations. However, the proofs, derivation and properties of each equation and consequent identities are omitted, although they are usually used in simplifying equations and computations. The aim is to familiarise the reader with the logical flow of these topics and how they build together and used in machine learning algorithms. Visualisations from interesting books have been used, and python libraries in which these mathematical operators are implemented are provided. These are usually used at the bottom of the stack of a machine learning algorithm. Understanding what is happening under the hood gives the reader a better opportunity to choose the suitable algorithms with the suitable parameters, and the ambitious reader might start developing these algorithms further or apply them better in new domains.

Motivation and Intuition

In the previous two chapters, we have seen examples of data that is usually collected in a 2-dimensional array as rows being samples or entities and features being columns. However, multiple datasets are usually interacting or correlated in latent variables; hence, multiple datasets need to be studied together in multi-way analysis. The following are generalisations of all ranks of tensors that data can appear in:

- A single temperature value as a scalar can be a rank-zero tensor.
- 1-way data for temperatures can be a vector as a rank-one tensor of a set of values for a given set of cities. Cities are the labels of the columns.
- 2-way data describing temperatures for different map locations can be a matrix as a rank-two tensor, where the first coordinate is latitude, the second is the longitude coordinate, and the value of the tensor is the temperature. We do not have labels for rows and columns in data science, but in tensors, we can. This is achieved by Pivot tables that can be created of two columns in a Pandas data frame.
- 3-way data for the same example can add a third dimension of time for the different temperature reading in the different locations at different times on the third mode.
- 2-way data in which columns describe features against entities in rows such as student marks in different subject matrices as rank two tensors, such that the student identity is in mode 1, and subjects in mode 2, and values are the marks. The school grade can also be added as another mode and so forth.
- Brain-Computer Interface (BCI) based on EEG signals are naturally multi-mode due to the data recording mechanism. For example, signals are recorded by multiple sensors (electrodes) in multiple trials and epochs for multiple subjects and with different tasks, conditions…, and so forth. This dataset can be represented in rank n tensors to enable multi-way multi-block data analysis techniques.
- Magnetic resonance imaging (MRI), functional MRI, PET, and MEG datasets are also naturally multi-mode. For example, A NIfTI file for a typical MRI scan store the voxel values in an array of numbers. The coordinates for a single voxel within a NIfTI image volume can be specified as a 3-dimensional index (x, y, z) or a 4-dimensional index (x, y, z, t) for time. Then the subject is another mode, then the aim of the experiment is another, the resolution and so forth. Similar datasets can be found at https://openneuro.org/.

- Examples from psychometrics are provided by (Kiers and Mechelen, 2001) in their overview of three-way component analysis techniques. The overview is a good introduction to three-way methods, explaining when to use three-way techniques rather than two-way (based on an ANOVA test), how to preprocess the data, guidance on choosing the rank of the decomposition and an appropriate rotation and methods for presenting the results.

Some datasets will be originally produced in tensor form but usually in multiple files, and a tensorisation step will always be required. Python notebooks accompanying this book, such as “tensorisation.ipynb” and “multi-wayExamples.ipynb”, contain sample tensorisation tailored to particular datasets and analysis requirements. These implementations are very simple for illustration purposes. Many opportunities for generalisations, quantisation, sampling and aggregation, and optimisation can be achieved. Tensorisation applied in these examples is so far artistic. It might be a talent that can be enhanced by practice and exposure to different datasets, preprocessing requirements and analysis requirements. The thesis in (Debals, 2017) presents a more formal introduction to tensorisation as suitable for BSS and clustering problems. The discussion is rather theoretical, leaving many implementation details to the programmer's creativity.

The opposite of tensorisation can be achieved, as chapter one explained. This happens when the data is naturally in tensor form, but the analyst needs to matricise or vectorise it for 2-way analysis. An N-mode tensor can be unfolded or matricized into a matrix in N ways for each mode. The n-mode matricization of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted as $\mathcal{X}_{(n)} \in \mathbb{R}^{I_n \times I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \ldots \times I_N}$ and is taken by keeping the nth mode intact and concatenating the slices of the rest of the modes into an extended matrix (Kolda and Bader, 2009).

Keeping these datasets in their original tensor-form structure and order as collected maintains the characteristics that will help estimate a manifold representative of the intrinsic structure that might get lost during the transformations such as vectorisation or matricisation. These transformations are meant to enable linear algebra analysis at the expense of losing the remarkable power of higher-order instruments that are not available in the lower order. This power is not a result of the generation of more data, as it is the result of the structure of the data in the higher order as it is naturally collected in different experiments and different domains and dimensionality (Smilde, Bro and Geladi, 2004). These heterogeneous and multi-aspect data are multimodal, measured using different sensors (experiments or data collection measures), and subject to different kinds of errors and uncertainties.

Some representation analysis and pre-processing might be needed for any dataset to enable multi-way analysis based on the problem definition and data collection methods. Tensor summation notation representation preserves the multilinearity of data as it comes in nature (Mangan, 2008).

If the complete tensor object in a full format is created in memory, it will require huge memory, and analysing a high dimensional space will suffer from the curse of dimensionality. However, since the non-empty elements of a tensor object are often highly correlated with neighbouring elements in most applications of interest, these tensor objects are highly constrained and limited to a subspace, a manifold of intrinsically low dimension (Zhang, Li and Wang, 2005). Methods of feature extraction, dimensionality reduction, or decomposition/factorisation transform a high-dimensional data set into a low-dimensional equivalent representation while retaining most of the variance and interactions among the elements, capturing the underlying structure or the actual physical phenomenon modelled. Pair-wise interactions do not capture the multi-way interactions by the cancellation of effects and lack of interpretation clarity. In Python code in “multi-wayExamples.ipynb”, examples of pivot tables in matrix form were difficult to interpret when more features were added. In comparison, the tensorisation captures every pair-wise and higher multi-way interaction.

In summary, an alternative definition of tensors is that they are like vectors that contain collections of components with magnitude and direction but in higher-dimensional space. Tensors can describe a function that houses the transformations of these components on a discrete high-dimensional grid capturing their interactions as the basis of these coordinates change (Charles Van Loan *et al.*, 2009). Not all multidimensional arrays are tensors. However, scalars, vectors, matrices or n-dimensional arrays are tensors and subject to multi-way analysis, when their structures can transform to different coordinates (reference axes or scale) than those they are measured on, maintaining their properties, such as invariance, covariance, contravariance, and some form of distance measure or neighbourhood as explained in chapter two.

3.1 N-D Arrays / Tensor definition

Motivation: Any given dataset in a matrix form is said to be high dimensional as the number of columns increases. Because the degree of freedom increases with the number of columns, for example, the Matrix-High-DImensional-DF.ipynb shows the Kaggle Cardiovascular Disease dataset's degree of freedom increases as we add a new column in the analysis. The concept is emphasised by the pair-wise plots and how different they are for any pair of columns. This illustrates that matricising a high-dimensional dataset loses important structure. The importance of having various indices in tensor forms rather than the two indices in the matrix form lies in capturing this multi-way structure or variance between each pair or more dimensions in the higher dimensional space. However, this easily creates the curse of dimensionality problem as N grows for an Nth-order tensor of size $(I_1, I_2, \dots, I_N)$, as the number of elements exponentially grows. As will be discussed in chapters two and three, this will be handled by learning subspaces that approximate the high-dimensional space with reasonable accuracy deterioration. Chapter four discusses other approaches to dividing the large-scale tensors into several lower-rank tensors that only capture the relevant information.

3.1.1 Tensor Definition

Having seen why stacking more columns in the same matrix is not good, we can now increase the number of indices and use the original data arrangements in tensors.

Tensors are defined to be higher-order matrices of N dimensions, so we need n indices to scan the elements of the tensor by its given shape vector or bounds. A tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_n}$ is an element of the tensor product of N vector spaces, such that the corresponding multi-dimensional array is $\mathcal{A}(i_1, \dots, i_n)$, $i_k \in [1, I_k]$ and:

- **Index:** i_k represents the index along the kth dimension, known as mode;
- **Order:** N is the order of A, i.e. the number of modes/dimensions/ways as iluustrated in Figure 1;
- **Size:** I_k represents the size along the kth mode, while the tensor shape is the vector $[I_1, I_2, \dots, I_N]$. The size of a tensor is the range of values that can be obtained for a dimension of the tensor. For example, a tensor $\mathcal{X} \in \mathbb{R}^{3\times 4\times 5\times 6}$ is of order 4, size 3 in mode-1, size 4 in mode-2, size 5 in mode-3 and size 6 in mode-4, and can have $3 \times 4 \times 5 \times 6 = 360$ values, and also denoted as having shape vector [3, 4, 5, 6].
- **Fibres:** The mode-j vectors of $\mathcal{A}$ are defined as the I_j dimensional vectors obtained from $\mathcal{A}$ by varying the index I_j, while keeping all the other indices fixed as shown in Figure 2 a, b, c and d.
- **Slices:** The mode-j slice of $\mathcal{A}$ is defined as an (N−1)th-order tensor obtained by fixing the mode-j index of $\mathcal{A}$ to be in $\mathcal{A}$ (:, ..., :, I_j, :, ..., :) as shown in Figure 2 e, f and g.
- **Blocks:** block matrices are represented by tensors of order four, as shown in Figure 3 *(a)*
- Hierarchical/nested
- **Tensor Product:** A tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_n}$ is composed from the outer product of N vectors: $\mathcal{A}$ = a(1) ∘ a(2) ∘ ... ∘ a(N), where $a^{(k)}$ is the k-dimensional vector

corresponding to the kth mode in the tensor. This is illustrated for a tensor of order three in Figure 3 *(b)* (Kolda and Bader, 2009; Lu, Plataniotis and Venetsanopoulos, 2014).

- **Rank-1 tensor:** is an n-way tensor object that can be strictly decomposed as the tensor product of n vectors. A tensor decomposition expresses a tensor in terms of a sequence of sums and products operating on simpler multi-way tensor objects. These methods will be explained in chapters three and four.

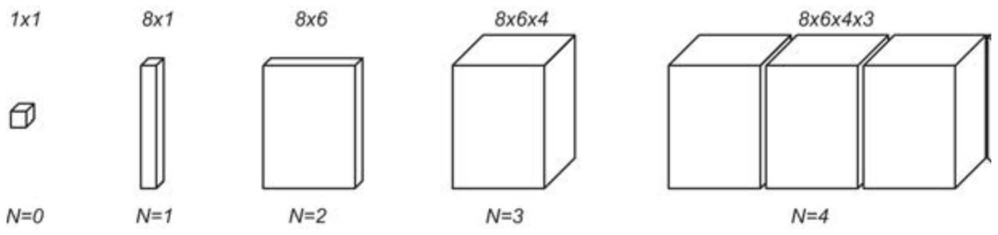


Figure 1: Illustration of tensors of order N = 0, 1, 2, 3, 4. (Lu, Plataniotis and Venetsanopoulos, 2014)

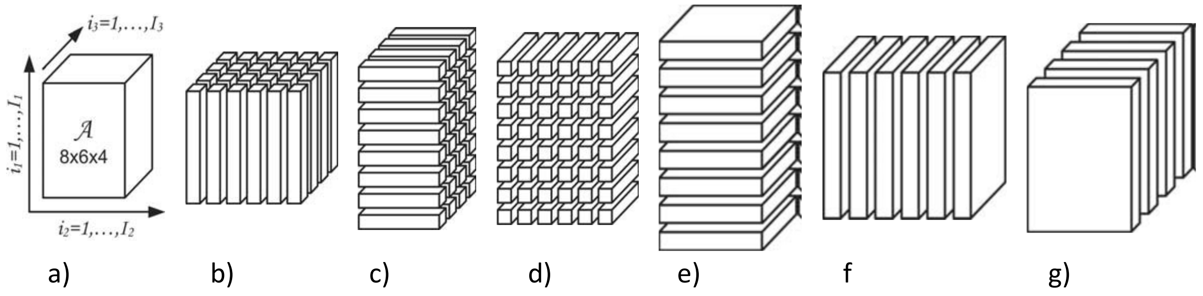


Figure 2: Illustration of the mode-n vectors/fibres and mode-n slices: (a) a tensor $\mathcal{A} \in \mathbb{R}^{8\times 6\times 4}$, (b) the mode-1 vectors or fibers, (c) the mode-2 vectors or fibers, and (d) the mode-3 vectors or fibers, (e) the mode-1 slices $\mathcal{A}_{i::}$, (f) the mode-2 slices $\mathcal{A}_{:i:}$, and (g) the mode-3 slices: $\mathcal{A}_{::i}$.

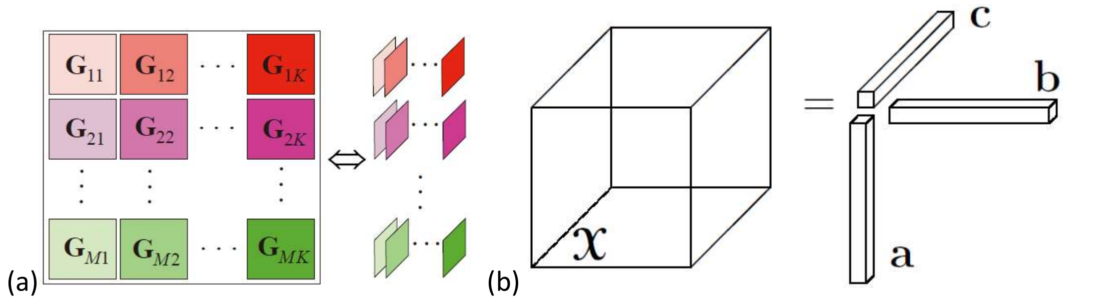


Figure 3: (a) A block matrix represented as a 4th-order tensor (b) Rank-one third-order tensor, $\mathcal{X} = a \circ b \circ c$.

3.1.2 Tensor Indexing

In general, there are two possibilities for the representation of the tensors and the tensorial equations:

1) The direct/ Einstein (symbolic, coordinate-free) notation (The Python NumPy function: np.einsum uses Einstein notation to express tensor contractions). This notation uses superscripts and subscripts to represent an operation concisely using known information about the tensors involved, such as their dimensionality and shape/size. This notation is suitable for Riemannian space, a smooth manifold or Minkowski space, which combines three-dimensional Euclidean space and time into a four-dimensional manifold where the space-time interval between any two events is independent of the inertial frame of reference in which they are recorded. This is visualised in https://youtu.be/CliW7kSxxWU.

A summation index, when repeated twice or more across the tensors involved in the operation, such as $a_i x_i$ means i is the summation index of all possible values for vectors a and x. This can be written as $a_i x^i$, indicating that a is a covariant vector (can be thought of as row vector with lower indices) and x is a contravariant vector/covector (can be thought of as a column vector with upper indices). This means they transform covariantly or contravariantly, with respect to change of basis. The variance will be explained in chapter two, and change of basis and more examples will be explained in chapter three. For now, a change of basis is a change of scale, unit of measurement, or coordinate system. The upper index for a column vector goes from up to down, and the lower index for a row vector goes from left to right. $A_{ji} x_i$ means i is the summation index, and j is a second index to matrix A. This is more concise than writing $\sum_i \mathrm{A}_{ji} \mathrm{x}_i$. The summation index, also called the dummy index, should have the same range of values for tensors involved in the operation. The non-dummy indices are called free indices and take their ranges from the shape/size of the tensor and are used once in a term. The free index is fixed in the operation where a dummy variable is used, such that $A_{ji} x_i$ for j = 3, and i takes all available values in its range, such as $A_{ji} x_i = A_{31} x_1 + A_{32} x_2 + A_{33} x_3$.

Some examples are as follows (more explanation using tensor notation will follow):

- **Vector Space V with elements v:** $v^i e_i$ written in terms of basis vectors. The basis elements are written with lower indices, and components or vector coefficients are written with upper indices.
- **Inner product:** $a \cdot x = a_i x^i$

- **Kronecker delta:** given the vector space V, its dual space (dual will be explained in chapter three), is V* has dual basis e_i^*, the Kronecker delta is defined as: $< e_i^*, e_j > = e^i(e_j) = \delta_{ij} = \begin{cases} 1 & if\ i = j \\ 0 & if\ i \neq j \end{cases} = \delta_j^i$, here the dual basis elements are written with upper indices. More on this will be explained in chapter three.
- **Cross product:** $a \times x = \sum_{i=1}^{N} a^i e_i \times \sum_{j=1}^{N} x^j e_j = a^i e_i \times x^j e_j = (a^i x^j)(e_i \times e_j) = a^i x^j \varepsilon_{ij}^k e_k$, where $(e_i \times e_j) = \varepsilon_{ij}^k e_k$ and $\varepsilon_{ij}^k = \delta^{kl} \varepsilon_{lij}$, $\varepsilon_{ij}^k$is the permutation component as the Levi-Civita symbol, and δ^{kl} is the Kronecker delta of the indices kl (1 when equal and zero otherwise).
- **Outer Product:** $A_j^i = a^i x_j = (ax)_j^i$, i and j are different and are not eliminated by multiplication
- **Matrix-Vector Multiplication:** $u = A \cdot x \rightarrow u_i = (Ax)_i = \sum_j A_{ij}\, x^j \rightarrow u^i = A_j^i x^j$
- **Matrix-Matrix Multiplication:** $C = A \cdot B \rightarrow C_{ik} = (AB)_{ik} = \sum_j A_{ij}\, B_k^j \rightarrow C_k^i = A_j^i B_k^j$

2) The index (component) notation (NumPy arrays accommodate higher dimensional tensors, same as tensorly package and scikit-tt package). This notation is suitable for the Euclidean space and uses subscripts only.

3.1.3 Tensor Transformations

The mode-n matricization,$mat(\mathcal{X})_n$, also defined as mode-n unfolding $\mathcal{X}_{[n]}$of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \ldots \times I_N}$, is defined as follows:

$$mat(\mathcal{X})_n = \mathcal{X}_{[n]} = \mathcal{X}_{mn} \in \mathbb{R}^{I_\text{n} \times I_1 I_2 \ldots I_{\text{n}-1} I_{\text{n}+1} \ldots I_N}$$

Where the matrix element is indexed with two indices (i, j), the first is from 1 to I_n, and the second j from 1 to:

$$\prod_{k=1, k \neq n}^{N} i_k$$

The mode-n vectorisation of a tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \ldots \times I_N}$, is defined as follows:

$$vec(\mathcal{X})_n = \mathcal{X}_{vn} \in \mathbb{R}^{I_\text{n} I_1 I_2 \ldots I_{\text{n}-1} I_{\text{n}+1} \ldots I_N}$$

Which is a mode-n matricization, followed by vertical stacking of the matrix in one column vector. For example, given $\mathcal{X} \in \mathbb{R}^{2\times2\times2} = \left[\begin{bmatrix}1 & 3\\2 & 4\end{bmatrix}\begin{bmatrix}5 & 7\\6 & 8\end{bmatrix}\right]$

mode-1 matricization of $\mathcal{X} = \begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\end{bmatrix}$

mode-1 vectorisation of $\mathcal{X} = \begin{bmatrix}1\\5\\2\\6\\3\\7\\4\\8\end{bmatrix}$.

An essential property of vectorisation of two tensors $\mathcal{X}, \mathcal{Y}$, is that $vec(\mathcal{X})^T vec(\mathcal{Y}) = trace(\mathcal{X}^T\mathcal{Y})$.

The n-unfolding of a tensor, $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times I_3 \dots\times I_N}$ creates a matrix that is defined as follows:

$$\mathcal{X}_{<n>}(i_1 \dots i_n, i_{n+1} \dots i_N) = \mathcal{X}(i_1 i_2 \dots i_N)$$

where the first n indices enumerate the rows of $\mathcal{X}_{<n>}$, and the last N - n indices for its columns. Then $\mathcal{X}_{<n>} \in \mathbb{R}^{m\times l}$, such that $m = \prod_{j=1}^{n} I_j$ and $l = \prod_{j=n+1}^{N} I_j$.

The tensor n-rank of a tensor, $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times I_3 \dots\times I_N}$ namely $rank_n(\mathcal{X})$ is the column rank of the n-unfolding $\mathcal{X}_{<n>}$ such that it computes the number of dimensions in the vector span by n-mode of $\mathcal{X}$.

The tensor rank, namely rank($\mathcal{X}$), is defined to be the minimum number of the sum of rank-one tensor that can exactly factorise tensor $\mathcal{X}$.

3.1.4 Tensor Element-wise operations

Element-wise tensor operations such as addition, subtraction, Hadamard product and division are implemented in NumPy and presented in the python notebook (xxxx) along with other examples used below. The transposition of a tensor is as follows: For a given tensor $\mathcal{X} \in \mathbb{R}^{I_1, I_2, I_3 \dots, I_N}$, the transpose is defined as: $\mathcal{X}^T \in \mathbb{R}^{I_N, I_{N-1}, I_{N-2} \dots, I_1}$.

3.1.5 Tensor Products

The element-wise product followed by a summation is the scalar product (a generalisation of the **inner or dot product**) of two same-sized tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ results in a scalar value, which is defined as follows:

$$z = \langle \mathcal{X}, \mathcal{Y}\rangle = \langle vec(\mathcal{X}), vec(\mathcal{Y})\rangle = \sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\ldots\sum_{i_n=1}^{I_N}\mathcal{X}(\mathrm{i}_1,\mathrm{i}_2,\ldots\mathrm{i}_n).\mathcal{Y}(\mathrm{i}_1,\mathrm{i}_2,\ldots\mathrm{i}_n) = x^T y$$

Where y is the vectorised form of $\mathcal{Y}$, and x is vectorised $\mathcal{X}$. Similarly, the Frobenius norm is extended in the higher dimension as:

$\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X}\rangle}$ when the origin is zero, it is the length, but a distance between $\mathcal{X}$ and $\mathcal{Y}$ tensors: $dist(\mathcal{X}, \mathcal{Y}) = \|\mathcal{X} - \mathcal{Y}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{Y}\rangle}$

The outer product for $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_P}$ with $\mathcal{Y} \in \mathbb{R}^{J_1\times J_2\times\ldots\times J_Q}$ results in a tensor $\in \mathbb{R}^{I_1\times I_2\times\ldots\times I_P\times J_1\times J_2\times\ldots\times J_Q}$ computed as:

$\mathcal{C} = \mathcal{X} \circ \mathcal{Y}$ such that $c_{i_1,\ldots,i_p,j_1,\ldots,j_Q} = x_{i_1,\ldots,i_p} y_{j_1,\ldots,j_Q}$

The contracted product for $\mathcal{X} \in \mathbb{R}^{I_1\times\ldots\times I_M\times J_1\ldots\times J_P}$ with $\mathcal{Y} \in \mathbb{R}^{I_1\times\ldots\times I_M\times K_1\times\ldots\times K_Q}$ with equal size along the first M modes results in a tensor $\in \mathbb{R}^{J_1\ldots\times J_P\times K_1\ldots\times K_Q}$ computed by:

$$c_{j_1\ldots,j_P,k_1\ldots,k_Q} = \langle \mathcal{X}, \mathcal{Y}\rangle_{1,\ldots,M,1,\ldots,M(j_1\ldots,j_P,k_1\ldots,k_Q)} = \sum_{i_1=1}^{I_1}\ldots\sum_{i_M=1}^{I_M} x_{i_1,\ldots,i_M,j_1\ldots,j_P} y_{i_1,\ldots,i_M,k_1k_1,\ldots,k_Q}$$

The results' entries are the summing of the product of the two input tensors along with the common indices. The inner/dot product is a special case of the contracted product when all modes are common, producing a scalar only as $c = \langle \mathcal{X}, \mathcal{Y}\rangle_{1,\ldots,M,1,\ldots,M}$. Similarly, the outer product is a special case of the contracted product when no modes are common such that for $\mathcal{X} \in \mathbb{R}^{J_1\ldots\times J_P}$ with $\mathcal{Y} \in \mathbb{R}^{K_1\times\ldots\times K_Q}$: $\mathcal{C} = \mathcal{X} \circ \mathcal{Y} = \langle \mathcal{X}, \mathcal{Y}\rangle_{0,0(j_1\ldots,j_P,k_1\ldots,k_Q)}$.

The Tensor products are extensions to matrix multiplication that apply the dot product in a given order of the inputs (along axis 1/rows of the first matrix and axis 0/columns of the second input that need to be conforming) to produce an output on the intersecting index in the output relative to the input indices used. The extension to higher dimensions requires specification of the conforming axis to apply the product on from the inputs, which will decide the position in the output where the result will be placed. The tensor Dot product can be used in different ways, and it is implemented as a function in the NumPy package. Below is the definition, followed by several examples.

Tensor n-mode Products: for $\mathcal{X} \epsilon \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ with matrix U $\epsilon\ \mathbb{R}^{J\times I_n}$, where n is a mode from the N modes of the first tensor, results in tensor $\epsilon\ \mathbb{R}^{I_1\times I_2\times\ldots\times I_{n-1}\times \mathrm{J}\times I_{n+1}\ldots\times I_N}$, and is computed as follows:

$$(\mathcal{X} \times_n U)_{\mathrm{i}_1,\,\mathrm{i}_2,\,\ldots\,\mathrm{i}_{n-1},\mathrm{j},\,\mathrm{i}_{n+1},\ldots,\ \mathrm{i}_N} = \sum_{i_n=1}^{I_N} x(\mathrm{i}_1,\mathrm{i}_2,\ldots\mathrm{i}_N)u_{ji_n}$$

The n-mode product of a tensor $\mathcal{X}$ with a matrix U is related to a change of basis in the case when a tensor defines a multilinear operator. This is equivalent to pre multiplying each mode-n vector of $\mathcal{X}$ by U. Thus, the mode-n product above can be written using the mode-n unfolding as $\mathcal{C}_{(n)}$ = U $\mathcal{X}_{(n)}$ where each entry of $\mathcal{C}$ is defined as the sum of products of corresponding entries in $\mathcal{X}$ and **U**. This is also called tensor contraction because the resulting tensor dimension is the sum of the dimensions of the original two tensors minus the dimension of the contraction.

The Numpy package tensordot function sums the products of both input tensors' elements (components) over the axes specified by the third argument "axis", which designates the axes along which to perform the reduction (multiplication/addition). When axis = 0, each instance of both input tensors is a scalar input to the tensor product $\otimes$, producing a shape that concatenates input shapes. When axis = 1, or (1, 1), each instance from both inputs is a vector that is used in the tensor dot product. When axis = 2, which is the default, it will do double tensor contractions, producing a scalar. To sum over more than one axis, the third argument specifies one sequence to apply on both inputs or two sequences of the same length to apply on each of the inputs in order, indicating which axis to sum over as one and others as zero. For an N-D tensor, if we set axes argument = N, then we will multiply both tensors element-wise and then sum all values to get a single scalar result.

For the same vectors defined above: $v = \begin{bmatrix}10\\2\\-6\end{bmatrix}, u = \begin{bmatrix}-3\\0\\-2\end{bmatrix}$, a mode-0 product is defined as $vu^T = \begin{bmatrix}10\\2\\-6\end{bmatrix} \times [-3 \quad 0 \quad -2] = \begin{bmatrix}-30 & 0 & -20\\-6 & 0 & -4\\18 & 0 & 12\end{bmatrix}$, which is the cross-product defined earlier.

The Numpy package does not have an unfold function in one implementation, but the Tensorly package has. This can be produced in Python as a mode-0 tensordot operation. Mode-0 means the vectors along the zeroth axis in both inputs. Since the first vector is a column vector, then each element is a vector to apply the dot product with the zeroth axis vectors of the second vector. Since the second vector is a row vector, then the mode-0 returns this row vector every time. This will be a scalar multiplication between each mode-0 from the first vector (scalar) and the only row from mode-0 from the second vector.

A mode-1 product is defined as $v^T u$ =$[10 \quad 2 \quad -6] \times \begin{bmatrix}-3\\0\\-2\end{bmatrix}$= -18, which is the dot product defined earlier. This can be produced in Python as mode-1 tensordot operation, which applies the dot product on the first axis of v, which is a row vector now after transpose, and the first axis of u, which u, which is the column vector. In vectors, there are no more modes to attempt except 0 and 1.

For matrices example using tensordot, the matrix multiplication example of gas distribution presented earlier, we set the axis argument =1, or (1, 0), specifying clearly that vectors along axis=1 (which is each instance of (2,3) in this example) from first input is used as input to the dot product with vectors along last one axis from second input: so that will give us four vectors of length 3, and perform the dot product as follows:

$$\mathrm{P} \in \mathbb{R}^{2\times3} \times_1 \mathrm{S} \in \mathbb{R}^{3\times4} = \begin{bmatrix}0.5 & 0.2 & 0.3\\0.0 & 0.4 & 0.6\end{bmatrix} \times \begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix}$$

$$= \begin{bmatrix} [0.5 \quad 0.2 \quad 0.3\,] \times \begin{bmatrix}0.4\\0.0\\0.0\end{bmatrix} & [0.5 \quad 0.2 \quad 0.3\,] \times \begin{bmatrix}0.6\\0.7\\0.5\end{bmatrix} & [0.5 \quad 0.2 \quad 0.3\,] \times \begin{bmatrix}0.0\\0.3\\0.0\end{bmatrix} & [0.5 \quad 0.2 \quad 0.3\,] \times \begin{bmatrix}0.0\\0.0\\0.5\end{bmatrix} \\ [0.0 \quad 0.4 \quad 0.6\,] \times \begin{bmatrix}0.4\\0.0\\0.0\end{bmatrix} & [0.0 \quad 0.4 \quad 0.6\,] \times \begin{bmatrix}0.6\\0.7\\0.5\end{bmatrix} & [0.0 \quad 0.4 \quad 0.6\,] \times \begin{bmatrix}0.0\\0.3\\0.0\end{bmatrix} & [0.0 \quad 0.4 \quad 0.6\,] \times \begin{bmatrix}0.0\\0.0\\0.5\end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix}0.20 & 0.59 & 0.06 & 0.15\\0.00 & 0.58 & 0.12 & 0.30\end{bmatrix} \in \mathbb{R}^{2\times4}$$

To apply tensordot across axis 2 on both inputs, we take the last two axes of P, with the first two axes of S, it will return each scalar and multiply with each scalar in the second input, but require that the number of elements in both input tensors matches. Therefore it will not work in the above example, but will work on similar shape inputs as below, and sum all products:

$$\mathrm{P} \times_2 \mathrm{S} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.0 & 0.4 & 0.6 \end{bmatrix} \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} = 0.78$$

To apply tensordot across the axis 0 on both inputs, it will perform the dot product for every instance of P with every instance of S, which will result in an expansion in the form of the cross product on the higher dimension and produce an output shape that concatenates the shape of both inputs.

$$\mathrm{P} \times_0 \mathrm{S} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.0 & 0.4 & 0.6 \end{bmatrix} \in \mathbb{R}^{2\times3} \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} \in \mathbb{R}^{2\times3}$$

$$= \begin{bmatrix} \begin{bmatrix} 0.5 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} & 0.2 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} & 0.3 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} 0.0 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} & 0.4 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} & 0.6 \times \begin{bmatrix} 0.4 & 0.6 & 0.0 \\ 0.0 & 0.7 & 0.3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 0.2 & 0.3 & 0.0 \\ 0.0 & 0.35 & 0.15 \end{bmatrix} & \begin{bmatrix} 0.08 & 0.12 & 0.00 \\ 0.00 & 0.14 & 0.06 \end{bmatrix} & \begin{bmatrix} 0.12 & 0.18 & 0.00 \\ 0.00 & 0.21 & 0.09 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} & \begin{bmatrix} 0.16 & 0.24 & 0.00 \\ 0.00 & 0.28 & 0.12 \end{bmatrix} & \begin{bmatrix} 0.24 & 0.36 & 0.00 \\ 0.00 & 0.42 & 0.18 \end{bmatrix} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{2\times3\times2\times3}$$

To reproduce the example in Figure 4 using NumPy tensordot, while having different shapes for the two input tensors $\mathcal{A} \in \mathbb{R}^{8x6x4}$ and $\mathrm{U} \in \mathbb{R}^{3\times8}$, the axes argument can be a sequence of (0, 1). This will produce dot products between mode-1 vectors of $\mathcal{A}$ ($\in \mathbb{R}^{8}$) by $\mathrm{U} \in \mathbb{R}^{3\times8}$ (which is a projection to a higher dimension) to obtain a vector $\mathrm{b} \in \mathbb{R}^{3}$, as the differently shaded vector indicates in the right of the figure, producing an output $\in \mathbb{R}^{3\times6\times4}$. The tensordot NumPy implementation produced the output shape $\in \mathbb{R}^{6\times4\times3}$, because of reversing the indices.

The tensorly python package has a tensor algebra mode dot function as “tenalg.mode_dot”, which only contracts a specified axis in the mode argument and concatenates all other shape vectors in inputs to create the output shape. So, only mode =0 can work in **Error! Reference source not found.** example, as axis=0 in the first input has the same shape “8” as axis=1 in the second input, and the output is of shape $\in \mathbb{R}^{3\times6\times4}$ as intended in (Kolda and Bader, 2009; Lu, Plataniotis and Venetsanopoulos, 2014).

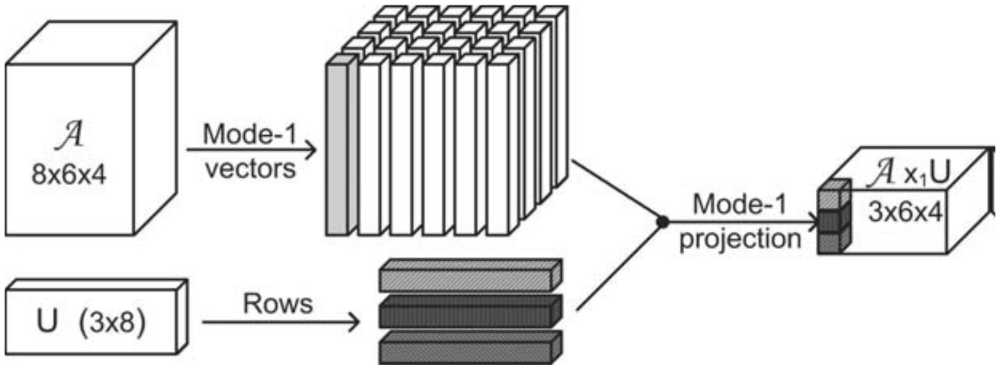


Figure 4: Visual illustration of the mode-n (mode-1) multiplication

The Kronecker product for $A \in \mathbb{R}^{I\times J}$with matrix U $\in \mathbb{R}^{K\times L}$results in a block tensor $\in \mathbb{R}^{IKxJL}$ computed by a scalar multiplication of every element in the first tensor with the full tensor as a block of the second tensor:

$$A \otimes U = \begin{bmatrix} a_{1,1}U & \cdots & a_{1,J}U \\ \vdots & \ddots & \vdots \\ a_{I,1}U & \cdots & a_{I,J}U \end{bmatrix}$$

It can be computed as columnwise Kronecker product as follows:

$$A \otimes U = \left[a_1 \otimes u_1, a_1 \otimes u_2,\ a_1 \otimes u_3\ \ldots\ a_1 \otimes u_L,\ a_2 \otimes u_1, \ldots\ a_J \otimes u_{L-1}, a_J \otimes u_L\right]$$

$$= [vec(u_1 a_1^T), vec(u_2 a_1^T)], vec(u_3 a_1^T), \ldots\ vec(u_L a_1^T), vec(u_1 a_2^T), \ldots, vec\left(u_{L-1} a_J^T\right), vec\left(u_L a_J^T\right)$$

The mixed-product property of Kronecker product states that if X, Y, A and B are conformable matrices whose dimensions are suitable for multiplications such that the matrix product XY and AB can be formed, then we have $(X \otimes A)(Y \otimes B) = XY \otimes AB$

Kronecker product is defined for higher order tensors such as $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ to yield a tensor $\in \mathbb{R}^{I_1 J_1 \times \ldots \times I_N J_N}$.

This is implemented in the Numpy package as “kron” function, producing a composite array made of blocks of the second tensor scaled by the first tensor. For the same previously defined v and u vectors:

$$\mathrm{v}\otimes\mathrm{u} = \begin{bmatrix}10\\2\\-6\end{bmatrix}\otimes\begin{bmatrix}-3\\0\\-2\end{bmatrix} = [10\times[-3 \quad 0 \quad -2] \quad 2\times[-3 \quad 0 \quad -2] \quad -6\times[-3 \quad 0 \quad -2]]$$

$$= [-30 \quad 0 \quad -20 \quad -6 \quad 0 \quad -4 \quad 18 \quad 0 \quad 12]$$

For the gas distribution matrices example:

$$\mathrm{P}\in\mathbb{R}^{2\times3}\otimes\mathrm{S}\in\mathbb{R}^{3\times4} = \begin{bmatrix}0.5 & 0.2 & 0.3\\0.0 & 0.4 & 0.6\end{bmatrix}\otimes\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} =$$

$$= \begin{bmatrix} 0.5\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} & 0.2\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} & 0.3\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} \\ 0.0\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} & 0.4\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix} & 0.6\times\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\0.0 & 0.7 & 0.3 & 0.0\\0.0 & 0.5 & 0.0 & 0.5\end{bmatrix}\end{bmatrix}$$

$$\begin{bmatrix}0.20 & 0.30 & 0.00 & 0.00 & 0.08 & 0.12 & 0.00 & 0.00 & 0.12 & 0.18 & 0.00 & 0.00\\0.00 & 0.35 & 0.15 & 0.00 & 0.00 & 0.14 & 0.06 & 0.00 & 0.00 & 0.21 & 0.09 & 0.00\\0.00 & 0.25 & 0.00 & 0.25 & 0.00 & 0.10 & 0.00 & 0.10 & 0.00 & 0.15 & 0.00 & 0.15\\0.00 & 0.00 & 0.00 & 0.00 & 0.16 & 0.24 & 0.00 & 0.00 & 0.24 & 0.36 & 0.00 & 0.00\\0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.28 & 0.12 & 0.00 & 0.00 & 0.42 & 0.18 & 0.00\\0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.20 & 0.00 & 0.20 & 0.00 & 0.30 & 0.00 & 0.3\ 0\end{bmatrix}$$
$$\in\mathbb{R}^{6\times12}$$

The Khatri-Rao product for $A\,\epsilon\,\mathbb{R}^{I\times K}$ with matrix U $\epsilon\,\mathbb{R}^{J\times K}$ requires having equal number of columns in input tensors such that a columnwise Kronecker product of the input tensors results in a tensor that $\epsilon\,\mathbb{R}^{IJ\times K}$

$$A\odot U = [a_1\otimes u_1 \quad a_2\otimes u_2 \quad \dots \quad a_K\otimes u_k]$$

Tensorly package has a Khatri-Rao product function, but Numpy does not have one, but a fully vectorised version can be implemented as follows:

```
def khatri_rao(a, u):
    c = a[...,:,np.newaxis,:] * u[...,np.newaxis,:,:]
    # collapse the first two axes
    return c.reshape((-1,) + c.shape[2:])
```

Khatri-Rao product is defined for each mode in the higher-order tensors such as given $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times \ldots \times J_N}$ where for which $I_n = J_n$, then $\mathcal{A} \odot_n \mathcal{U} = \mathcal{C} \epsilon \mathbb{R}^{I_1 J_1 \times \ldots \times I_{n-1} J_{n-1} \times I_n \times I_{n+1} J_{n+1} \times \ldots \times I_N J_N}$ with entries $\mathcal{C}(:, \ldots, :, i_n, :, \ldots, :) = \mathcal{A}(:, \ldots, :, i_n, :, \ldots, :) \otimes \mathcal{U}(:, \ldots, :, i_n, :, \ldots, :)$.

The scientific Python package “SciPy” implements the Khatri-Rao product “scipy.linalg.khatri_rao” implements it as the Kronecker product of every column of the first tensor by the second tensor. It does not work for vectors. For the gas distribution matrix example, the number of columns of the arrays should match, so a transpose on the second matrix results in the following:

$$A \in \mathbb{R}^{2\times3} \odot U^T \in \mathbb{R}^{4\times3} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.0 & 0.4 & 0.6 \end{bmatrix} \otimes \begin{bmatrix} 0.4 & 0.0 & 0.0 \\ 0.6 & 0.7 & 0.5 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.5 \end{bmatrix} =$$

$$\begin{bmatrix} [0.5 \quad 0.2 \quad 0.3] \otimes \begin{bmatrix} 0.4 & 0.0 & 0.0 \\ 0.6 & 0.7 & 0.5 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.5 \end{bmatrix} \\ [0.0 \quad 0.4 \quad 0.6] \otimes \begin{bmatrix} 0.4 & 0.0 & 0.0 \\ 0.6 & 0.7 & 0.5 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.5 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 0.20 & 0.00 & 0.00 \\ 0.30 & 0.14 & 0.15 \\ 0.00 & 0.06 & 0.00 \\ 0.00 & 0.00 & 0.15 \\ 0.00 & 0.00 & 0.00 \\ 0.00 & 0.28 & 0.30 \\ 0.00 & 0.12 & 0.00 \\ 0.00 & 0.00 & 0.30 \end{bmatrix}$$

Direct sum of tensors is defined for the Nth-order tensors $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times \ldots \times J_N}$ yields a tensor $\mathcal{C} \epsilon \mathbb{R}^{(I_1+J_1) \times \ldots \times (I_N+J_N)}$,

with entries $\mathcal{C}(k_1, \ldots, k_N) = \mathcal{A}(k_1, \ldots, k_N)$ if $1 \leq k_n \leq I_n, \forall n$,

$\mathcal{C}(k_1, \ldots, k_N) = \mathcal{U}(k_1 - I_1, \ldots, k_N - I_N)$ if $I_n < k_n \leq I_n + J_n, \forall n$,

and $\mathcal{C}(k_1, \ldots, k_N) = 0$, otherwise (see Figure 5(a)).

Partial mode-n sums for tensors is defined for the Nth-order tensors $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times \ldots \times J_N}$ for which $I_n = J_n$, then $\mathcal{A} \oplus_n \mathcal{U} = \mathcal{C} \epsilon \mathbb{R}^{(I_1+J_1) \times \ldots \times (I_{n-1}J_{n-1}) \times I_n \times (I_{n+1}J_{n+1}) \times \ldots \times (I_N J_N)}$ with entries $\mathcal{C}(:, \ldots, :, i_n, :, \ldots, :) = \mathcal{A}(:, \ldots, :, i_n, :, \ldots, :) \oplus \mathcal{U}(:, \ldots, :, i_n, :, \ldots, :)$ (see Figure 5(c)).

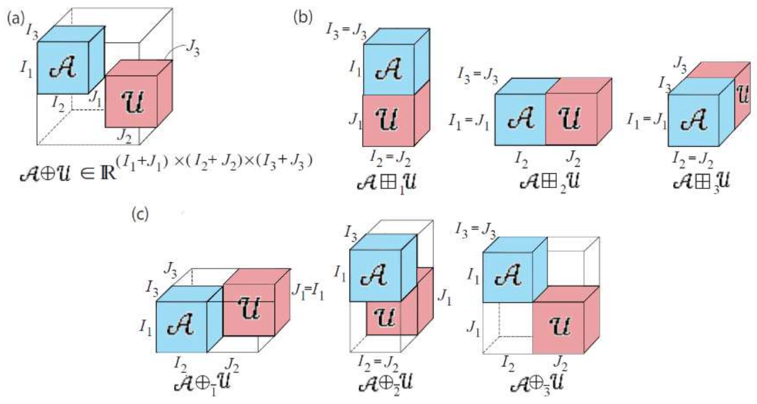

(a)
(b)
(c)
$\mathcal{A}\oplus\mathcal{U} \in \mathbb{R}^{(I_1+J_1)\times(I_2+J_2)\times(I_3+J_3)}$
$\mathcal{A}\boxplus_1\mathcal{U}$
$\mathcal{A}\boxplus_2\mathcal{U}$
$\mathcal{A}\boxplus_3\mathcal{U}$
$\mathcal{A}\oplus_1\mathcal{U}$
$\mathcal{A}\oplus_2\mathcal{U}$
$\mathcal{A}\oplus_3\mathcal{U}$

Figure 5: Illustration of the direct sum, partial direct sum and concatenation operators of two 3rd-order tensors. (a) Direct sum (b) Concatenations along mode-1,2,3. (c) Partial (mode-1, mode-2, and mode-3) direct sum.

Concatenation of Nth-order tensors along mode-n of tensors $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times \ldots \times J_N}$, for which $I_m = J_m, \forall m \neq n$ yields a tensor $\mathcal{C} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_{n-1} \times (I_n+J_n) \times I_{n+1} \times \ldots \times I_N} = \mathcal{A} \boxplus_n \mathcal{U}$ with subtensors $\mathcal{C}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) = \mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) \oplus \mathcal{U}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N)$

as illustrated in Figure 5(b).

3D convolution for two 3rd-order tensors $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times J_2 \times J_3}$, yields a tensor $\mathcal{C} \epsilon \mathbb{R}^{(I_1+J_1-1) \times (I_2+J_2-1) \times (I_3+J_3-1)} = \mathcal{A} * \mathcal{U}$, with entries: $\mathcal{C}(k_1, k_2, k_3) = \sum_{j_1} \sum_{j_2} \sum_{j_3} \mathcal{U}(j_1, j_2, j_3)\, \mathcal{A}(k_1 - j_1, k_2 - j_2, k_3 - j_3)$ as illustrated in Figure 6 for 2D convolution.

Partial (mode-n) convolution for two tensors $\mathcal{A} \epsilon \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{U} \epsilon \mathbb{R}^{J_1 \times \ldots \times J_N}$, yields a tensor $\mathcal{C} \epsilon \mathbb{R}^{I_1 J_1 \times \ldots \times I_{n-1} J_{n-1} \times (I_n + J_n - 1) \times I_{n+1} J_{n+1} \times \ldots \times I_N J_N} = \mathcal{A} \boxdot_n \mathcal{U}$, the subtensors of which are $\mathcal{C}(k_1, \ldots, k_{n-1}, :, k_{n+1}, \ldots, k_N) = \mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) * \mathcal{U}(j_1, \ldots, j_{n-1}, :, j_{n+1}, \ldots, j_N)$, where $k_1 = i_1 j_1$, …., and $k_N = i_N j_N$.

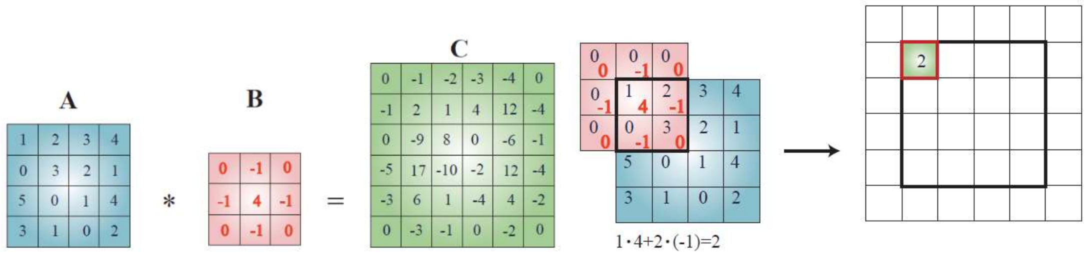


Figure 6: Illustration of the 2D convolution operator, performed through a sliding window operation along both the horizontal and vertical index (Cichocki et al., 2016, p. 1).

3.1.6 Orthonormal Tensor

Given $\mathcal{A} \in \mathbb{R}^{N}$, such that N=($N_1$, N_2, …. N_d)^T, we split $\mathcal{A}$ into two subtensors, the first is $\mathcal{A}_1 \in \mathbb{R}^{N'}$, such that N'=($N_{k1}$, N_{k2}, …. N_{ke})^T, and the second is $\mathcal{A}_2 \in \mathbb{R}^{N'}$, such that N''=(N_{l1}, N_{l2}, …. N_{lf})^T, such that e+f =d. $\mathcal{A}$ is orthonormal with respect to the $\mathcal{A}_1$ if the N' -matricization of $\mathcal{A}$ ($\mathcal{A}_{N'} = mat(\mathcal{A})_{N'}$) satisfies

$\mathcal{A}_{N'} \cdot (\mathcal{A}_{N'})^T = \mathcal{A}_{N'} \cdot (\mathcal{A}_{N''})^T = I \in \mathbb{R}^{N' \times N'}$. It is necessary that:

$$\prod_{i=1}^{e} N_{ki} \leq \prod_{i=1}^{f} N_{li}$$

This is illustrated in Figure 7 using a graphical notation similar to the Tensor Networks notation that will be explained in chapter four. $\mathcal{A}$ is orthonormal with respect to $\mathcal{A}_2$ if the N'' -matricization of $\mathcal{A}$ ($\mathcal{A}_{N''} = mat(\mathcal{A})_{N''}$) satisfies $\mathcal{A}_{N''} \cdot (\mathcal{A}_{N''})^T = \mathcal{A}_{N'} \cdot (\mathcal{A}_{N'})^T = I \in \mathbb{R}^{N'' \times N'}$.

Knowing that a tensor is orthonormal makes its decomposition easier, as explained in chapter two.

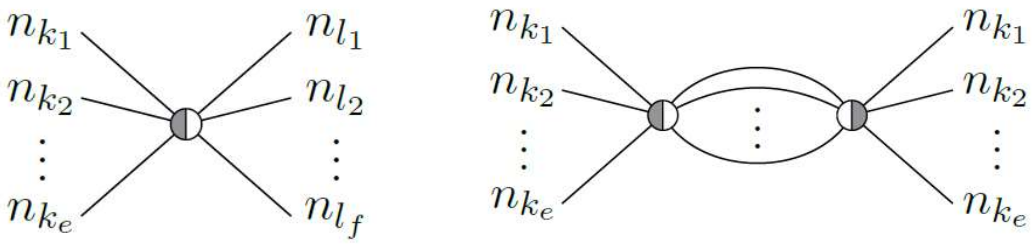


Figure 7: Orthonormal tensors: (a) Graphical representation of a tensor $\mathcal{A} \in \mathbb{R}^{N}$, which

is orthonormal with respect to the set N'=(N_{k1}, N_{k2}, ... N_{ke})^T ⊂ N. (b) Tensor multiplication of $\mathcal{A}$ and $\mathcal{A}^T$. The result is the identity tensor I ∈ $\mathbb{R}^{N'\times N'}$ (Gelß, 2017).

3.3 Motivational Problem

In python notebook, tensorisation.ipynb, the Kaggle Heart disease factors dataset is tensorised using both the direct columns from the dataset and the PCA as a 2-way dimensionality reduction approach from which the first three principal components created a three-way tensor. Several tensorisation examples are demonstrated, with many more used in the literature mentioned. Then, various regression and compression approaches are demonstrated. For a small dataset, it was easy to create the full-format tensor space. Other examples attempted to allocate space that was not available in the testing machine. Two approaches can solve this dimensionality curse: 1) Sparse tensor structures are required, and 2) creating the decomposed tensor components and working on them directly without creating the complete tensor. These methods are embedded in the tensorisation step by various approaches.

3.3.1 Regression/Classification

Regression is a supervised learning algorithm that estimates the decision line equation parameters (weights) w from a prelabelled dataset X with y label. Either analytically or using the gradient descent algorithm, the aim is to minimise the error ε by adjusting the weights using matrix form. $Xw + \varepsilon = y$, where y is the target/response/dependent variable in the dataset such that y ∈ $\mathbb{R}^{N}$, and N is the number of samples (data points or rows), X is the remaining columns in the dataset that represent the features/predictors/independent variables such that X ∈ $\mathbb{R}^{N\times d+1}$ and w is the coefficients or weights of each feature such that w ∈ $\mathbb{R}^{d+1}$, and d is the number of features (columns) or the dimensionality. We add one extra dimensionality for the bias weight(0) to be multiplied by x[0] = 1. Since N is usually much more than d, we have more equations than variables and can not use Gaussian Elimination to solve this system of equations. The Normal Equation can be used to find the least error squares solutions (best fit) to systems with more equations than unknowns: $X^T Xw = X^T y$, solving for $w = (X^T X)^{-1} X^T y$.

The normal equation solution is computationally expensive for larger matrices. Other solutions include minimising the error by partial derivatives with respect to each weight being set to zero or maximising the likelihood function by using the gradient descent algorithm and its variants. The performance evaluation of the regression is estimated by mean squared error: $\hat{\sigma}^2 = \frac{\sum_{i=1}^{N}(y_i - \hat{y_i})}{N} = \frac{\sum_{i=1}^{N}(y_i - (w_i x_i))}{N}$.

The same algorithm can be used for classification when Y is a categorical value (the class labels). Then the performance evaluation becomes the accuracy rate or the confusion matrix. Most machine learning algorithms and neural networks start from the regression and further specify the equation to add more objectives or constraints. The tensorisation.ipynb Python notebook shows more details.

3.3.2 Clustering

Clustering is an unsupervised machine learning approach such that given an unlabelled data matrix X, it can be represented as $X \approx AB^T$, such that each row in A (the canonical basis vector) selects a row in B, which contains the clustering vectors. Estimating H and C from X enables multi-way clustering.

3.3.3 Data Distributions and Mixtures:

Data distribution guides the statistical analysis and the generative methods in machine learning. A Gaussian distributed feature has its expected value calculated using an equation completely different from poisson, bernoulli, binomial or any known distribution. Predefining the distribution of the dataset is used in estimating the parameters in parametric machine learning methods. When the distribution is not known, non-parametric or kernel methods are used. Bayesian methods are probabilistic methods that rely on understanding the data distribution. These methods are generative such as they know how samples are generated in each class, rather than the discriminative machine learning methods that can only identify the sample's class without creating a complete generative model. The assumption of Gaussian distributions because of the central limit theorem has failed in many situations. Some analysts justify the 2008 financial crises to be caused by this assumption in banking and financial modelling and prediction software (Watts, 2016).

Data often come from various distributions and mixtures of Gaussians. Studying the data distribution properly leads to appropriate analysis. Often there are latent variables that are not directly observed in the dataset that methods like factor analysis reveal. Chapter two will explain these methods and more as we proceed. There is a lot to learn about the various data distribution shapes, expected variable, standard deviation calculations, and their effects on building a model. Books such as (Sun and Kim, 2020), (Downey, 2013) cover topics required to understand parametric Bayes, and a deep dive into the non-parametric Bayes will make it even clearer, such as in (Ghosal and Vaart, 2017).

3.3.4 Graph Structures:

Graphs and networks: Graphs are made up of vertices and edges, where edges are defined as ordered pairs of vertices, like (i,j). A graph with n vertices and m edges can be represented as an $n \times n$ matrix M, where M[i,j] denotes the number (or weight) of edges from vertex i to vertex j. There are surprising connections between combinatorial properties and linear algebra, such as the relationship between paths in graphs and matrix multiplication and how vertex clusters relate to the eigenvalues/vectors of appropriate matrices.

Tensors elements can be sparse, and the coordinate connecting them is not steadily growing to contain data points in every unit increment across all coordinates. A Manifold of points in a dataset can be connected (mapped) to a lower dimension embedding forming a network or a graph structure that does not need to follow a Euclidean coordinate system.

The graph structure of the gas distribution example earlier is illustrated in **Error! Reference source not found.**. Various schematic higher dimension graph representations are shown in Figure 9, such as a graph is just the data structure representing the entity, such as a real value or class in a. A one-dimensional tensor (vector) is represented with one arrow connecting two points and can be extended to more points on the same single dimension. A two-dimension tensor (matrix) is represented with a graph with two arrows at every node, representing every change of coordinate value. Similarly, in the d to f illustrations in Figure 9, you will find that each node has arrows in each dimension to connect to the previous or following index value on that coordinate.

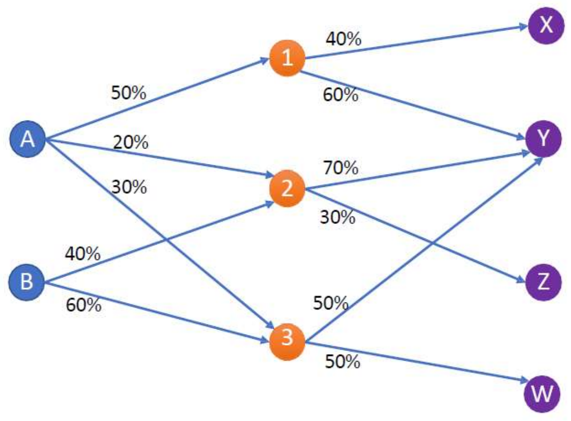


Figure 8: Gas Distribution Example Graph structure

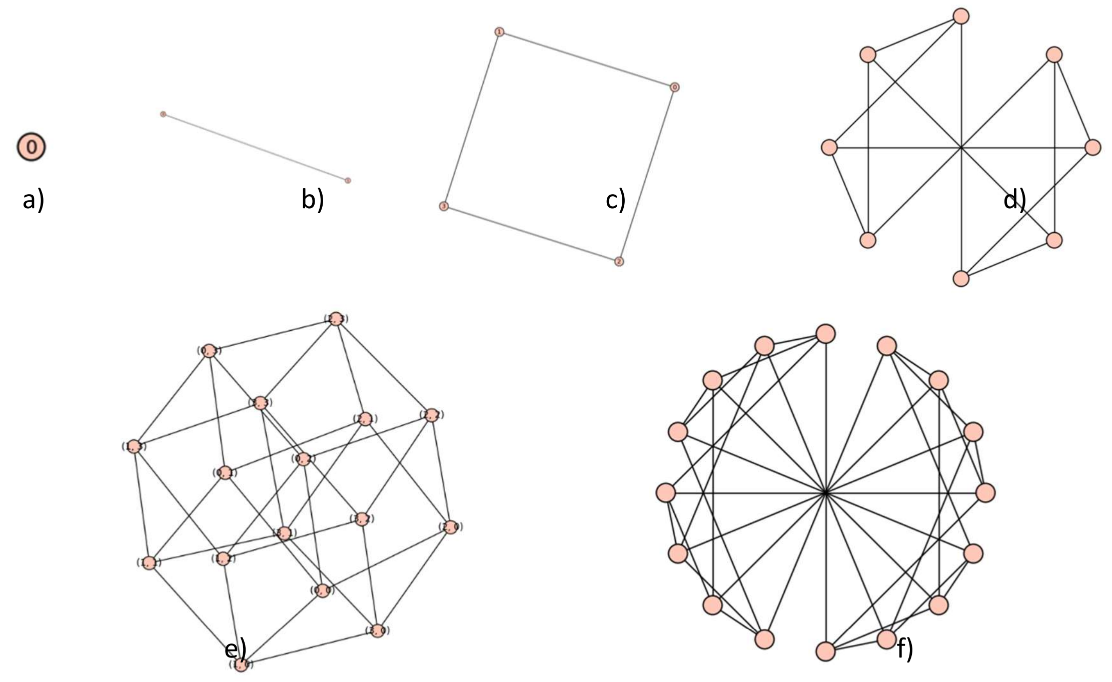


Figure 9: Tensors Graph Structures: a) scalar, b) 2D, c) 3D, d) 4D, e) 5D, and f) 6D

Python has several stable packages implementing graph/network data structures with various analysis tools and efficient parallel implementation. Python-igraph is the set of Python bindings for igraph, a collection of network analysis tools emphasising efficiency, portability and ease of use. NetworkX is another package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It is implemented based on NumPy and SciPy and therefore supports all common platforms. graph-tool is yet another efficient package for manipulation and statistical analysis of graphs, based on the C++ Boost Graph Library and parallelised using OpenMP.

3.4 Multilinear Algebra

The tensor operations presented in chapter one are the foundation for Multilinear algebra expressed in Euclidean geometry. This section will discuss non-linear functions, followed by a summary of differential geometry on manifold visualizations. These will help visualise the algorithmic concepts discussed in this book and the literature.

3.4.1 Non-Linearly Separable Datasets

Data is a matrix of entities in m rows and n features in columns. A point p in the dataset is a row vector of all features describing one entity in the n-dimensional space. The number of the features N is the dimensionality of the dataset $\mathbb{R}^n$, in which the data vectors from the vector space of the problem with basis e_i $0 \leq i < N$, defined as:

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \dots, e_{n-1} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} e_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

Any dataset can be described as the Manifold M in $\mathbb{R}^n$ that contains all points (rows) in the dataset. Chapter one explains that machine learning aims to approximate a function describing a given dataset. The equation can be linear, such that the linear regression/classification method explained in chapter one will provide a good approximation. When data is not linearly separable, a curve equation can be the one that describes its dynamics. Curves can be defined in the non-Euclidian space, such as hyperbolic geometry and elliptic geometry that this section will focus on, and it can also be defined in the Riemannian geometry that the next section and chapter five will dive more into it.

Curves equations are described as one of the four types of conic sections, which are the intersections of the surface of the cone with a plane at different angles. Figure 10 shows the sections as cuts through a cone described in Wikipedia. Each type is described by a focus point, directrix line, and eccentricity ratio. Eccentricity is the constant multiple that describes the distance between all the points on the curve of the conic section. The focus point is multiples of the eccentricity constant of the distance to the directrix line. Although it is not considered a conic section anymore, the first type is the circle. Circles are generated when the cutting plane is parallel to the plane of the generating circle of the cone. It is a particular type with a focus point, to which all the points on the circumference are of equal distance and no directrix line; hence, eccentricity is equal to zero. The second type is the ellipse, with eccentricity equal to ½, forming a closed curve inside the cone. The third type is the parabola with eccentricity equal to one. The fourth type is the hyperbola, with eccentricity equals two.

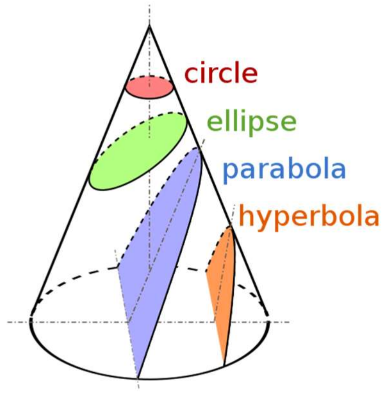


Figure 10: Curves as conic sections.

A quadratic equation of degree two can describe a non-linear decision plane, forming an arc or curve. A polynomial equation of degree two can be represented algebraically using a matrix as well. For example, given a dataset as follows:

x	y
-1	3
0	1
1	1
2	4
3	6

A system of equations can be defined as follows:

$w_0 - 1w_1 + 1w_2 = 3$
$w_0 + 0w_1 + 0w_2 = 1$
$w_0 + 1w_1 + 1w_2 = -1$
$w_0 + 2w_1 + 4w_2 = 1$
$w_0 + 3w_1 + 6w_2 = 3$

This dataset X fits a parabola. The equation of the parabola is: $y = w_0 + w_1x + w_2x^2$. We can use the normal equation again. The following matrix X contains an extra first column vector as constant 1, a second column vector as the given x variable/feature, and a third column vector as x^2. Given y, we can infer w (three parameters weights or coefficients to infer):

$$X = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, y = \begin{bmatrix} 3 \\ 1 \\ -1 \\ 1 \\ 3 \end{bmatrix}$$

$$X^TXc = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 4 & 9 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 4 & 9 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ -1 \\ 1 \\ 3 \end{bmatrix} = X^Ty$$

$$= \begin{bmatrix} 5 & 5 & 15 \\ 5 & 15 & 35 \\ 15 & 35 & 99 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 33 \end{bmatrix}$$

Now, we can solve by Gaussian Elimination as follows:

1. Form the Augmented Matrix: $\left[\begin{array}{ccc|c} 5 & 5 & 15 & 7 \\ 5 & 15 & 35 & 7 \\ 15 & 35 & 99 & 33 \end{array}\right]$

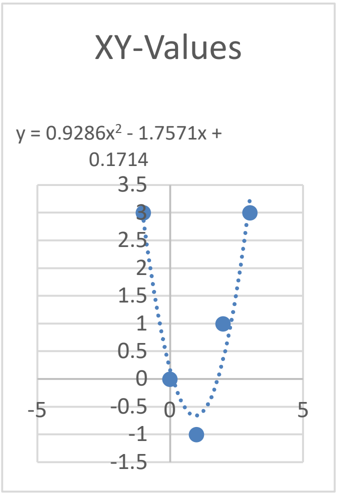
2. $r_1 \div 5$ $\left[\begin{array}{ccc|c} 1 & 1 & 3 & 1\frac{2}{5} \\ 5 & 15 & 35 & 7 \\ 15 & 35 & 99 & 33 \end{array}\right]$

3. $-5 * r_1 + r_2$ and $-15 * r_1 + r_3$. $\left[\begin{array}{ccc|c} 1 & 1 & 3 & 1\frac{2}{5} \\ 0 & 10 & 20 & 0 \\ 0 & 20 & 54 & 12 \end{array}\right]$

4. $r_2 \div 10$ $\left[\begin{array}{ccc|c} 1 & 1 & 3 & 1\frac{2}{5} \\ 0 & 1 & 2 & 0 \\ 0 & 20 & 54 & 12 \end{array}\right]$

5. $-20 * r_2 + r_3$. $\left[\begin{array}{ccc|c} 1 & 1 & 3 & 1\frac{2}{5} \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 14 & 12 \end{array}\right]$

6. $r_3 \div 14$ $\left[\begin{array}{ccc|c} 1 & 1 & 3 & 1\frac{2}{5} \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & \frac{6}{7} \end{array}\right]$



We can continue by back-substitution with $w_2 = \frac{6}{7}$

$w_1 + 2(\frac{6}{7}) = 0 \Rightarrow w_1 = -1\frac{5}{7}$

$w_0 + (-1\frac{5}{7}) + 3(\frac{6}{7}) = 1\frac{2}{5} \Rightarrow w_0 = \frac{19}{35}$

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{19}{35} \\ -1\frac{5}{7} \\ \frac{6}{7} \end{bmatrix} = \begin{bmatrix} 0.54 \\ -1.71 \\ 0.86 \end{bmatrix}$$

These are not precisely the coefficients calculated by the chart to identify the trend line on the parabola. However, they produce the least squared error as $\frac{32}{35}$ as shown below:

X-Values	Y-Values	Predicted Y	Error	Error^2
-1	3	$3\frac{4}{35}$	$\frac{-4}{35}$	$\frac{16}{1225}$
0	1	$\frac{19}{35}$	$\frac{16}{35}$	$\frac{256}{1225}$
1	-1	$\frac{-11}{35}$	$\frac{-24}{35}$	$\frac{576}{1225}$
2	1	$\frac{19}{35}$	$\frac{16}{35}$	$\frac{256}{1225}$
3	3	$3\frac{4}{35}$	$\frac{-4}{35}$	$\frac{16}{1225}$

The excel plot might use a more computationally efficient least squared error minimisation than the normal equation, in which computational complexity is O($n^{2.4}$) to O(n^3). The most efficient alternative is the gradient descent algorithm, with computational complexity O(kn^2), and k is the number of iterations (Carter, 1995).

We can expand the normal degree to a higher degree polynomial m for n data points as follows:

$$x = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^m \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}, \text{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix}, \text{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

Some datasets are difficult to estimate their curve functions. To know which conic section is represented by a dataset with the equation $6x^2 + 15y^2 + 19xy = 1$, a substitution T: u = 2x + 3y; v = 3x + 5y, clears the mixed term 19xy from the quadratic form $6x^2 + 15y^2 + 19xy$ and we get, uv = 1. This is achieved by diagonalisation. Then using the substitution S: w = u + v; z = u – v, we have the equation $w^2 + z^2 = 4$, which is hyperbola equation. These compose linear mapping from $S \circ T: \mathbb{R}^2 \to \mathbb{R}^2$, which is a multiplication of the matrices A and B capturing the linear transformation T and S, respectively. These are:

$A = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, then $S \circ T = BA = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix} =$
$\begin{bmatrix} 1 \times 2 + 1 \times 3 & 1 \times 3 + 1 \times 5 \\ 1 \times 2 - 1 \times 3 & 1 \times 3 - 1 \times 5 \end{bmatrix} = \begin{bmatrix} 5 & 8 \\ -1 & -2 \end{bmatrix}.$

More details about linear maps, diagonalisation, and various applications can be found in (Chahal, 2018). Chapter five focuses on mapping functions and achieving better representation on a new coordinate basis.

Global polynomial optimization is achieved in Tensor form using tensor decomposition approaches (Marmin, Castella and Pesquet, 2020). This is useful for various function approximation applications for multilinear functions, non-linear functions with polynomials, matrix-matrix multiplication, and systems of polynomial equations.

Python notebook Classification_Linear_NonLinear.ipynb shows examples of non-linearly separable datasets solved by different SVM kernel functions and other linear and non-linear algorithms, as discussed in chapter two.

When the non-linear solution is complex, zooming into any arc enough will approximate it as a line and can be solved linearly or in lower dimensions.

3.4.2 Differential Geometry on Manifolds

This section explains some mathematical preliminaries referenced in the following sections and chapters. The Riemannian space or Riemannian Manifolds describe curvatures in higher dimensions and provide geometric properties to facilitate the partial differential equations used in many Machine Learning (ML) algorithms. Some visualisations of these concepts are presented in https://youtube.com/playlist?list=PLbRB7u42hOE8rMIvShBxxiSdBdh9yQQL .
Since ML and Data science perspectives are the main focus of this book, we will cover the essentials of how calculus on Manifolds is applied in deriving some of the basic algorithms and explains their behaviour. Calculus on Manifolds is better explained by the derivation steps and visualisations, such as the style adopted in (Fortney, 2018). This book's main aim is to show how these concepts affect the behaviour of the functions in the higher space. Here, we will summarise the main findings relevant to ML rather than how they are derived.

Any equation dynamics are best captured in the form of the rate of change that is calculated using calculus methods. Plotting any dataset against the x-y coordinate partially visualises its dynamics. As the data increases in dimensionality, simple plotting will not be feasible. Alternatively, we learn a manifold, which is defined abstractly as a Euclidean space embedded locally in the higher dimensional space $\mathbb{R}^n$ for some value for n. Manifolds spaces are collections of data points, while vector spaces are collections of vectors with linear transformation functions defined on them. Physically, not all transformations are feasible for a dataset, so learning a manifold representing the given points only, reduces the dimensionality. Then, we can algebraically work with gradients of the data to understand its dynamics. Gradients are partial first derivatives that are slopes of lines tangent to the graph of the function at a certain point.

Manifolds' data points are assumed to have local Euclidean space on which an equation of neighbourhood can retrieve equally spaced points using Euclidean distance measures and on which parallelism and perpendicular relationships can be identified. Figure 11 (a) shows an example of Euclidean space on $\mathbb{R}^2$ on which a parallel transport from v_p = (2, 1) at point p to point q is done, and parallel vectors are created. Another example is that the distance from point a to point b is measurable using the Euclidean distance metric. Figure 11 (b) shows an $\mathbb{R}^2$ Manifolds represent the earth map on which point p is at the north pole and point q is on the equator. Neither distance nor parallelism can be calculated using Euclidean metrics. Root Mean Square Error (RMSE) is based on the Euclidean metric; hence all our optimisation algorithms, once fed with a new data structure for tensors, will also need to choose a suitable metric. A Minkowski metric is defined on 4-D spacetime as $\eta(v_1, v_2) \equiv x_1x_2 + y_1y_2 + z_1z_2 - t_1t_2$. Otherwise, a **Riemannian metric** can be used instead, such as the **Fisher metric** based on the Fisher information matrix that uses KL divergence as a distance measure. The KL divergence is a measure of dissimilarity between two distributions, and the Fisher information Matrix is the **Hessian Matrix** of the **KL divergence** at the point where two distributions are equal. These will be further explained in the last subsection of this section while discussing Tensors on Manifolds. Some preliminary definitions that are needed while working with Manifolds are as follows:

- **Coordinate maps** $\varphi_i : U_i \rightarrow \mathbb{R}^n$ are defined on an *n*-dimensional **manifold** space *M* that can be entirely covered by a collection of **local coordinate neighbourhoods** U_i with one-to-one mappings φ_i.
- **A coordinate patch** or a **chart** refers to both U_i and φ_i, (U_i, φ_i)
- **A coordinate system or an atlas of M {(Ui, ϕ i)}** is defined as the set of all the charts. The atlas of the world map is a collection of charts.
- Since the U_i cover all of *M* we write that $M = \bigcup U_i$. Also, since φ_i is one-to-one, it is invertible, so φ_i^{-1} exists and is well defined.
- If two charts have a non-empty intersection, $U_i \cap U_j \neq \emptyset$, then the functions $\varphi j \circ \varphi_i^{-1}: \mathbb{R}^n \rightarrow \mathbb{R}^n$ are called **transition functions**.
- A **differentiable manifold** is a set M, together with a collection of charts (U_i, φ_i), where $M = \bigcup U_i$, such that every mapping $\varphi j \circ \varphi_i^{-1}$, where $U_i \cap U_j \neq \emptyset$, is differentiable.

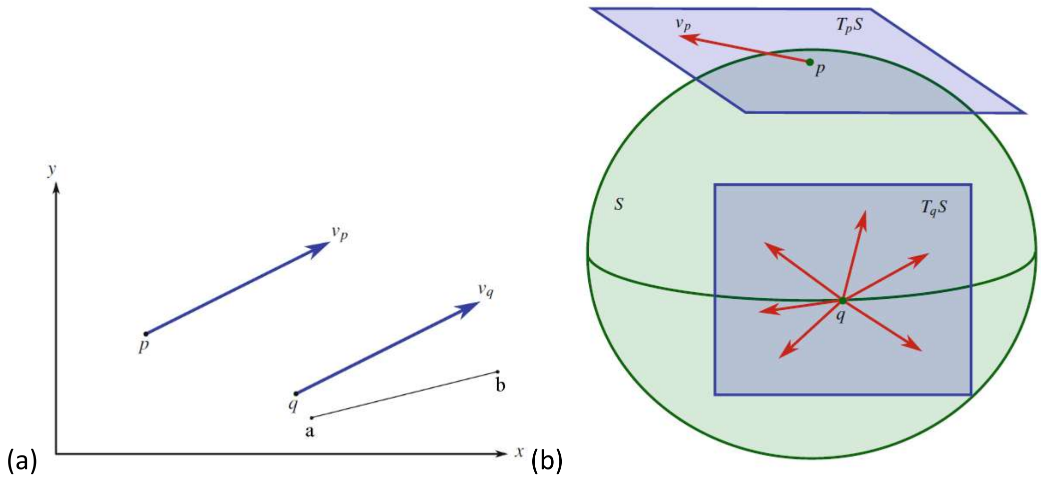


Figure 11: (a) The vector 2e$_1$ + e$_2$ drawn at two different points p and q in the manifold $\mathbb{R}^2$. These two vectors are parallel to each other using a parallel transport from v$_p$ to point q. A straight line between two points a and b in the manifold $\mathbb{R}^2$. It is not clear which of the vectors at q is parallel to the vector at p (Fortney, 2018).

This defines the **Manifold** as space covered by coordinate charts that are invertible and provide continuous mappings to some subset of $\mathbb{R}^n$. A manifold of a dataset connects the data points of the dataset using a coordinate chart basis calculated from their rate of change rather than a specific coordinate system. This will be clear by the end of this section.

Set of all vectors v$_p$ going through a data point p in Manifold M, forms a **tangent space** that is denoted by T$_p$M or $Tp\mathbb{R}^n$. Figure 11 (b) shows a green sphere S ⊂ $\mathbb{R}^3$ on which $\mathbb{R}^3$ vectors are embedded in the tangent spaces at a point p and point q, which are the blue $\mathbb{R}^2$ tangent plane (or hyperplane for higher dimensions) to the manifold at these points. The Whitney embedding theorem states that any reasonably nice manifold can be embedded into $\mathbb{R}^n$ for some sufficiently large n. The basis of the T$_p$M space is the **partial differential operator** $\boldsymbol{\frac{\delta}{\delta x_i}}$**,** or simplified to $\boldsymbol{\delta x_i}$**,** which is equivalent to basis vectors e$_i$ explained in chapter one, such as units of the coordinates of this space. In $\mathbb{R}^3$, $Tp\mathbb{R}^3 = span\left\{\left.\frac{\delta}{\delta x^1}\right|_p, \left.\frac{\delta}{\delta x^2}\right|_p, \left.\frac{\delta}{\delta x^3}\right|_p\right\}$. For vector $v \in Tp\mathbb{R}^3 = v^1\frac{\delta}{\delta x^1} + v^2\frac{\delta}{\delta x^2} + v^3\frac{\delta}{\delta x^3}$. The Einstein summation notation representation is $v^i\frac{\delta}{\delta x^i}$ for $\mathbb{R}^n$. As we can see, vectors have coefficients with upper indices. For every point in a manifold M, we have its own tangent space. The collection of these tangent spaces of all points in a manifold is called the tangent bundle. As defined in chapter one, a vector field is an operator on which vector spaces are defined. Vector fields also describe a section of the tangent bundle of a manifold mapping a vector to a point; this will be further explained below. For each coordinate neighbourhood of M we have a coordinate system (x$_1$, . . . ,x$_n$), which allows us to write a vectors field as v = v^i(x$_1$, . . .

,x$_n$) δ_{x^i}, where the v^i are real-valued functions on M. A vector field is illustrated in Figure 12(a).

(a) (b)

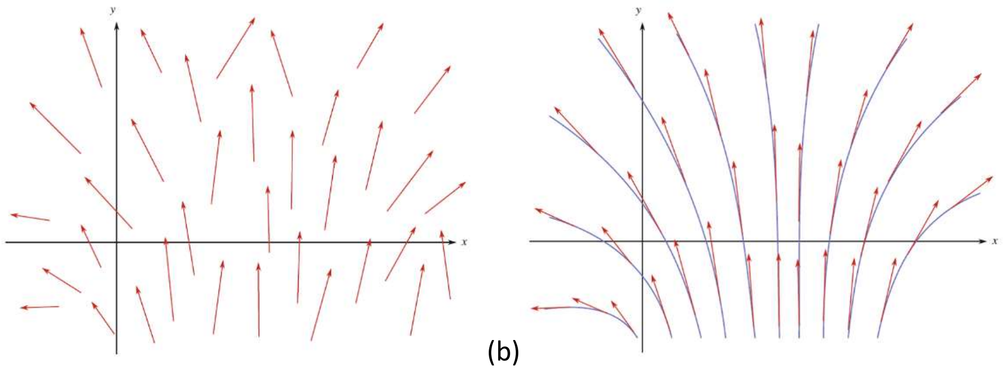


Figure 12: (a) A vector field drawn on the manifold $\mathbb{R}^2$. (b) A smooth vector field is drawn on the manifold $\mathbb{R}^2$. In both, each of the vectors shown are an element of a tangent space and not actually in the manifold $\mathbb{R}^2$(Fortney, 2018).

Tangent vectors are defined as smooth curves $\gamma: (-\epsilon, \epsilon) \subset \mathbb{R} \to M$ such that $\gamma(0) = p \in$ M and ϵ is just some small positive number, to avoid embedding M into some larger space $\mathbb{R}^n$ in order to have tangent Euclidean vectors v$_p$. The parameter that γ depends on, can be estimated as the time to avoid having sharp corners in a curve. If two curves have the same range close to p and have the same parametrization close to p they are called equivalent $\sim$, such as when you zoom on, they will be linear, not curves. The set of all equivalent curves is called an equivalence class: $[\gamma_1] \equiv \{\gamma | \gamma \sim \gamma_1\}$. Each equivalence class of curves at a point p, $[\gamma]_{p,}$ is defined to be a tangent vector at p. This defines the tangent space independent of any bigger space $\mathbb{R}^n$ the manifold is embedded in as $TpM = \{[\gamma]_p | \gamma: (-\epsilon, \epsilon) \subset \mathbb{R} \to M \; and \; \gamma(0) = p\}$. Going back to correlating these equivalent curves to the vector at point v$_p$, in the equivalence class definition $v_p = [\gamma]_p, \gamma(0) = p$, and we have$[(\gamma_1(t), \dots, \gamma_n(t))]$. Taking the derivative of γ with respect to time and evaluating the derivative at time t = 0, calculates v_p, $v_p = [\gamma']_p = [\gamma]'_p = [(\gamma_1'(t), \dots, \gamma_n'(t))]_p = [(\gamma_1(t), \dots, \gamma_n(t))]_p'$. This is calculated from the Jacobian matrix of the mapping $\gamma: (-\epsilon, \epsilon) \to M$ as $\begin{bmatrix} \gamma_1'(t=0) \\ \vdots \\ \gamma_n'(t=0) \end{bmatrix} = \begin{bmatrix} \left.\frac{\partial \gamma_1(t)}{\partial t}\right|_{t=0} \\ \vdots \\ \left.\frac{\partial \gamma_n(t)}{\partial t}\right|_{t=0} \end{bmatrix}$. Only the different classes of curves are important integral curves as will be explained below, forming different tangent vectors X, such as $X = \left.\frac{df(\gamma(t))}{dt}\right|_{t_0} = \sum_{i=1}^{n} \left.\frac{\delta f}{\delta x_i}\right|_p \left.\frac{dx^i(\gamma(t))}{dt}\right|_{t_0} = \sum_{i=1}^{n} X^i \left.\frac{\delta}{\delta x_i}\right|_p$. The following will continue using vectors but will use curves when needed.

In Figure 12(b), a smooth vector field is illustrated, which changes smoothly as you vary the point, such that it is possible to find curves on the manifold such that all the vectors in the vector field are tangent to the curves. The vector field v is called smooth if each of the functions $v^i(x_1, \ldots, x_n)$ is differentiable an infinite number of times with respect to the arguments $x_1, \ldots, x_n$. This means that there exists a function $\gamma\colon \subset \mathbb{R} \to M$ of a parameter t, such as time for continuity, defined as $\gamma(t) = (\gamma^1(t), \ldots, \gamma^n(t))$ that are differentiable at each point p in the Manifold M: $\left.\frac{d\gamma(t)}{dt}\right|_p = v_p$, which is the system of n differential equations $\frac{d}{dt}\gamma^i = v^i$. These integral curves are used below in the directional derivatives to measure the rate of a change of a function f over any curve γ passing through point p.

A **dual vector** is an object that eats a vector and spits out a number $\mathbb{R}^n \to \mathbb{R}$. The set of all dual vectors of a vector space V is the **dual space** V*. For example, given basis e_i, then vector v can be expressed as combinations of ei as $v = \sum_{i=1}^{n} v^i e_i$, its dual vector is $\tilde{v} = e^i(v) = v^i$, which is the coordinate function, or the projection of v on a specific coordinate i. This makes e^i the basis for the dual space V* denoted below as $\mathbf{d_{xi}}$. A dual vector $\tilde{v}$ is a one-to-one and onto map to v as $\tilde{v}(w) = \langle v. w\rangle$. This dot product is the metric dual of v.

The **dual space,** can also be denoted by $(\mathbb{R}^n)^*$, is the set of all functionals T on $\mathbb{R}^n \to \mathbb{R}$ satisfying two properties: i) T(v+w) = T(v) + T(w), and ii) T(c.v) = c. T(v). The dual space of the T_pM is the set of all differential one forms, which will be defined below, in the tangent space through point p in Manifold M and is denoted by $(TpM)^*$. $(TpM)^*$ is the **cotangent space** and has basis coordinates from the **differentials** $\mathbf{d_{xi}}$, which picks the ith component of the vector $v_p \in TpM$. v_p is the projection of the v_p vector on the ith coordinate. Therefore $d_{xi}(v_p)$ is equivalent to v_i. This makes the differential one forms in the $(TpM)^*$space to be acting as coordinate functions. For example, given a point p = (2, 3), the cartesian coordinate functions $\mathbb{R}^2 \to \mathbb{R}$ are x(p) = 2, and y(p) = 3. A point exists in any other coordinate system, such as polar, spherical, or cylindrical. The same p point is mapped to the polar coordinate system using mapping functions $r = \sqrt{x^2 + y^2}$, and $\theta = arctan\left(\frac{x}{y}\right)$, and inverse mapping functions are: x= r cos(θ) and y = r sin(θ). Working in the cotangent space using coordinate functions enables a coordinate-free approach, in which the most suitable coordinate system is used for any dataset. A transformation of components is achieved for point p in two charts x and y. Then their coordinate vectors transform with the Jacobian of the coordinate transformation x↦y $\left(\Lambda^i_{j'}\right)$: $\frac{\delta}{\delta y^i} = \sum_{j=1}^{n} \frac{\delta x^i}{\delta y^i}\frac{\delta}{\delta x^j}$, and inverse mapping $\Lambda^{j'}_i = \left(\Lambda^i_{j'}\right)^{-1}$. This is why in the tangent vector space, the $\frac{\delta}{\delta x^j}$ is the coordinate basis. For a vector v in a vector space with basis e_i is represented as v^ie_i , and covector ω

after a change of basis to e_i^* is defined as $\omega(v_1, \dots, v_n) = sgn(\pi) \cdot \omega\left(v_{\pi(1)}, \dots, v_{\pi(n)}\right)$ for any permutation of the n elements. When ω is acting on vector v, it is defined as $\omega = \omega e_i^* = \omega(v) = \sum_{i=1}^{n} \omega_i v^i$. The transformation to the cotangent space is a change of basis that is captured from the Jacobian matrix. For example in $\mathbb{R}^2$,$\omega_x^i\left(\frac{\delta}{\delta x^j}\right)= \delta_j^i$, and $\omega_y^i\left(\frac{\delta}{\delta y^j}\right) = \delta_j^i$, which is the inverse of the tangent vector space, as $\omega_y^i = \sum_{j=1}^{n} \frac{\delta y^i}{\delta x^j}\ \ \omega_x^j$, providing the cotangent vector space coordinate basis as $dy^j = \sum_{j=1}^{n} \frac{\delta y^i}{\delta x^j}\ \ dx^j$, which is generally expressed as dx^i. For example, the cartesian to polar change of basis can be achieved using the Jacobian matrix determinant as:

$$\begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{vmatrix} = \begin{vmatrix} \cos\theta & -\mathrm{r}\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} = r\ \cos^2\theta + \ r\sin^2\theta = r;\text{ therefore, } dx\, dy = r\, dr\, d\theta.$$

Figure 13 illustrates the relationships between the manifold $\mathbb{R}^3$, the tangent space $Tp\mathbb{R}^3$, and the cotangent space $T_p^*\mathbb{R}^3$. Although the cotangent space $T_p^*\mathbb{R}^3$ is attached to the manifold at the same point p that the tangent space $Tp\mathbb{R}^3$ is attached, for illustration, it is shown above the tangent space. In $\mathbb{R}^3$, $T_p^*\mathbb{R}^3 = Span\left\{dx^1|_p, dx^2|_p, dx^3|_p\right\}$**.** For differential form $\alpha \in T_p^*\mathbb{R}^3 = \alpha_1 dx^1 + \alpha_2 dx^2 + \alpha_3 dx^3$. The Einstein summation notation representation is $\alpha_i dx^i$ for $\mathbb{R}^n$ with α_i as real numbers. As we can see, covectors, that is, differential forms, also called coordinate functions, have coefficients with lower indices. Other notations and conventions need to be understood from every reference.

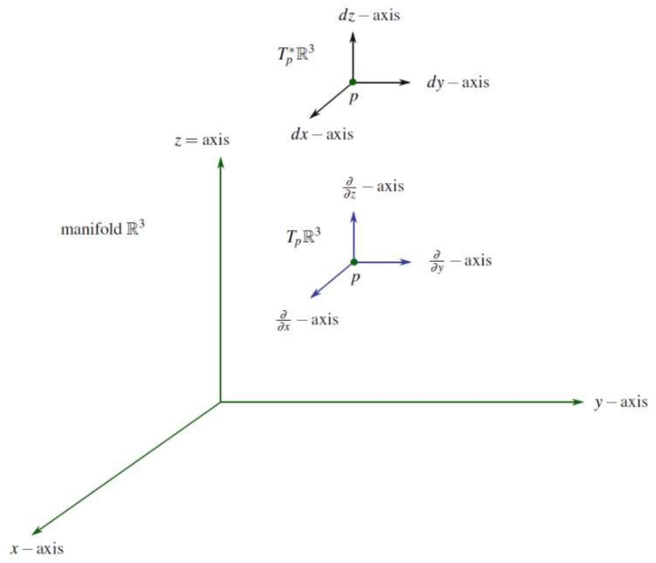


Figure 13: An illustration of the manifold $\mathbb{R}^3$ along with the tangent space $Tp\mathbb{R}^3$ "attached" to the manifold at point p. The dual space $Tp^*\mathbb{R}^3$ is drawn above the tangent space which it is dual to. Notice the different ways the axis are labelled (Fortney, 2018).

Differential forms start from zero-forms as normal functions/functionals that take scalar (one or more for multivariate functions), produce a real number, and go up to k-forms. A differential one-form $\alpha : T_p\mathbb{R}^n \to \mathbb{R}$, is defined as linear functional on the set of tangent vectors in $\mathbb{R}^n$ at each point p, it takes one vector as input and produces a real number that is its length (1-d volume) as projected on $\mathbb{R}$. Differential two-forms take two vectors and produce a real number representing their area; three-forms take three vectors and produce their volume, up to differential k-forms that take k-vectors and produce a real number representing the higher dimensional volume. Different subspace projections will be illustrated as we proceed. K here represents the number of coordinates (subspaces) used in the projection, which could be the whole dimensionality of the space n or lower down to 1. Figure 14 illustrates two ways to visualise differential one-form $\alpha = \alpha_1 d_x + \alpha_2 d_y + \alpha_3 d_z$ in the tangent space (left) and in the dual space / cotangent space $Tp^*\mathbb{R}^3$ (right). Notice each coordinate is isolated with its coefficient. The differential df_p is the linear approximation of the function f at the point p. In other words, the differential df_p "encodes" the tangent plane of f at p. Figure 15 shows the formula for the tangent plane T to the function f (x, y) at the point (x_0, y_0) as $T(x,y) = f(x_0,\ y_0) + \left.\frac{\delta f}{\delta x}\right|_{(x_0,\ y_0)} - (x - x_0) + \left.\frac{\delta f}{\delta y}\right|_{(x_0,\ y_0)} - (y - y_0)$. The figure shows vector v with end points in $\mathbb{R}^2$ is $(x_0 + v_1,\ y_0 + v_2)$. The tangent plane is the closest linear approximation of f at p. df_p can be thought of as the linear approximation of the function f at the point p. This is written as $df = \frac{\delta f}{\delta x}dx + \frac{\delta f}{\delta y}dy = \left[\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}\right]$ in $\mathbb{R}^2$ and extended to in $\mathbb{R}^n$as $df = \frac{\delta f}{\delta x_1}dx_1 + \frac{\delta f}{\delta x_2}dx_2 + \cdots \frac{\delta f}{\delta x_n}dx_n = \left[\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \ldots, \frac{\delta f}{\delta x_n}\right]$ which is the gradient vector of f, *grad(f)* or ∇*(f)*.

Differential one forms α are defined as linear functional on the set of tangent vectors in $\mathbb{R}^n$ at each point p such that $\alpha : T_p\mathbb{R}^n \to \mathbb{R}$. It takes as input one vector and produces its length (1-d volume) as projected on $\mathbb{R}$. Figure 14 illustrates two ways to visualise differential one-form $\alpha = \alpha_1 d_x + \alpha_2 d_y + \alpha_3 d_z$ in the tangent space (left) and in the dual space / cotangent space $Tp^*\mathbb{R}^3$ (right). The left graph shows the result of the one-form acting on the vector $v_p \in Tp\mathbb{R}^3$as the projection in the $\left.\frac{\delta}{\delta x_1}\right|_p \left.\frac{\delta}{\delta x_2}\right|_p \left.\frac{\delta}{\delta x_3}\right|_p$ −plane. The differential df_p is the linear approximation of the function f at the point p. In other words, the differential df_p "encodes" the tangent plane of f at p. Figure 15 shows the formula for the tangent plane T to the function f (x, y) at the point (x_0, y_0) as $T(x,y) = f(x_0,\ y_0) + \left.\frac{\delta f}{\delta x}\right|_{(x_0,\ y_0)} - (x - x_0) + \left.\frac{\delta f}{\delta y}\right|_{(x_0,\ y_0)} - (y - y_0)$. The figure shows vector v with end points in $\mathbb{R}^2$ is $(x_0 + v_1,\ y_0 + v_2)$.
The tangent plane is the closest linear approximation of f at p, so df_p can be thought of as

the linear approximation of the function f at the point p. This is written as $df = \frac{\delta f}{\delta x}dx + \frac{\delta f}{\delta y}dy = [\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}]$ in $\mathbb{R}^2$ and extended to in $\mathbb{R}^n$as $df = \frac{\delta f}{\delta x_1}dx_1 + \frac{\delta f}{\delta x_2}dx_2 + \cdots \frac{\delta f}{\delta x_n}dx_n = [\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \dots, \frac{\delta f}{\delta x_n}]$ which is the gradient vector of f, *grad(f)* or ∇*(f)*.

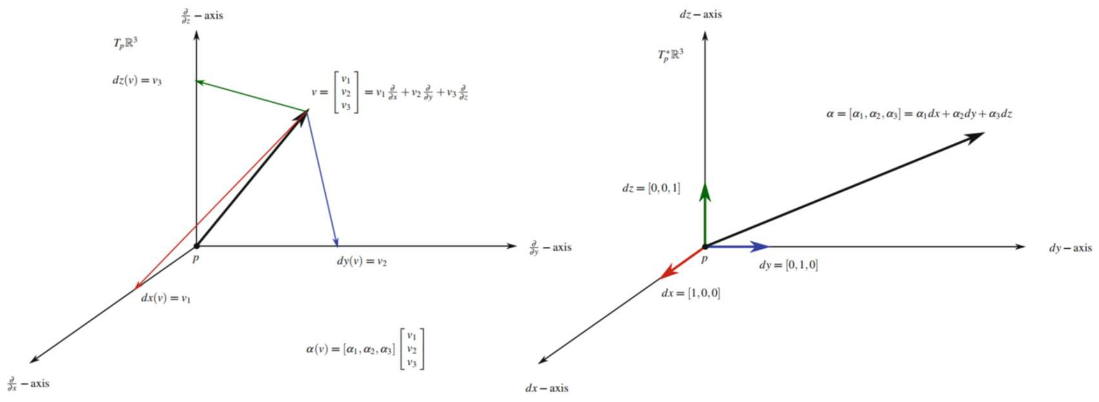

$\frac{\partial}{\partial z}$ − axis
$T_p\mathbb{R}^3$
$dz(v) = v_3$
$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = v_1\frac{\partial}{\partial x} + v_2\frac{\partial}{\partial y} + v_3\frac{\partial}{\partial z}$
$\frac{\partial}{\partial y}$ − axis
p
$dy(v) = v_2$
$dx(v) = v_1$
$\alpha(v) = [\alpha_1, \alpha_2, \alpha_3]\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$
$\frac{\partial}{\partial x}$ − axis
dz − axis
$T_p^*\mathbb{R}^3$
$\alpha = [\alpha_1, \alpha_2, \alpha_3] = \alpha_1 dx + \alpha_2 dy + \alpha_3 dz$
$dz = [0,0,1]$
dy − axis
p
$dy = [0,1,0]$
$dx = [1,0,0]$
dx − axis

Figure 14: Two different ways to visualize a differential one-form. As a linear combination of the projections onto the axes of the tangent space $Tp\mathbb{R}^3$ (left) or as dual-vectors/ co-vectors / row vectors in the cotangent space $Tp^*\mathbb{R}^3$ (right) (Fortney, 2018).

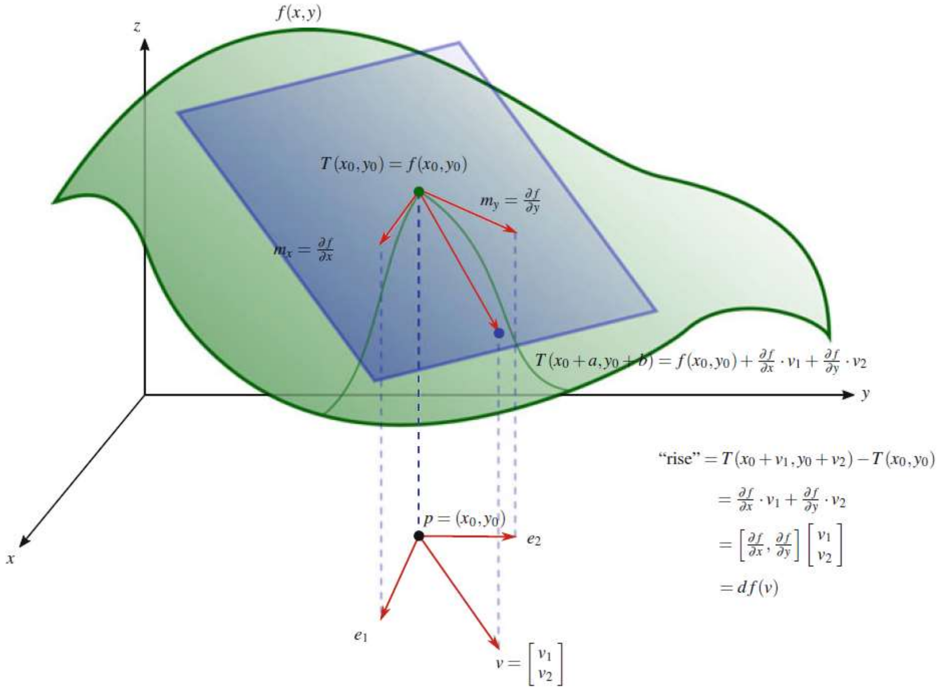

$f(x,y)$
z
$T(x_0,y_0) = f(x_0,y_0)$
$m_y = \frac{\partial f}{\partial y}$
$m_x = \frac{\partial f}{\partial x}$
$T(x_0+a, y_0+b) = f(x_0,y_0) + \frac{\partial f}{\partial x}\cdot v_1 + \frac{\partial f}{\partial y}\cdot v_2$
y
"rise" $= T(x_0+v_1, y_0+v_2) - T(x_0,y_0)$
$= \frac{\partial f}{\partial x}\cdot v_1 + \frac{\partial f}{\partial y}\cdot v_2$
$= \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$
$= df(v)$
$p = (x_0,y_0)$
e_2
x
e_1
$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

Figure 15: The differential df_p takes a vector v_p at some point p in Manifold $\mathbb{R}^2$ and produces a number which is the rise of the tangent plane to the graph of the function as it moves from p along the vector v_p. The illustration shows the projection on the tangent space and its equivalence with the basis coordinates at the bottom (Fortney, 2018).

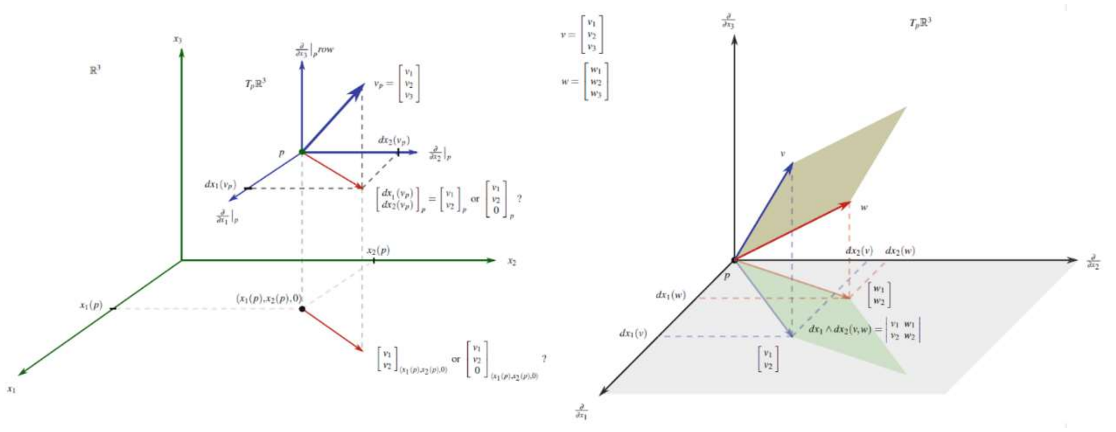

x_3
$\mathbb{R}^3$
$\frac{\partial}{\partial x_3}\Big|_p row$
$T_p\mathbb{R}^3$
$v_p = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$
$dx_2(v_p)$
$\frac{\partial}{\partial x_2}\Big|_p$
p
$dx_1(v_p)$
$\frac{\partial}{\partial x_1}\Big|_p$
$\begin{bmatrix} dx_1(v_p) \\ dx_2(v_p) \end{bmatrix}_p = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}_p$ or $\begin{bmatrix} v_1 \\ v_2 \\ 0 \end{bmatrix}_p$?
$x_2(p)$
x_2
$x_1(p)$
$(x_1(p), x_2(p), 0)$
$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}_{(x_1(p),x_2(p),0)}$ or $\begin{bmatrix} v_1 \\ v_2 \\ 0 \end{bmatrix}_{(x_1(p),x_2(p),0)}$?
x_1
$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$
$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$
$\frac{\partial}{\partial x_3}$
$T_p\mathbb{R}^3$
v
w
$dx_2(v)$ $dx_2(w)$
$\frac{\partial}{\partial x_2}$
p
$dx_1(w)$
$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$
$dx_1(v)$
$dx_1 \wedge dx_2(v,w) = \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix}$
$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$
$\frac{\partial}{\partial x_1}$

Figure 16: Projecting on lower dimensions (left). The parallelepiped spanned by v and w (brown) is projected onto the $\frac{\delta}{\delta x_1}\Big|_p \frac{\delta}{\delta x_2}\Big|_p$ *-plane in* $Tp\mathbb{R}^3$ *(right). We want* $dx_1 \wedge dx_2$ *to find the volume of this projected area (Fortney, 2018).*

Figure 16 (left) shows the spaces that various vectors belong to by taking point p in $\mathbb{R}^3$, and vector v_p from point p = $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$, then showing different projections such as = $\begin{bmatrix} dx_1(v)_p \\ dx_2(v)_p \\ 0 \end{bmatrix}_p$ in the tangent space $Tp\mathbb{R}^3$ or at point p in Manifold $\mathbb{R}^3$, $\begin{bmatrix} dx_1(v)_p \\ dx_2(v)_p \\ 0 \end{bmatrix}_{(x_1(p),x_2(p),0)}$ at point $(x_1(p), x_2(p), 0)$ in Manifold $\mathbb{R}^3$, $\begin{bmatrix} dx_1(v)_p \\ dx_2(v)_p \end{bmatrix}$ in the 2-d plane $\frac{\delta}{\delta x_1}\Big|_p \frac{\delta}{\delta x_2}\Big|_p$ of $Tp\mathbb{R}^3$, or $\begin{bmatrix} dx_1(v)_p \\ dx_2(v)_p \end{bmatrix}_{(x_1(p),x_2(p),0)}$ at the point $(x_1(p), x_2(p), 0)$in the xy-plane of $\mathbb{R}^3$.

Figure 16 (right) illustrates the wedge product of two one-forms to produce the area of the plane formed by two vectors on $\mathbb{R}^3$, the space of two-forms on $\mathbb{R}^3$ is denoted as $\Lambda^2(\mathbb{R}^3)$ and illustrated in Figure 16 (right). It shows the result of the one-form acting on the vector $v_p \in Tp\mathbb{R}^3$as the projection of $v_p \in \mathbb{R}^3$ in the $\frac{\delta}{\delta x_1}\Big|_p \frac{\delta}{\delta x_2}\Big|_p$ −plane of $\in Tp\mathbb{R}^2$. In Figure 16 (right), given two vectors v and w, the area (2-d volume) is calculated using the wedge product ∧ defined by the determinant of the matrix formed by the appropriate elements of the vectors. For example:

$v = \begin{bmatrix}1\\2\\3\end{bmatrix}, w = \begin{bmatrix}4\\5\\6\end{bmatrix}$ then, $dx_1 \wedge dx_2\left(\begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}4\\5\\6\end{bmatrix}\right) \equiv \begin{vmatrix}dx_1(v)_p & dx_1(w)_p\\ dx_2(v)_p & dx_2(w)_p\end{vmatrix} = \begin{vmatrix}1 & 4\\ 2 & 5\end{vmatrix} =$ $(1)(5)- (4)(2) = -3$, if we calculate $dx_2 \wedge dx_1\left(\begin{bmatrix}1\\2\\3\end{bmatrix} \quad \begin{bmatrix}4\\5\\6\end{bmatrix}\right) \equiv \begin{vmatrix}dx_1(v)_p & dx_1(w)_p\\ dx_2(v)_p & dx_2(w)_p\end{vmatrix} =$ $\begin{vmatrix}2 & 5\\ 1 & 4\end{vmatrix} = (2)(4)- (5)(1) = 3$. Therefore, the wedge product of two one-forms is defined in terms of the determinant of the appropriate vector projections. Similarly, the volume of other plane projections in higher dimensions can be derived from the matrix determinant. To generalise in the $\mathbb{R}^n$, given v$_1$, v$_2$, . . . ,v$_n$ vectors, the wedge product of n one-forms is defined as:

$$dx_1 \wedge dx_2 \wedge \ldots . \wedge dx_n(v_1, v_2, \ldots v_n) \equiv \begin{vmatrix} dx_1(v)_1 & dx_1(v)_2 & \ldots & dx_1(v)_n \\ dx_2(v)_1 & dx_2(v)_2 & \ldots & dx_2(v)_n \\ \vdots & \vdots & \ddots & \vdots \\ dx_n(v)_1 & dx_n(v)_2 & \ldots & dx_n(v)_n \end{vmatrix}$$

Another formula for the determinant is given in (Fortney, 2018) as:

$$\begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix} = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} a_{\sigma(i)i}$$

Then

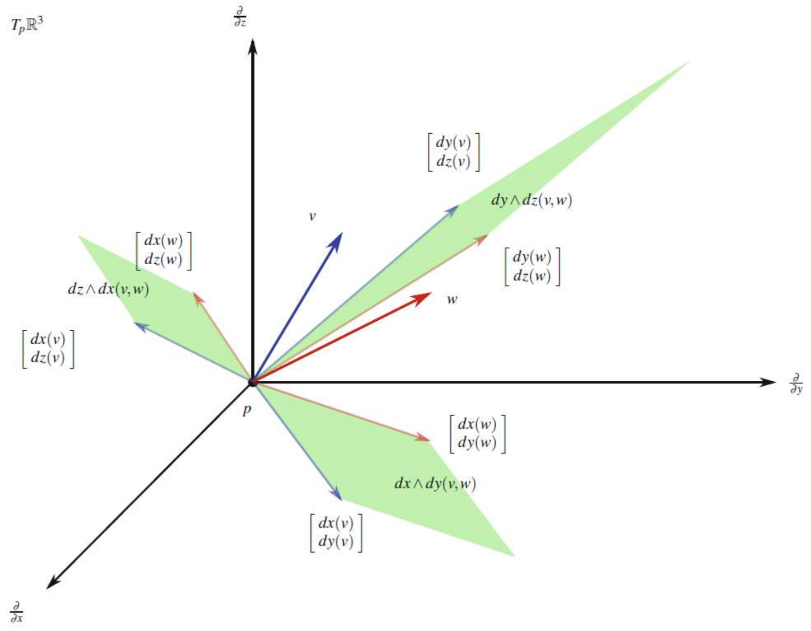
$$dx_1 \wedge dx_2 \wedge \ldots . \wedge dx_n(v_1, v_2, \ldots v_n) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{j=1}^{n} dx_{\sigma(i_j)}(v_j)$$

Where S$_n$ is The set of permutations of {1, …,n} with n! elements in it and particular permutation is denoted as σ such that $a_{\sigma(i)i}$ is the permuted row for column i.
There are many algebraic properties such as:

- if we have any two of the one-forms the same, that is, i$_j$ = i$_k$ for some j ≠ k then we have two rows that are the same, which gives a value of zero dx$_i$ ∧ dx$_i$ = 0.
- if i ≠ j and we switch dx$_i$ and dx$_j$ that the wedge product changes sign, dx$_i$ ∧ dx$_j$ = −dxj ∧ dxi. This defines the anti-symmetric property of the wedge product.

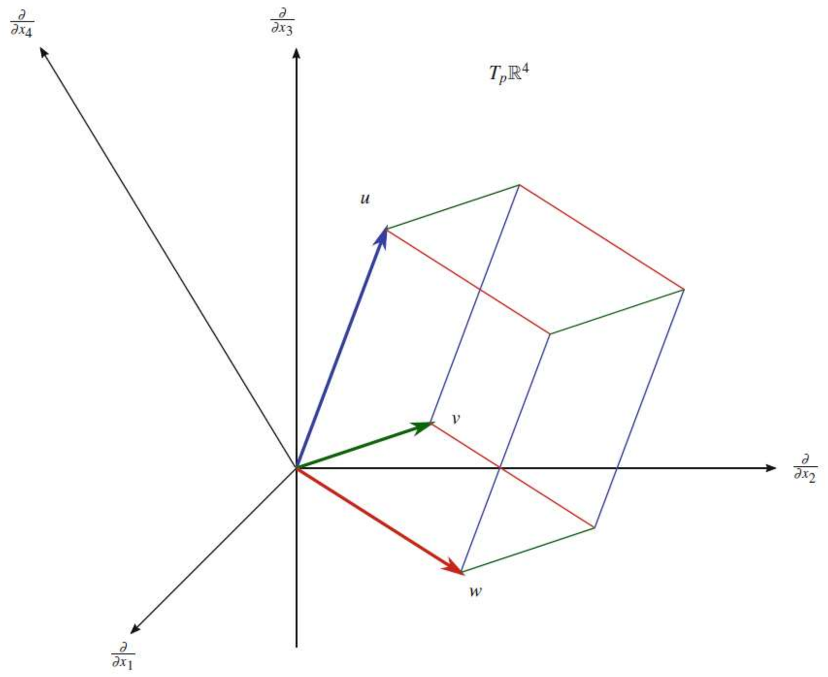
We can also find the wedge products of n-forms on subspaces projections of $\mathbb{R}^n$. Figure 17 illustrates an example of two vectors v and w in $\mathbb{R}^3$ projected on three 2-d planes in $Tp\mathbb{R}^3$and then summed up: $(dx \wedge dy + dy \wedge dz + dz \wedge dx)(v, w) = dx \wedge dy(v, w) + dy \wedge dz(v, w) + dz \wedge dx(v, w)$. To generalise for $\mathbb{R}^n$, the two forms on n-dimensional manifolds are defined by finding projections of the two vectors v, w on the appropriate two-dimensional subspaces of $Tp\mathbb{R}^n$, forming the two-dimensional parallelepipeds, their volumes are then scaled by the appropriate factor and then summed. The only distinction is that $Tp\mathbb{R}^n$ has $\frac{n(n-1)}{2}$ distinct two-dimensional subspaces. This is denoted as $\Lambda_p^2\mathbb{R}^n$, and generalised for $\Lambda_p^k\mathbb{R}^n$ for 0 <=k <= n. For example, zero-forms in $\mathbb{R}^3$: $\Lambda^0\mathbb{R}^3 = span\,\{1\}$, which is one-dimensional functions on $\mathbb{R}^3$; another example is $\Lambda^3\mathbb{R}^3 =$

span $\{dx \wedge dy \wedge dz\}$, which is one-dimensional three-forms in $\mathbb{R}^3$, with basis given by {dx ∧ dy ∧ dz}. Since the three-form dx ∧ dy ∧ dz projects three vectors u, v,w onto the $\delta x \delta y \delta z$ -subspace of $Tp\mathbb{R}^3$, which is the whole $Tp\mathbb{R}^3$. $\Lambda^2\mathbb{R}^3$is illustrated in Figure 17, as two-forms on $\mathbb{R}^3$, with basis given by { dx ∧ dy, dy ∧ dz , dz ∧ dx} projects the two vectors v,w onto the 2-d planes: $\delta x \delta y$ -plane, $\delta y \delta z$ -plane and $\delta z \delta x$ -plane as subspaces of $Tp\mathbb{R}^3$. Another example is given as $\Lambda_p^3\mathbb{R}^4$ as three-forms in $\mathbb{R}^4$ with basis given by $\{ dx_1 \wedge dx_2 \wedge dx_3, dx_1 \wedge dx_2 \wedge dx_4, dx_1 \wedge dx_3 \wedge dx_4, dx_2 \wedge dx_3 \wedge dx_4\}$ projects the three vectors u, v,w onto the 3-d planes of $\delta x_1 \delta x_2 \delta x_3$, illustrated in Figure 18, and also $\delta x_1 \delta x_2 \delta x_4$, $\delta x_1 \delta x_3 \delta x_4$, $\delta x_2 \delta x_3 \delta x_4$ as subspaces of $Tp\mathbb{R}^4$.



$T_p\mathbb{R}^3$
$\frac{\partial}{\partial z}$
$\frac{\partial}{\partial y}$
$\frac{\partial}{\partial x}$
p
v
w
$\begin{bmatrix} dy(v) \\ dz(v) \end{bmatrix}$
$dy \wedge dz(v,w)$
$\begin{bmatrix} dy(w) \\ dz(w) \end{bmatrix}$
$\begin{bmatrix} dx(w) \\ dz(w) \end{bmatrix}$
$dz \wedge dx(v,w)$
$\begin{bmatrix} dx(v) \\ dz(v) \end{bmatrix}$
$\begin{bmatrix} dx(w) \\ dy(w) \end{bmatrix}$
$dx \wedge dy(v,w)$
$\begin{bmatrix} dx(v) \\ dy(v) \end{bmatrix}$

Figure 17: The action of the two-forms dx ∧ dy + dy ∧ dz + dz ∧ dx on two vectors v and w, projected on the three plans: $\left.\frac{\delta}{\delta x}\right|_p \left.\frac{\delta}{\delta y}\right|_p$ *-plane,* $\left.\frac{\delta}{\delta y}\right|_p \left.\frac{\delta}{\delta z}\right|_p$ *-plane,* $\left.\frac{\delta}{\delta z}\right|_p \left.\frac{\delta}{\delta x}\right|_p$ *-plane in* $Tp\mathbb{R}^3$*, and then summed up (Fortney, 2018).*



$\frac{\partial}{\partial x_4}$
$\frac{\partial}{\partial x_3}$
$T_p\mathbb{R}^4$
u
v
$\frac{\partial}{\partial x_2}$
w
$\frac{\partial}{\partial x_1}$

Figure 18: Illustration of three vectors u, v,w onto the 3-d planes of $\delta x_1\delta x_2\delta x_3$*, as a subspace of* $Tp\mathbb{R}^4$ *(Fortney, 2018).*

For arbitrary k-forms in $\Lambda^k\mathbb{R}^n$. The notation can be simplified such that we do not write all the elements of the basis of $\Lambda^k\mathbb{R}^n$. For example, for arbitrary two-forms in $\Lambda^2\mathbb{R}^3$, we can define $\alpha = a_{12}dx_1 \wedge dx_2 + a_{23}dx_2 \wedge dx_3 + a_{31}dx_3 \wedge dx_1$. We can then generalise to $\Lambda^k\mathbb{R}^n$ as $\alpha = \sum_I a_I dx^I$, such that I stands for elements in the set of k increasing indices i_1, i_2, …, i_k, where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, defined as $I \in J_{(k,n)} = \{(i_1, i_2, \ldots, i_k) | 1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq n\}$. Other examples:

- For arbitrary three-forms in $\Lambda^3\mathbb{R}^4$, $I \in J_{3,4} = \{123, 124, 134, 234\}$, we can define $\alpha = \sum_I a_I dx^I = a_{123}dx^{123} + a_{124}dx^{124} + a_{134}dx^{134} + a_{234}dx^{234} = a_{123}dx_1 \wedge dx_2 \wedge dx_3 + a_{124}dx_1 \wedge dx_2 \wedge dx_4 + a_{134}dx_1 \wedge dx_3 \wedge dx_4 + a_{234}dx_2 \wedge dx_3 \wedge dx_4$, such that $dx^{123} = dx_1 \wedge dx_2 \wedge dx_3$ and so forth.
- For arbitrary two-forms in $\Lambda^2\mathbb{R}^4$, $I \in J_{2,4} = \{12, 13, 14, 23, 24, 34\}$, we can define $\alpha = \sum_I a_I dx^I = a_{12}dx^{12} + a_{13}dx^{13} + a_{14}dx^{14} + a_{23}dx^{23} + a_{24}dx^{24} + a_{34}dx^{34} = a_{12}dx_1 \wedge dx_2 + a_{13}dx_1 \wedge dx_3 + a_{14}dx_1 \wedge dx_4 + a_{23}dx_2 \wedge dx_3 + a_{24}dx_2 \wedge dx_4 + a_{34}dx_3 \wedge dx_4$, such that $dx^{12} = dx_1 \wedge dx_2$ and so forth

Given $\alpha \in \Lambda^k\mathbb{R}^n$, and $\beta \in \Lambda^l\mathbb{R}^n$, where $\alpha = \sum_I a_I dx^I$ and $\beta = \sum_J b_J dx^J$ defined as above, then $\alpha \wedge \beta = \sum_I (a_I dx^I) \wedge \sum_J (b_J dx^J) = \sum a_I b_J dx^I \wedge dx^J = \sum_k \left(\sum_{I \cup J, I, J \text{ are disjoint}} \pm a_I b_J\right) dx^K$. If I and J are disjoint then we have $dx^I \wedge dx^J = \pm dx^K$

where K = I ∪ J, but is reordered to be in increasing order and elements with repeated indices are dropped.

For example, given $\alpha \in \Lambda^2\mathbb{R}^8 = \sum_I a_I dx^I = 5dx_1 \wedge dx_2 - 6dx_2 \wedge dx_4 + 7dx_1 \wedge dx_7 + 2dx_2 \wedge dx_8$, where I has the elements of the set {12, 24, 17, 28} as a ⊂ $J_{2,8}$ and the coefficients are a_{12} = 5, a_{24} = −6, a_{17} = 7, and a_{28} = 2. Given also, $\beta \in \Lambda^3\mathbb{R}^8 = \sum_J b_J dx^J = 3dx_3 \wedge dx_5 \wedge dx_8 - 4dx_5 \wedge dx_6 \wedge dx_8$, where I has the elements of the set {358, 568} as a ⊂ $J_{3,8}$ and the coefficients are b_{358} = 3, and b_{568} = -4 , then $I \cup J =$ $\{12358, 24358, 17358, 28358, 12568, 24568, 17568, 28568\}$. Both 28358 and 28568 repeat the 8, since $dx_2 \wedge dx_8 \wedge dx_3 \wedge dx_5 \wedge dx_8 = 0$ and $x_2 \wedge dx_8 \wedge dx_5 \wedge dx_6 \wedge dx_8 = 0$, then we can drop them. K, after having the indices in increasing order, will be K = $\{12358, 23458, 13578, 12568, 24568, 15678\}$. Therefore, $\alpha \wedge \beta = \sum_K \left(\sum_{I \cup J, I, J\ are\ disjoint} \pm a_I b_J\right) dx^K = a_{12}b_{358}dx^{12358} + a_{24}b_{358}dx^{23458} + a_{17}b_{358}dx^{13578} + a_{12}b_{568}dx^{12568} + a_{24}b_{568}dx^{24568} + a_{17}b_{568}dx^{15678}$.

The general formula is given as $\alpha \wedge \beta\,(v_1, v_2, \dots v_{k+l}) = \frac{1}{k!l!}\sum_{\sigma \in S_{k+l}} sgn(\sigma)\alpha(v_{\sigma(1)}, v_{\sigma(2)}, \dots v_{\sigma(k)})\beta((v_{\sigma(k+1)}, v_{\sigma(k+2)}, \dots v_{\sigma(k+l)})$, this means that σ is a(k+l)−shuffle.

The general formula is given as $\alpha \wedge \beta\,(v_1, v_2, \dots v_{k+l}) = \frac{1}{k!l!}\sum_{\sigma \in S_{k+l}} sgn(\sigma)\alpha(v_{\sigma(1)}, v_{\sigma(2)}, \dots v_{\sigma(k)})\beta(v_{\sigma(k+1)}, v_{\sigma(k+2)}, \dots v_{\sigma(k+l)})$, this means that σ is a(k+l)−shuffle.

In tensor form, this is expressed as $\alpha \wedge \beta = \frac{(k+l)!}{k!l!}\mathcal{A}(\alpha \otimes \beta)$, where $\mathcal{A}$ is the skew-symmetrisation (or anti-symmetrisation) operator that takes a tensor and returns its differential form as skew-symmetric and $\otimes$ is the tensor product as defined in chapter one. This is also called anti-symmetric multilinear covectors, because it takes a number of vectors and gives a number, changing sign if we reorder its input vectors. This completes the definition of the wedge product of n-forms as calculating the volume of the parallelopiped formed by the n input vectors as column vectors in the matrix from which the determinant is calculated. Other operators can be defined as follows:

- **The inner product between vector and k-form**: given vector v and k-form α, their inner product is defined as: $i_v\alpha\,(v_1, v_2, \dots v_{k-1}) = \alpha\,(v, v_1, v_2, \dots v_{k-1})$, i.e. the resulting (k-1)-form puts the vector v in front.
- **The inner product between vector and two added k-forms**: given vector v and two k-form α & β, their inner product is defined as: $i_v(\alpha + \beta) = i_v\alpha + i_v\beta$.
- **The inner product between two vectors and k-form**: given two vectors v & w, and k-form α, their inner product is defined as: $i_{(v+w)}\alpha = i_v\alpha + i_w\alpha$.

- **Inner Product between two k**-forms: Given two one-forms α, β in $\mathbb{R}^3$: which $\in \Lambda^1\mathbb{R}^3 = \text{span}\{\text{dx}, \text{dy}, \text{dz}\} = \text{Tp}^*\mathbb{R}^3$, which is three-dimensional vector space, such that $\alpha = adx + bdy + cdz = [a, b, c]$, $\beta = rdx + sdy + tdz = [r, s, t]$, their inner product is defined as:

$$\langle \alpha, \beta \rangle = \langle [a,b,c],[r,s,t] \rangle \equiv [a,b,c] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} [r,s,t]^T = [a,b,c] \begin{bmatrix} r \\ s \\ t \end{bmatrix} = ar + bs + ct$$

  - Another example: given two two-forms η, ξ in $\mathbb{R}^3$: $\Lambda^2\mathbb{R}^3 = \text{span}\{\text{dx} \wedge \text{dy}, \text{dy} \wedge \text{dz}, \text{dz} \wedge \text{dx}\}$, which is three dimensional vector space such that $\eta = a(\text{dx} \wedge \text{dy}), \text{b}(\text{dy} \wedge \text{dz}), \text{c}(\text{dz} \wedge \text{dx}) = [\text{a}, \text{b}, \text{c}], \xi = r(\text{dx} \wedge \text{dy}), \text{s}(\text{dy} \wedge \text{dz}), \text{t}(\text{dz} \wedge \text{dx}) = [\text{r}, \text{s}, \text{t}]$; their inner product is defined as

$$\langle \eta, \xi \rangle = \langle [a,b,c],[r,s,t] \rangle \equiv [a,b,c] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} [r,s,t]^T = [a,b,c] \begin{bmatrix} r \\ s \\ t \end{bmatrix} = ar + bs + ct$$

- **Hodge star operator**: or Hodge star dual operator takes a k-form in $\mathbb{R}^n$ to $(n-k)-form$ as: $*: \Lambda^k\mathbb{R}^n \to \Lambda^{n-k}\mathbb{R}^n$, such that *α in* $\Lambda^k\mathbb{R}^n$ *has an equivalent* $*\alpha$ *in* $\Lambda^{n-k}\mathbb{R}^n$ *such that* $\alpha \wedge \beta = \langle * \alpha, \beta \rangle \sigma$ for all β, where <., .> is the inner product that associate a real number to a pair of vectors or differential forms, here both are $\Lambda^{n-k}\mathbb{R}^n$, and σ is the n dimensional volume form. Notice that <., .> if containing an one-form and a vector , it is canonical pairing between them as used earlier. This means that the Hodge star, takes an n-form as input, and uses the inner product of the n-k forms (which produces a scalar) to multiply with an n-form σ. This unique mapping between α and $*\alpha$ is valid for any chosen β. The derivation is lengthy, but interesting symmetry is noticed as $*1 = dx \wedge dy$, dx = *dy, *dy = dx in $\mathbb{R}^2$, $*1 = dx \wedge dy \wedge dz$, $*dx = dy \wedge dz$, $*dy = dz \wedge dx$, $*dz = dx \wedge dy$ in $\mathbb{R}^3$, $*1 = dx^1 \wedge dx^2 \wedge dx^3 \wedge dx^4$, $*dx^1 = dx^2 \wedge dx^3 \wedge dx^4$, $*dx^2 = dx^1 \wedge dx^3 \wedge dx^4$, $*dx^3 = dx^1 \wedge dx^2 \wedge dx^4$, *and* $*dx^4 = dx^1 \wedge dx^2 \wedge dx^3$ in $\mathbb{R}^4$, and so forth. Also Hodge star operator can be applied on k-forms in $\mathbb{R}^n$, and higher such as $*(dx \wedge dy \wedge dz) = 1$ *in* $\mathbb{R}^3$, $*(dx^1 \wedge dx^2)$ as two forms in $\mathbb{R}^{n>=2}$, $*(dx^1 \wedge dx^3)$ as two forms in $\mathbb{R}^{n>=3}$, $*(dx^1 \wedge dx^2 \wedge dx^3)$ as three forms in $\mathbb{R}^{n>=3}$, $*(dx1 \wedge dx2 \wedge dx3 \wedge dx4)$ as four forms in $\mathbb{R}^{n>=4}$and so forth for any value of k <= any value for n, both > 0.

The intuition behind these different spaces, and their implications in machine learning, is focused on multi-way associations of n features' weights on the function output that is estimated to represent a dataset. This means, instead of learning a weight for each feature/column/dimension in the dataset, you can learn a combined weight for every n-k permutation of the features, where 0 <=k <= n, and then reduce the dimensionality by ignoring the irrelevant mapping and focus on the strongly correlated mapping with the output. From the 1 to n-dimensional, the Hodge star operator uses symmetry to reduce the number of combinations required. For example, a price of a house can be a function of many features. These features are combined in multiple different factors that can be estimated. A neighbourhood factor might be combined from the distance of the nearest business area, area pollution measures, demographics of people, crime rate, and many other features related to the quality of the neighbourhood. Some other groups of features might be combined as house-specific features, such as the number of rooms, land area, built land area, renovations, number of floors, and others. Standardisation unifies the unit of measures in a dataset such that the net weight of each feature is not affected by the different measures used in collecting the data. Also, combining together different features forming one coordinate against the remaining coordinates for the remaining combinations, then estimating a weight for each new coordinate, reveal the multi-way correlations of the dataset.

3.4.2.1 Directional Derivatives

In chapter one, real-valued functions' **total derivative** $f: \mathbb{R} \to \mathbb{R}$ was introduced to describe the rate of change of the function output with respect to the displacement of a point and the slope of the tangent line at the given point. If the function is differentiable, then the total derivative is known as the gradient. This is a directional derivative in the direction of only one variable.

Also, in chapter one, the **partial/directional derivative** was defined for multivariable functions as $f: \mathbb{R}^n \to \mathbb{R}$ in the direction of a specific variable displacement, treating other variables as constants. This calculates the rate of change of the function output as the function moves in the direction of the variable used in the differentiation. A partial derivative has more options in the direction and needs a vector v_p to define it, such as $df(v_p) = v_p[f] = \lim_{t \to 0} \frac{f(p+tv_p)-f(p)}{f}$, where d is the operator that, given a function f (zero-form) as input, yields a one-form output. The directional derivative is the dot product of the gradient with the desired direction. The Jacobian matrix is the matrix of all partial derivatives explaining all ways the output and input are related, providing a total derivative from the partials. It approximates the function for a given point and estimates the change as the function moves along a vector from this point. This is achieved by dot product the Jacobian matrix with this vector yielding a vector. This vector is in the new Manifold with a new coordinate system; therefore, the Jacobian matrix is a mapping function between two coordinate systems.

Exterior differentiation d is the extended directional derivative when given zero-forms, as in vector calculus. In a coordinate-free approach, when given one-forms and higher k-forms, exterior differentiation extends the vector derivative by contracting the differential form by a given vector v, yielding a k+1-form output. Vector v provides the direction of the displacement required for the differentiation.

The local (in-coordinates x_i) exterior derivative of k-form α (which itself is defined as differential forms) is $d\alpha = \sum df_i \wedge dx_i$ where $f_i$are functions or zero-forms. For example, given $\alpha = f_1dx + f_2dy$ is a one-form on the manifold $\mathbb{R}^2$ for some functions $f_1, f_2\colon \mathbb{R}^2 \to \mathbb{R}$, then:

$$d\alpha = df_1 \wedge dx + df_2 \wedge dy$$

$$= \left(\frac{\delta f_1}{\delta x}dx + \frac{\delta f_1}{\delta y}dy\right) \wedge dx + \left(\frac{\delta f_2}{\delta x}dx + \frac{\delta f_2}{\delta y}dy\right) \wedge dy$$

$$= \frac{\delta f_1}{\delta x} \underset{=0}{\overset{dx \wedge dx}{\Longleftrightarrow}} + \frac{\delta f_1}{\delta y} \underset{=-dx\wedge dy}{\overset{dy \wedge dx}{\Longleftrightarrow}} + \frac{\delta f_2}{\delta x} dx \wedge dy + \frac{\delta f_2}{\delta y} \underset{=0}{\overset{dy \wedge dy}{\Longleftrightarrow}}$$

$$= \left(\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}\right) dx \wedge dy$$

Given two vectors $v = v_1\frac{\delta}{\delta x} + v_2\frac{\delta}{\delta y} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ and $w = w_1\frac{\delta}{\delta x} + w_2\frac{\delta}{\delta y} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, then

$d\alpha(v, w) = \left(\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}\right) dx \wedge dy(v, w)$

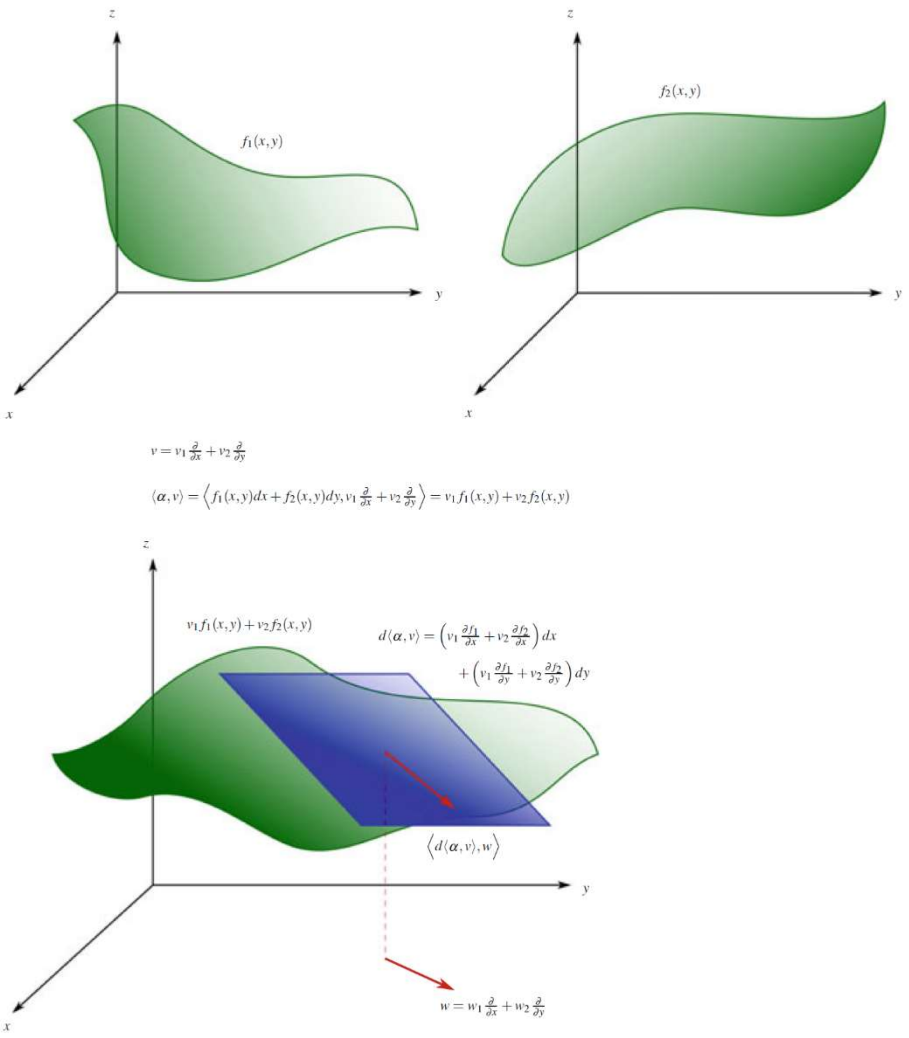
$$= \left(\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}\right) \begin{vmatrix} dx(v) & dx(w) \\ dy(v) & dy(w) \end{vmatrix}$$

$$= \left(\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}\right) \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix}$$

$$= \left(\frac{\delta f_2}{\delta x} - \frac{\delta f_1}{\delta y}\right) (v_1w_2 - w_1v_2)$$

$$= v_1w_2\frac{\delta f_2}{\delta x} - v_1w_2\frac{\delta f_1}{\delta y} - w_1v_2\frac{\delta f_2}{\delta x} + w_1v_2\frac{\delta f_1}{\delta y}$$

Exterior differentiation with constant vector fields is illustrated in Figure 19. The one form α on the manifold $\mathbb{R}^n$ is a mapping $\alpha\colon T_p\mathbb{R}^n \to \mathbb{R}$, with a given vector v on the manifold $\mathbb{R}^n$, then for each $p \in \mathbb{R}^n$, we have $\alpha_p(v_p) \in \mathbb{R}$ is a real number. Therefore $\alpha(v)$ is a function on the manifold $\mathbb{R}^n$, with input as point p and output as real numbers. This is denoted < α, v>, where the notation <·, ·> means the canonical pairing between a one-form/covector and a vector. This can be < α, (v_1, v_2, … v_k)> for k-form α.



z
$f_1(x,y)$
y
x
z
$f_2(x,y)$
y
x
$v = v_1 \frac{\partial}{\partial x} + v_2 \frac{\partial}{\partial y}$
$\langle \alpha, v \rangle = \left\langle f_1(x,y)dx + f_2(x,y)dy, v_1 \frac{\partial}{\partial x} + v_2 \frac{\partial}{\partial y} \right\rangle = v_1 f_1(x,y) + v_2 f_2(x,y)$
z
$v_1 f_1(x,y) + v_2 f_2(x,y)$
$d\langle \alpha, v \rangle = \left(v_1 \frac{\partial f_1}{\partial x} + v_2 \frac{\partial f_2}{\partial x} \right) dx$
$+ \left(v_1 \frac{\partial f_1}{\partial y} + v_2 \frac{\partial f_2}{\partial y} \right) dy$
$\left\langle d\langle \alpha, v \rangle, w \right\rangle$
y
x
$w = w_1 \frac{\partial}{\partial x} + w_2 \frac{\partial}{\partial y}$

Figure 19: The one-form $\alpha(x,y) = f_1(x,y)\, dx + f_2(x,y)\, dy$ on the manifold $\mathbb{R}^2$ is made up of two functions $f_1, f_2 \colon \mathbb{R}^2 \to \mathbb{R}$, shown above. Once we are given a vector field v on a manifold in $\mathbb{R}^3$, here the constant vector field $v = v_1\delta x + v_2 \delta y$, then this can be used to find the real-valued function $< \alpha, v > = v_1 f_1 + v_2 f_2$, which can be viewed as a linear combination of the two functions f_1, f_2. The directional derivative of this function can then be found in the direction of another given vector $w = w_1 \delta x + w_2 \delta y$. Here the differential d<α, v> in essence encodes the information about the tangent plane to the function $< \alpha, v >$, shown in green (Fortney, 2018).

The global (coordinate-free or invariant) exterior differentiation formula works well in any coordinates such as cartesian, polar, spherical, cylindrical, etc. Given 2-form α, the global exterior derivative is defined as $d\alpha\ (v, w) = v[\alpha(w)] - w[\alpha(v)] - \alpha([v, w])$, where v[f] is one notation for the directional derivative of f in the direction of v, [*v*,*w*] is the lie-bracket of two vector fields v and w, which is defined by [v,w] = vw − wv. Lie brackets will be further explained in chapter 5. This is extended to the k-form as $d\alpha\ (v_0, v_1, \dots, v_k) = \sum_i (-1)^i\, v_i[\alpha(v_0, v_1, \dots \hat{v}_i, \dots, v_k)] + \sum_{i<j} (-1)^{i+j}\, \alpha\big([v_i, v_j], (v_0, v_1, \dots \hat{v}_i, \dots, \hat{v}_j \dots, v_k)\ \big),$

such that the hat of a vector means omitting the vector. The derivation can be found in (Fortney, 2018).

For example, given non-constant vector fields $f, g\colon \mathbb{R}^2 \to \mathbb{R}$, the exterior differentiation is defined as $df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy$ and $dg = \frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy$, their wedge product is

$$df \wedge dg = \left(\frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy\right) \wedge \left(\frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy\right)$$

$$= \frac{\partial f}{\partial x}dx \wedge \left(\frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy\right) + \frac{\partial f}{\partial y}dy \wedge \left(\frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy\right)$$

$$= \frac{\partial f}{\partial x}\frac{\partial g}{\partial x}\underset{= 0}{\overset{dx \wedge dy}{\Longleftrightarrow}} + \frac{\partial f}{\partial x}\frac{\partial g}{\partial y}dx \wedge dy + \frac{\partial f}{\partial y}\frac{\partial g}{\partial x}\underset{=-dx \wedge dy}{\overset{dy \wedge dx}{\Longleftrightarrow}} + \frac{\partial f}{\partial y}\frac{\partial g}{\partial y}\underset{= 0}{\overset{dy \wedge dy}{\Longleftrightarrow}}$$

$$= \frac{\partial f}{\partial x}\frac{\partial g}{\partial y}dx \wedge dy - \frac{\partial f}{\partial y}\frac{\partial g}{\partial x}dx \wedge dy$$

$$= \left(\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial f}{\partial y}\frac{\partial g}{\partial x}\right)dx \wedge dy$$

$$= \underset{Jacobian}{\underbrace{\begin{vmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{vmatrix}}} \underset{Area\ form}{\overset{dx \wedge dy}{\Longleftrightarrow}}$$

You can see how this extends to $f, g, h\colon \mathbb{R}^3 \to \mathbb{R}$, we have:

$$df \wedge dg \wedge dh = \underset{Jacobian}{\underbrace{\begin{vmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial z} \end{vmatrix}}} \underset{Volumn\ form}{\overset{dx \wedge dy \wedge dz}{\Longleftrightarrow}}$$

For example, given $x = r\cos(\theta) and\ y = r\sin(\theta)$, then

$$dx \wedge dy = \begin{vmatrix} \frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial r} \\ \frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial r} \end{vmatrix} d\theta \wedge dr = \begin{vmatrix} \frac{\partial r\cos(\theta)}{\partial \theta} & \frac{\partial r\cos(\theta)}{\partial r} \\ \frac{\partial r\sin(\theta)}{\partial \theta} & \frac{\partial r\sin(\theta)}{\partial r} \end{vmatrix} d\theta \wedge dr$$

$$= \begin{vmatrix} -r\sin(\theta) & \cos(\theta) \\ r\cos(\theta) & \sin(\theta) \end{vmatrix} d\theta \wedge\ dr = (-r\sin^2(\theta) - r\cos^2(\theta))d\theta \wedge\ dr$$

$$= -rd\theta\ \wedge\ dr$$

This describes a polar change of coordinates. Another example of coordinate change is given the new coordinates as u = x + y and v = x – y to map points in the plane $\mathbb{R}^2_{xy} \to \mathbb{R}^2_{uv}$, such that the area in an xy-plane:

$$dx \wedge dy \Bigg(\underbrace{\begin{bmatrix}1\\0\end{bmatrix}_{(0,0)}, \begin{bmatrix}0\\1\end{bmatrix}_{(0,0)}}_{\overleftrightarrow{} \ vectors\ in\ xy-plane} \Bigg) = \begin{vmatrix}1 & 0\\0 & 1\end{vmatrix} = 1 \text{ and}$$

$$du \wedge dv \Bigg(\underbrace{\begin{bmatrix}1\\1\end{bmatrix}_{(0,0)}, \begin{bmatrix}1\\-1\end{bmatrix}_{(0,0)}}_{\overleftrightarrow{} \ mapped\ vectors\ in\ uv-plane} \Bigg) = \begin{vmatrix}1 & 1\\1 & -1\end{vmatrix} = -2,$$ we can define $du \wedge dv$ in terms of $dx \wedge dy$ as $du = d(x+y) = dx + dy$, and $dv = d(x-y) = dx - dy$, then $du \wedge dv = (dx + dy) \wedge (dx - dy)$, then simplified to $du \wedge dv = -2dx + dy$. The inverse mapping is as follows: $x = \frac{1}{2}(u+v)$, and $y = \frac{1}{2}(u-v)$, then $dx = d\left(\frac{1}{2}(u+v)\right) = \frac{1}{2}du + \frac{1}{2}dv$ and $dy = d\left(\frac{1}{2}(u-v)\right) = \frac{1}{2}du - \frac{1}{2}dv$, then $dx \wedge dy = \left(\frac{1}{2}du + \frac{1}{2}dv\right) \wedge \left(\frac{1}{2}du - \frac{1}{2}dv\right)$, then simplified to $dx \wedge dy = -\frac{1}{2}du \wedge dv$. The volume of mapped vectors as shown in Figure 20 is double the size as expected from the equations above, and the negative sign is because of the counter clockwise rotation.

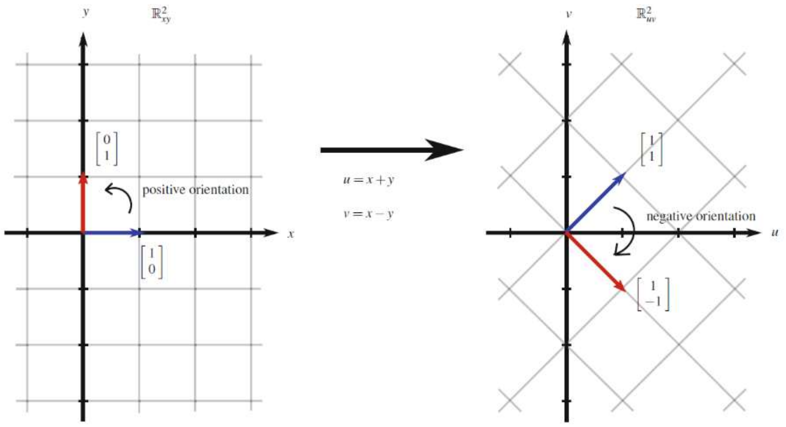

y
$\mathbb{R}^2_{xy}$
$\begin{bmatrix}0\\1\end{bmatrix}$
positive orientation
$\begin{bmatrix}1\\0\end{bmatrix}$
x
$u = x + y$
$v = x - y$
v
$\mathbb{R}^2_{uv}$
$\begin{bmatrix}1\\1\end{bmatrix}$
negative orientation
u
$\begin{bmatrix}1\\-1\end{bmatrix}$

Figure 20: The basis vectors in the xy-plane mapped to two vectors in the uv-plane. Notice the orientation changes

The above examples implicitly rely on a specific coordinate system using a single coordinate patch (U_i, ϕ_i). However, a general manifold does not have a single coordinate system. Instead, it has an atlas {(U_i, ϕ_i)} of coordinate patches (U_i, ϕ_i). The same argument and computation can be done on each of the many coordinate patches. This means that on each of the many coordinate patches we found a formula that gives the exterior derivative of a differential form d_{U_i}, as long as that form is written in the coordinates of that particular coordinate patch. For $U_i \cap U_j \neq \emptyset$, if d_{U_i} exists and is unique on Ui and d_{U_j} exists and is unique on U_j , then on $U_i \cap U_j$ we must have $d_{U_i} = d_{U_j} = d$. Since this is true on the intersection of all coordinate patches, then we have d existing and unique globally, that is, over $M = \bigcup U_i$.

Push-forward of a vector moves it from one manifold to another or the same manifold with a different coordinate system. This is often called tangent mapping. Given the coordinate change $\mathbb{R}^2_{xy} \to \mathbb{R}^2_{uv}$, mapping function $f(x, y) = (x + y, x - y) = (u, v)$ explained above, we can use the Jacobian Matrix to do the mapping from a point in $\mathbb{R}^2_{xy}$ to a point in $\mathbb{R}^2_{uv}$. Given a point p = (1, 1) and v_p from this point = (1, 2) we evaluate the Jacobian Matrix $D_pf: D_p\mathbb{R}^2_{xy} \to D_{f(p)}\mathbb{R}^2_{uv}$ for point (x, y):

$$\begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix}_{(x,y)} = \begin{vmatrix} \frac{\partial(x+y)}{\partial x} & \frac{\partial(x+y)}{\partial y} \\ \frac{\partial(x-y)}{\partial x} & \frac{\partial(x-y)}{\partial y} \end{vmatrix}_{(x,y)} = \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}_{(x,y)},$$ then apply the mapping on v_p: to get its mapping (push-forward) in $\mathbb{R}^2_{uv}$ for the mapped point f(p):

$$\begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix}_{(x,y)} = \begin{vmatrix} \frac{\partial(x+y)}{\partial x} & \frac{\partial(x+y)}{\partial y} \\ \frac{\partial(x-y)}{\partial x} & \frac{\partial(x-y)}{\partial y} \end{vmatrix}_{(x,y)} = \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}_{(x,y)},$$ then apply the mapping on v_p: to get its mapping (push-forward) in $\mathbb{R}^2_{uv}$ for the mapped point f(p):

$$D_pf.v_p = \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}_{(1,1)} . \begin{bmatrix} 1 \\ 2 \end{bmatrix}_{(1,1)} = \begin{bmatrix} 1.1 + 1.2 \\ 1.1 \pm 1.2 \end{bmatrix}_{f(1,1)} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}_{f(1,1)} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}_{(2,0)},$$ this push-forward changed the basis from point (1, 1) to the mapped point (2,0). This is generalised as the Jacobian mapping at point p as: $$D_pf = \begin{vmatrix} \left.\frac{\partial f_1}{\partial x_1}\right|_p & \left.\frac{\partial f_1}{\partial x_2}\right|_p & \cdots & \left.\frac{\partial f_1}{\partial x_n}\right|_p \\ \left.\frac{\partial f_2}{\partial x_1}\right|_p & \left.\frac{\partial f_2}{\partial x_2}\right|_p & \cdots & \left.\frac{\partial f_2}{\partial x_n}\right|_p \\ \vdots & \vdots & \ddots & \vdots \\ \left.\frac{\partial f_n}{\partial x_1}\right|_p & \left.\frac{\partial f_n}{\partial x_2}\right|_p & \cdots & \left.\frac{\partial f_n}{\partial x_n}\right|_p \end{vmatrix}$$

Pull-back of a differential form moves it from one manifold to another manifold or the same manifold with a different coordinate system. This is often called the cotangent mapping D_p^*f, and is dual to the push-back of vectors. Given the coordinate change $\mathbb{R}^2_{xy} \to \mathbb{R}^2_{uv}$, mapping function f explained above,

$$\xLeftrightarrow[\text{pull−back of } (du \wedge dv)]{D_p^*f.(du \wedge dv),}$$

$$\underbrace{\begin{vmatrix}1 & 1\\ 1 & -1\end{vmatrix}.(dx \wedge dy)_p\left(\underbrace{\begin{bmatrix}1\\0\end{bmatrix}_p,\begin{bmatrix}0\\1\end{bmatrix}_p}_{vectors\ in\ D_p\mathbb{R}^2_{xy}}\right)}_{=(-2)(1)=-2} = \underbrace{(du \wedge dv)_{f(p)}\left(\underbrace{D_pf.\begin{bmatrix}1\\0\end{bmatrix}_p,D_pf.\begin{bmatrix}0\\1\end{bmatrix}_p}_{\substack{push-forward\ of\ vectors\\ in\ D_p\mathbb{R}^2_{xy}\to D_{f(p)}\mathbb{R}^2_{uv}}}\right)}_{=-2}$$

This is simplified to $-2dx \wedge dy = du \wedge dv$ such that the $-2dx \wedge dy$ is the pull-back of $du \wedge dv$. This is generalised to $\mathbb{R}^n$, given the mapping function (change in basis) $\varphi\colon \mathbb{R}^n_{(x_1,\dots,x_n)} \to \mathbb{R}^n_{(\varphi_1,\dots,\varphi_n)}$ and a volume form $d\varphi_1 \wedge \dots \wedge d\varphi_n$, the pull-back is defined as (omitting the point p):

$$D^*\varphi.\left(d\varphi_1 \wedge \dots \wedge d\varphi_n\right) = \begin{vmatrix}\frac{\partial\varphi_1}{\partial x_1} & \frac{\partial\varphi_1}{\partial x_2} & \cdots & \frac{\partial\varphi_1}{\partial x_n}\\ \frac{\partial\varphi_2}{\partial x_1} & \frac{\partial\varphi_2}{\partial x_2} & \cdots & \frac{\partial\varphi_2}{\partial x_n}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial\varphi_n}{\partial x_1} & \frac{\partial\varphi_n}{\partial x_2} & \cdots & \frac{\partial\varphi_n}{\partial x_n}\end{vmatrix} dx_1 \wedge \dots \wedge dx_n$$

This is for mapping between Manifolds of the same dimensions and for volume forms only, not the general differential forms. A general map $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$ is defined as $\varphi(x_1,\dots,x_n) = \varphi^1(x_1,\dots,x_n),\dots,\varphi^m(x_1,\dots,x_n)$, and the Jacobian $D\varphi = \begin{vmatrix}\frac{\partial\varphi_1}{\partial x_1} & \frac{\partial\varphi_1}{\partial x_2} & \cdots & \frac{\partial\varphi_1}{\partial x_n}\\ \frac{\partial\varphi_2}{\partial x_1} & \frac{\partial\varphi_2}{\partial x_2} & \cdots & \frac{\partial\varphi_2}{\partial x_n}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial\varphi_m}{\partial x_1} & \frac{\partial\varphi_m}{\partial x_2} & \cdots & \frac{\partial\varphi_m}{\partial x_n}\end{vmatrix}$ and input k-form $\alpha \in \Lambda^k\mathbb{R}^n$, the pull-back yields a k-form $\in \Lambda^k\mathbb{R}^m$ is defined as follows:

$$(D^*\varphi.\alpha)(v_1,\dots,v_k) \equiv \alpha\left(D\varphi.v_1,\dots,D\varphi.v_k\right)$$

For the above simple linear mapping example, we have: $D^*\varphi.(vdu + udv) = 2xdx + 2ydy$. The derivation and more examples of other coordinate mappings can be found in (Fortney, 2018).

In chapter one, vector spaces are defined as the set of vectors, including the zero vector and another vector that, when added or multiplied by a scalar, produces vectors that belong to this space. **Vector fields** are defined above as sections of manifolds because they provide a mapping between each vector in a given vector space and a point. In $\mathbb{R}^3$, a vector field $\mathbb{F}$ is defined as $\mathbb{F}$ = P(x, y, z)i, Q(x, y, z)j, R(x, y, z)k, such that $P, Q, R\colon \mathbb{R}^3 \to \mathbb{R}$ and I, j, k are the unit vectors in x, y, and z coordinates, defined as e_1, e_2, and e_3, respectively. The operator

Δ is defined as $\Delta = \frac{\partial}{\partial x}e_1 + \frac{\partial}{\partial y}e_2 + \frac{\partial}{\partial z}e_3$, which is the gradient when applied on a function (zero-form), turning it into a vector field: $\Delta f = \frac{\partial f}{\partial x}i + \frac{\partial f}{\partial y}j + \frac{\partial f}{\partial z}k = \frac{\partial f}{\partial x}\frac{\partial}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial}{\partial y} + \frac{\partial f}{\partial z}\frac{\partial}{\partial z}$.

In chapter one, the **integration in the vector calculus** was defined using the Riemann sums operator$\int f(x)dx$. Similarly, we can define the **integration of differential forms** as a generalisation of the vector calculus integration to differential forms. For example, in $\mathbb{R}^2$, we integrate the differential form $\alpha = f(x,y)\, dx \ \wedge dy$, as $\int\int f(x,y)\, dx \ \wedge \ dy$, which keeps track of the orientation (order/sign of the coordinates) and generalises to $\mathbb{R}^n$ as $\int \ldots \int f(x_1, \ldots, x_n)\, dx_1 \ \wedge \ldots \wedge \ dx_n$. In the coordinate-free approach, the integration over differential forms works when given a mapping $\mathbb{R}^n_{x_1,\ldots,x_n} \overset{\phi}{\rightarrow} \mathbb{R}^n_{\phi_1,\ldots,\phi_n}$ and is defined as :

$\int_R \ f(x_1, \ldots, x_n)\, dx_1 \ \wedge \ldots \wedge \ dx_n = \int_{\phi(R)} f \circ \phi^{-1}(\phi_1, \ldots, \phi_n)\, T^*\phi^{-1}.(dx_1 \ \wedge \ldots \wedge \ dx_n)$,
such that the left-hand side shows the integral taking place in $x_1, \ldots, x_n -$ coordinates and integrates the function $f(x_1, \ldots, x_n)$ over the region R using the volume form $dx_1 \ \wedge \ldots \wedge$ dx_n associated with the $x_1, \ldots, x_n -$ coordinates. The right-hand side shows the region we are integrating over in $\mathbb{R}^n_{\phi_1,\ldots,\phi_n}$ is its image $\phi(R)$. The function $f \circ \phi^{-1}$ is a function in the variables $\phi_1, \ldots, \phi_n$, and pull-back of the area-form $T\phi^{-1}.(dx_1 \ \wedge \ldots \wedge \ dx_n)$, and not the area form $d\phi_1 \ \wedge \ldots \wedge \ d\phi_n$, which is essential when a change of variables is needed.

Divergence measures how vector fields vary, diverge or spread out at a given point. This is defined for $\mathbb{F}$ as the dot product of Δ with the vector field F in cartesian coordinates (x, y, z):

$$div\ \mathbb{F} = \Delta.\mathbb{F} = \left(\frac{\partial}{\partial x}e_1 + \frac{\partial}{\partial y}e_2 + \frac{\partial}{\partial z}e_3\right).(Pe_1 + Qe_2 + Re_3) = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}$$

The flux of the vector field $\mathbb{F}$ through the surface S is a measure of vector flow over a surface. This can measure fluid flow over a surface, electricity flow, magnetic field flow, and others. To abstract this flow without fluid density or other physical interpretations for a given problem, the flux at each point *p* on the surface *S* we have that $\mathbb{F}_p.\hat{n}_p$ is a real number. That is, we can think of $\mathbb{F}.\hat{n}$ as a real-valued function on *S*, $\mathbb{F}.\hat{n} : S \rightarrow \mathbb{R}$, which means we can easily integrate it over the surface *S*.

$$Flux\ of\ \ \mathbb{F} = \lim_{|\Delta S| \to 0} \sum_i \mathbb{F}.\hat{n}\Delta S = \int_S \mathbb{F}.\hat{n}dS$$
dS comes from the *ΔS*, which represents the area-form of a small bit of surface *S*.

This derives the divergence at a given point in a surface as follows. Given a small three-dimensional region *V* about the point *(x_0, y_0, z_0)* with boundary *∂V* and volume *ΔV* the divergence of $\mathbb{F}$ at *(x_0, y_0, z_0), and ∂V* is a closed surface like a sphere with no edges*, and* the normal vector $\hat{n}$ point outwards from the *∂V* surface, is defined by:

$$div\ \mathbb{F} = \lim_{\Delta V \to 0} \frac{1}{\Delta V} \int_{\partial V} \mathbb{F}.\hat{n} dS$$

In cylindrical coordinates (r, θ, z), the divergence of vector $field\ \mathbb{F} = \mathbb{F}_r e_r + \mathbb{F}_\theta e_\theta + \mathbb{F}_z e_z$ is defined as:

$$div\ \mathbb{F} = \frac{1}{r}\frac{\partial(r\mathbb{F}_r)}{\partial r} + \frac{1}{r}\frac{\partial \mathbb{F}_\theta}{\partial \theta} + \frac{\partial \mathbb{F}_z}{\partial z}$$

In spherical coordinates (r, θ, φ), the divergence of vector field $\mathbb{F} = \mathbb{F}_r e_r + \mathbb{F}_\theta e_\theta + \mathbb{F}_\varphi e_\varphi$ is defined as:

$$div\ F = \frac{1}{r^2}\frac{\partial(r^2\mathbb{F}_r)}{\partial r} + \frac{1}{r\sin(\theta)}\frac{\partial(\sin(\theta)\,\mathbb{F}_\theta)}{\partial \theta} + \frac{1}{rsi\,\mathrm{n}(\theta)}\frac{\partial \mathbb{F}_\varphi}{\partial \varphi}$$

Curl also measures how vector fields vary, such as the "circulation" per unit area of vector field F over an infinitesimal path around some point. For the same vector field defined above for the cartesian coordinates, the curl is defined as the cross product for the same operator Δ defined above and the vector field:

$$curl\ \mathbb{F} = \Delta \times \mathbb{F} = \left(\frac{\partial R}{\partial y} - \frac{\partial Q}{\partial z}\right)i + \left(\frac{\partial P}{\partial z} - \frac{\partial R}{\partial x}\right)j + \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right)k$$

Given S as a surface bounded by the closed curve C = ∂S, ΔS is the area of that surface, $\hat{n}$ is the unit normal vector to that surface, the s in ds is the infinitesimal arc length element, and the surface area ΔS shrinks to zero about the point (x_0, y_0, z_0), and $\hat{t}$ as the unit tangent vectors to *C*. Then, the curl $\mathbb{F}$ at a point (x_0, y_0, z_0) is defined as:

$$\hat{n}.curl\ \mathbb{F} = \lim_{|\Delta S| \to 0} \frac{1}{\Delta S} \int_{C} \mathbb{F}.\hat{t}ds$$

The definition of curl derives the Stokes' theorem. Given any surface *S*, not necessarily in a plane, whose boundary is the closed curve *C*, we can break up at surface into sub-surfaces S_i with boundaries C_i, if two C_i share an edge, the terms cancel out, and we end up with the Stokes theorem stated below.

In cylindrical coordinates (r, θ, z), the curl of vector field $\mathbb{F} = \mathbb{F}_r e_r + \mathbb{F}_\theta e_\theta + \mathbb{F}_z e_z$ is defined as:

$curl\ \mathbb{F} = (curl\ \mathbb{F})_r e_r + (curl\ \mathbb{F})_\theta e_\theta + (curl\ \mathbb{F})_z e_z$, where:

$(curl\ \mathbb{F})_r = \frac{1}{r}\frac{\partial \mathbb{F}_z}{\partial \theta} - \frac{\partial \mathbb{F}_\theta}{\partial z}$,

$(curl\ \mathbb{F})_\theta = \frac{\partial \mathbb{F}_r}{\partial z} - \frac{\partial \mathbb{F}_z}{\partial r}$, and

$$(curl\ \mathbb{F})_z = \frac{1}{r}\frac{\partial(r\mathbb{F}_\theta)}{\partial r} - \frac{1}{r}\frac{\partial \mathbb{F}_r}{\partial \theta}$$

In spherical coordinates (r, θ, φ), curl of vector field $\mathbb{F} = \mathbb{F}_r e_r + \mathbb{F}_\theta e_\theta + \mathbb{F}_\varphi e_\varphi$ is defined as:

$curl\ \mathbb{F} = (curl\ \mathbb{F})_r e_r + (curl\ \mathbb{F})_\theta e_\theta + (curl\ \mathbb{F})_\varphi e_{z\varphi}$, where:

$(curl\ \mathbb{F})_r = \frac{1}{r\, si\,\mathrm{n}(\theta)}\frac{\partial(\sin(\theta)\mathbb{F}_\varphi)}{\partial\theta} - \frac{1}{r\, si\ \ (\theta)}\frac{\partial \mathbb{F}_\theta}{\partial\varphi}$,

$(curl\ \mathbb{F})_\theta = \frac{1}{r\, si\,\mathrm{n}(\theta)}\frac{\partial \mathbb{F}_r}{\partial\varphi} - \frac{1}{r}\frac{\partial(r\mathbb{F}_\varphi)}{\partial r}$, and

$$(curl\ \mathbb{F})_\varphi = \frac{1}{r}\frac{\partial(r\mathbb{F}_\theta)}{\partial r} - \frac{1}{r}\frac{\partial \mathbb{F}_r}{\partial \theta}$$

The gradient of function f, is the vector field:

$grad\ f = \Delta. f = \frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j + \frac{\partial f}{\partial z} k$, this can be a dot product with a vector u to give the directional derivative in the direction of u: $gradf.u = u[f]$.

The Laplacian of a function f is defined to be $div(grad\ f) = \nabla \cdot (\nabla f) = \nabla \cdot \nabla f = \nabla^2 f$, which is the divergence of the gradient that is the trace (tr) of the function's Hessian, H(f). tr(H(f)) is the sum of the eigenvalues of the Hessian and is invariant of change of basis and a measure of the function curvature. More on this will be discussed in chapter five.

The **flat ♭ and the sharp ♯** operators are called musical isomorphisms because the symbols are taken from musical note notations. The flat ♭ in musical notes means "lower the pitch", and sharp ♯ means "raise the pitch". Similarly, the ♭ isomorphism means "lower the indices" by going from the tangent space TpM to the cotangent space Tp^*M, such that a vector v is mapped as in $v^\flat : v^I \frac{\partial}{\partial x^I} \to v_i dx^i$. The ♯ isomorphism means "raise the indices" by going from the cotangent space Tp^*M to the tangent space TpM, such that a differential form α is mapped as in $\alpha^\sharp : \alpha_i dx^i \to \alpha^I \frac{\partial}{\partial x^I}$.

the sharp and flat operators

Given a vector field $\mathbb{F} = P\frac{\partial}{\partial x}, Q\frac{\partial}{\partial y}, R\frac{\partial}{\partial z}$, we will define a **Hodge star mapping** ($*\circ\flat$) as first flattening the vector field to get a one-form and then Hodge staring that one-form to get a two form: $(*\circ\ \flat)\mathbb{F} = *\left(\mathbb{F}^\flat\right) = *\left(\left(P\frac{\partial}{\partial x}, Q\frac{\partial}{\partial y}, R\frac{\partial}{\partial z}\right)^\flat\right) = *\left(Pdx + Qdy + Rdz\right) = Pdy \wedge dz + Q\ dz \wedge dx + R\ dx \wedge dy.$

3.4.2.2 Summary of Mappings & Generalised Stokes Theorem:

In chapter 9 in (Fortney, 2018), relationships between differential calculus and vector calculus are summarised in this diagram in Figure 21 for $\mathbb{R}^3$. The mappings between each space are shown in the arrows. It is obvious that the identity (id) mapping is used between continuous function and the zero form on the same Manifold because they describe the same thing. The grad mapping applies the gradients of a continuous function to provide the tangent bundle. The differential operator moves the one-form differentials to their two-form equivalent, while the flat operator ♭ move the tangent bundle to the two-form equivalent.

$$\begin{tikzcd}
\text{Space of continuous functions on the manifold } (\mathbb{R}^3) & C(\mathbb{R}^3) \arrow[r, "\text{grad}"] \arrow[d, "\text{id}"] & T\mathbb{R}^3 \arrow[d, "\flat"] & \text{Tangent Bundle (all vector fields) of } (\mathbb{R}^3) \\
\text{The zero-forms on } (\mathbb{R}^3) & \Lambda^0(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^1(\mathbb{R}^3) & \text{The one-forms on } (\mathbb{R}^3)
\end{tikzcd}$$

Figure 21: Illustration of relationships between differential calculus and vector calculus

The “diagram commutes” from C($\mathbb{R}^3$) to $\Lambda^1(\mathbb{R}^3)$ through two paths: $C(\mathbb{R}^3) \xrightarrow{(grad)} T\mathbb{R}^3 \xrightarrow{\flat} \Lambda^1(\mathbb{R}^3)$ and C(ℝ^3) $\xrightarrow{\text{id}} \Lambda^0(\mathbb{R}^3) \xrightarrow{\text{d}} \Lambda^1(\mathbb{R}^3)$. In other words, given a continuous function f, we have $(grad\ f\,)^\flat = d\big(id(f)\big) = df$. The left-hand side says that flattening the gradient of the continuous function is equivalent to the exterior derivative of the function.

$$\Delta f = \frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j + \frac{\partial f}{\partial z} k = \frac{\partial f}{\partial x}\frac{\partial}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial}{\partial y} + \frac{\partial f}{\partial z}\frac{\partial}{\partial z}$$

$$(\Delta f)^\flat = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial z} dz$$

$$df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial z} dz = (\Delta f)^\flat$$

Another commuting diagram shown in Figure 22 shows two paths that illustrate that given a vector Field $\mathbb{F} \in T\mathbb{R}^3$, the differential of flattened $\mathbb{F}$ is equivalent to Hodge star of the flattened curl of $\mathbb{F}$: $d(\mathbb{F}^\wedge \flat\ ) = *\,((curl\ \mathbb{F})^\flat\,) = (*\circ \flat\,)(curl\ \mathbb{F})$. You can work out the mapping operators on the arrows to derive this identity or check the detailed derivation steps in (Fortney, 2018).

$$\begin{tikzcd}
\text{Tangent Bundle (all vector fields) of } (\mathbb{R}^3) & T\mathbb{R}^3 \arrow[r, "\text{curl}"] \arrow[d, "\flat"] & T\mathbb{R}^3 \arrow[d, "*\circ\ \flat"] & \text{Tangent Bundle (all vector fields) of } (\mathbb{R}^3) \\
\text{The on-forms on } (\mathbb{R}^3) & \Lambda^1(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^2(\mathbb{R}^3) & \text{The two-forms on } (\mathbb{R}^3)
\end{tikzcd}$$

Figure 22: Another commuting diagram

Another commuting diagram shown in Figure 23 shows two paths that illustrate that given a vector Field $\mathbb{F} \in T\mathbb{R}^3$, the Hodge star of divergence of $\mathbb{F}$ is equivalent to the differential of the Hodge star of the flattened $\mathbb{F}$: $*\,(div\ \mathbb{F}) = d\big((*\circ\ \ \flat\,)\mathbb{F}\big) = d\big(*\,\mathbb{F}^{\flat}\,\big)$. You can work out the mapping operators on the arrows to derive this identity or check the detailed derivation steps in (Fortney, 2018).

$$\begin{tikzcd}
\text{Tangent Bundle (all vector fields) of } (\mathbb{R}^3) & T\mathbb{R}^3 \arrow[r, "\text{div}"] \arrow[d, "*\circ\ \flat"] & C(\mathbb{R}^3) \arrow[d, "*"] & \text{Space of continuous functions on the manifold } (\mathbb{R}^3) \\
\text{The two-forms on } (\mathbb{R}^3) & \Lambda^2(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^3(\mathbb{R}^3) & \text{The three-forms on } (\mathbb{R}^3)
\end{tikzcd}$$

Figure 23: Another commuting diagram

$$\begin{tikzcd}
C(\mathbb{R}^3) \arrow[r, "\text{grad}"] \arrow[d, "\text{id}"] & T\mathbb{R}^3 \arrow[r, "\text{curl}"] \arrow[d, "\flat"] & T\mathbb{R}^3 \arrow[r, "\text{div}"] \arrow[d, "*\circ \flat"] & C(\mathbb{R}^3) \arrow[d, "*"] \\
\Lambda^0(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^1(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^2(\mathbb{R}^3) \arrow[r, "d"] & \Lambda^3(\mathbb{R}^3)
\end{tikzcd}$$

Figure 24: Connecting the three commuting diagrams

Figure 24 connects the three commuting diagrams together, illustrating that the three vector calculus operators: the gradient, the curl and the divergence on top, are all different forms of exterior differentiations on the bottom of the diagram. The vector calculus cannot generalise to $\mathbb{R}^n$, while the differential calculus does.

Poncaré lemma is a powerful tool for the study of manifolds. It states that every closed form on $\mathbb{R}^n$ is exact. A differential form α is closed if dα = 0. A differential k-form α is exact if there is another k-1-form differential form β such that α = dβ.

The following identities from vector calculus can be written in terms of the exterior derivative as follows from the Poncaré lemma: $\nabla \times (\nabla f) = 0 \leftrightarrow d(df) = 0$ and $\nabla \times (\nabla F) = 0 \leftrightarrow d(d\alpha) = 0$.

The fundamental theorem of line integrals, given a curve C, given by $c(s) = (x(s), y(s), z(s))$ with endpoints $c(a) = (x(a), y(a), z(a))$, and $c(b) = (x(b), y(b), z(b))$, where a, b ∈ $\mathbb{R}$ and a ≤ b, then the fundamental theorem of line integrals is given by $\int (grad\ f).ds = f\big(c(b)\big) - f(c(a))$, $grad\ f$ produces the vector field F, and flattening it is equal to the differential of it as $(grad\ f)^\flat = df$. We can also write the boundary of curve C as $\partial C = \{c(b) - c(a)\}$. Combining these, we can write the theorem of line integral as: $\int_c\ df = \int_{\partial C} f$.

Also, since f is a zero-form, which we could denote as α, and C is a one-dimensional manifold M we could rewrite the fundamental theorem of line integrals as: $\int_M\ d\alpha = \int_{\partial M} \alpha$.

The **vector calculus version of Stokes' theorem** states that: $\int_S\ curl\ F.\hat{n}dS = \int_{\partial S} F.\acute{c}(s)ds$.

Similar to the line integral, the right-hand of Stokes' theorem can be rewritten in terms of $\int_{\partial S} F^\flat$ And similar derivation of the left-hand side, and by replacing the vector field F by the vector field curl F, to reach: $\int_S\ curl\ F.dS = \int_S * ((curl\ F)^\flat) = \int_S\ d(F^\flat)$.

Since $F^\flat$ is a one-form α, and S is a two-dimensional manifold M, which means we have arrived again at the Stokes' theorem as above: $\int_M d\alpha = \int_{\partial M} \alpha$.

The **vector calculus version of the divergence theorem** states that: $\int_V \ div\, F\, dV = \int_{\partial V} F.dS$.

Again, the integrand on the left-hand side is equivalent to $*\,(div\, \boldsymbol{F})$, which can be rewritten as $d\big((* \circ \ \flat\,)F\big)$, and the right-hand side is equivalent to the right-hand side as follows: $\int_V \ d(\, * \circ \ \flat\,)F) \ = \int_{\partial V} * \,(F^{\,\flat}\,)$.

Writing the three-dimensional manifold V as M and the three two-form $(* \circ \ \flat\,)F$ as α, we have the divergence theorem as the generalised Stokes Theorem above: $\int_M \ d\alpha = \int_{\partial M} \alpha$.

- This means that the line integrals, Stokes theorem, and the divergence theorem are all special cases of what is called the generalized Stokes theorem. At this stage, calculus on manifolds is introduced well enough, and the reader is ready to study differential geometry if required. Differential geometry is important for machine learning to achieve independence in the representation of the dataset. These are called the non-parametric models, such as kernels, Gaussian processes, Bayesian non-parametric, and VC dimensions. Chapter five will explain further representation theory and its applications in machine learning. A more advanced representation invariance such as homotopy can be encodable in deep learning (Haarmann *et al.*, 2014)

Python notebook ch3.ipynb show examples of calculus in differential geometry operations on manifolds using the Sympy (Python symbolic math) package. Other Python packages provide various functions from differential geometry, and geometric statistics, such as the following:

- The Geomstats: https://geomstats.github.io/.
- The Geometric Algebra: https://galgebra.readthedocs.io/.

3.4.3 Tensors on Manifolds/Tensor Transformation Rules

Generally, Tensors are defined in Physics as functions which eat a certain number of vectors (known as the rank r of the tensor) and produce a number. These input vectors are Nd-arrays indices, and the output number is the value stored in this index location. This is very much the definition of differential forms. The multilinearity in Tensors means that the tensor function is linear in each of its r arguments in terms of the values of the functions/components on the r basis vectors. For example, a rank-2 tensor eats two vectors, v and w, such that multilinearity means:

T $(v_1 + cv_2,w)$ = T (v_1,w) + cT (v_2,w)

T $(v,w_1 + cw_2)$ = T (v,w_1) + cT (v,w_2)

For vector basis/coordinate for our vector space, say $\hat{x}, \hat{y}$, and $v = v_x\hat{x} + v_y\hat{y}$, $w = w_x\hat{x} + w_y\hat{y}$, then

$$T(v,w) = T\big(v_x\hat{x} + v_y\hat{y}, w_x\hat{x} + w_y\hat{y}\big) = v_xT\big(\hat{x}, w_x\hat{x} + w_y\hat{y}\big) + v_yT\big(\hat{y}, w_x\hat{x} + w_y\hat{y}\big)$$
$$= v_xw_xT(\hat{x},\hat{x}) + v_xw_yT(\hat{x},\hat{y}) + v_yw_xT(\hat{y},\hat{x}) + v_yw_yT(\hat{y},\hat{y})$$

Which reduces to coordinate projections on different combinations of the coordinates. The components $(v_xw_x, v_xw_y, \ldots)$ are the tensors evaluation on the coordinate system $(T(\hat{x},\hat{x}), T(\hat{x},\hat{y}), \ldots)$ or denoted $(T_{x,x}, T_{x,y}, \ldots)$. This derives the tensor transformation laws to the new coordinate system, as reevaluating the new components on the basis of the new coordinate $\hat{x}', \hat{y}'$, using mapping A. Then deriving the new components is performed as $\hat{x}' = A_{x',x}\hat{x} + A_{x',y}\hat{y},\ \hat{y}' = A_{y',x}\hat{x} + A_{y',y}\hat{y}$. This is interpreted as not affecting the action of Tensor T, as it exists independently of its coordinate system, and its components can be reevaluated in terms of the new coordinate system as the matrix multiplication:

$$T(v,w) = \begin{bmatrix} v_x & v_y \end{bmatrix} \begin{bmatrix} T_{x,x} & T_{x,y} \\ T_{y,x} & T_{y,y} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix}$$

For an example from Physics that has applications in computer graphics, robotics navigation control and others, the moment of Inertia tensor I is a quantity that determines the rotational force (torque) required to achieve the required angular acceleration, as explained in the second law of motion. It is a scalar for a rotation around an axis perpendicular to a plane (2D rotation), $I = mr^2 = \int (r^2) dm$, where m is the object mass, and r is the distance to the rotational axis. To freely rotate around three axes, creating a 3D rotation, the Intertia I becomes a symmetric 3x3 matrix (2nd rank tensor) that captures the fact that a torque around one axis can create acceleration on other axes, is derived as,

$$I = \begin{bmatrix} \int (r_z^2 + r_y^2)dm & -\int (r_x^2 + r_y^2)dm & -\int (r_x^2 + r_z^2)dm \\ -\int (r_y^2 + r_x^2)dm & \int (r_z^2 + r_x^2)dm & -\int (r_y^2 + r_z^2)dm \\ -\int (r_z^2 + r_x^2)dm & -\int (r_z^2 + r_y^2)dm & \int (r_x^2 + r_y^2)dm \end{bmatrix}$$

This matrix can be factored into a rotational matrix R (which is the Eigenvectors of I matrix) and diagonal matrix D (the eigenvalues of I matrix) , I=RDR^T. Applying this tensor on two copies of an angular velocity vector w, produces the Kinetic Energy KE computed as $KE = \frac{1}{2}I(w,w) = \frac{1}{2}\begin{bmatrix} w_x & w_y & w_z \end{bmatrix}\begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$

The I Inertia tensor can also be used as a linear operator that can be applied on the angular velocity vector w to produce the angular Momentum $L = Iw = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$

Tensors on a manifold are defined as multilinear mapping:

$$T: T^*M \times \ldots \times T^*M \times TM \times \ldots TM \to \mathbb{R}$$

$$\underset{\substack{r\ contravariant \\ degree}}{\Longleftrightarrow} \underset{\substack{s\ covariant \\ degree}}{\Longleftrightarrow}$$

$$T(\alpha_1, \ldots \alpha_r, v_1, \ldots, v_s) \to \mathbb{R}$$

This map T takes r one-forms and s vectors as input and produces a real number. This means this tensor of rank (r, s), i.e. r contravariant degrees and s covariant degrees.

Rank-1 Tensors are either covariant tensors or covariant tensors.

$T: TM \to \mathbb{R}$ is a Rank (0, 1)-tensor, also called a **rank-1 covariant tensor**. It is a linear mapping of a **vector to a real number**, which is equivalent to the **differential one-form** explained previously. This means that $T \in T^*M$, and expressed in terms of its components $T_i$and basis vectors dx^i as $T = T_1dx^1 + \cdots + T_ndx^n = \sum_{i=1}^n T_idx^i = T_idx^i$. A change of basis/coordinate functions from $(x^1, \ldots, x^n)$ to $(u^1, \ldots, u^n)$ using n functions on the same Manifold as: $u^i(x^1, \ldots, x^n) = u^i$ for I from 1 to n. The new components in the new coordinates are $T = \tilde{T}_1du^1 + \cdots + \tilde{T}_ndu^n$. This transformation mapping is captured in the Jacobian, and the transformed components can be calculated as follows:

$$\begin{bmatrix}\tilde{T}_1\\ \vdots \\ \tilde{T}_n\end{bmatrix} = \begin{bmatrix}\frac{\partial x^1}{\partial u^1} & \cdots & \frac{\partial x^n}{\partial u^1}\\ \vdots & \ddots & \vdots\\ \frac{\partial x^1}{\partial u^n} & \cdots & \frac{\partial x^n}{\partial u^n}\end{bmatrix}\begin{bmatrix}T_1\\ \vdots \\ T_n\end{bmatrix}$$

Such that $\tilde{T}_i = \frac{\partial x^j}{\partial u^i} T_j$. The tensor T itself does not change, but its coordinate basis and components change.

$T: T^*M \to \mathbb{R}$ is a Rank (1, 0)-tensor, also called a **rank-1 contravariant tensor**. It is a linear mapping of a **one-form paired with a vector $< \alpha, v >$ to a real number**, which is equivalent to the **vector field** explained previously. It is expressed in terms of its components and basis $\frac{\partial}{\partial x^i}$ as: $T \ = T^1 \frac{\partial}{\partial x^1} + \cdots + T^n \frac{\partial}{\partial x^n} = \sum_{i=1}^{n} T^i \frac{\partial}{\partial x^i} = T^i \frac{\partial}{\partial x^i}$. Notice that hat the components of the previous covariant tensor were indicated with lower indices, and the components of this contravariant tensor are indicated by upper indices when using Einstein summation notation. Previously the *i* in $\frac{\partial x^j}{\partial u^i}$ is regarded as a lower index of the whole term even though it is an upper index of the *u because it is in the denominator*, while the *j* is considered as an upper index of the term because it is in the numerator.
To transform the basis of this contravariant tensor from $\frac{\partial}{\partial x^i}$ to $\frac{\partial}{\partial u^i}$, we need the mappings from the Jacobian matrix to compute the new components, such as:

$$\begin{bmatrix}\tilde{T}^1\\ \vdots \\ \tilde{T}^n\end{bmatrix} = \begin{bmatrix}\frac{\partial u^1}{\partial x^1} & \cdots & \frac{\partial u^1}{\partial x^n}\\ \vdots & \ddots & \vdots\\ \frac{\partial u^n}{\partial x^1} & \cdots & \frac{\partial u^n}{\partial x^n}\end{bmatrix}\begin{bmatrix}T^1\\ \vdots \\ T^n\end{bmatrix}$$

Such that $\tilde{T}^i = \frac{\partial u^i}{\partial x^j} T^j$. The basis elements of T^*M transform as follows:

$$\begin{bmatrix}du^1\\ \vdots \\ du^n\end{bmatrix} = \begin{bmatrix}\frac{\partial u^1}{\partial x^1} & \cdots & \frac{\partial u^1}{\partial x^n}\\ \vdots & \ddots & \vdots\\ \frac{\partial u^n}{\partial x^1} & \cdots & \frac{\partial u^n}{\partial x^n}\end{bmatrix}\begin{bmatrix}dx^1\\ \vdots \\ dx^n\end{bmatrix}$$, such that $du^i = \frac{\partial u^i}{\partial x^j} dx^j$

For Rank-2 tensors, we have three possibilities:
1. **(0, 2)-Tensors (Rank-Two Covariant Tensor):** $T: TM \times TM \to \mathbb{R}$, such that two-forms are a subset. It takes two vectors and produces a real number. $T \in T^*M \times T^*M = span\{dx^i \otimes dx^j | 1 \leq i, j, \leq n\}$. If M is an $\mathbb{R}^2$Manifold, then $T = \ T_{11} dx^1 \times dx^1 + T_{12} dx^1 \times dx^2 + \ T_{21} dx^2 \times dx^1 + \ T_{22} dx^2 \times dx^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \ T_{ij} dx^i \otimes dx^j = T_{ij} dx^i \otimes dx^j$. It is obvious how this generalises to $\mathbb{R}^n$Manifold. For a change of basis mapping from $(x^1, \dots, x^n)$ to $(u^1, \dots, u^n)$, given the appropriate Jacobian matrix, we have the $T = \tilde{T}_{kl} du^k \otimes du^l$Such that the transformed components are calculated as $\tilde{T}_{kl} = \frac{\partial x^i}{\partial u^k} \frac{\partial x^j}{\partial u^l} \ T_{ij}$. Two forms are a subset of these general Rank-2 covariant tensors because two forms have the special property that $dx^i \ \wedge \ dx^j(v, w) = \ -dx^i \ \wedge \ dx^j(w, v)$, and this

generalises such that k-form is a skew-symmetric rank k covariant tensor $\wedge^{k}(M) \subset T^*M \otimes \ldots \otimes T^*M$. If a Tensor has components with opposite sign when two indices are swapped, such as $T(v_1, \ldots, v_i, \ldots, v_j, \ldots, v_k) = -T(v_1, \ldots, v_j, \ldots, v_i, \ldots, v_k)$ this property is satisfied, then these tensors are skew-symmetric or anti-symmetric.

2. **(2, 0)-Tensors (Rank-Two Contravariant Tensor):** $T: T^*M \times T^*M \to \mathbb{R}$. It takes two one-forms and produces a real number. $T \in TM \times TM = span\{\frac{\partial}{\partial x^i} \otimes \frac{\partial}{\partial x^j} | 1 \leq i, j, \leq n\}$. Again, if M is an $\mathbb{R}^2$Manifold, then $T = T^{11}\frac{\partial}{\partial x^1} \otimes \frac{\partial}{\partial x^1} + T^{12}\frac{\partial}{\partial x^1} \otimes \frac{\partial}{\partial x^2} + T^{21}\frac{\partial}{\partial x^2} \otimes \frac{\partial}{\partial x^1} + T^{22}\frac{\partial}{\partial x^2} \otimes \frac{\partial}{\partial x^2} = \sum_{i=1}^{n}\sum_{j=1}^{n} T^{ij}\frac{\partial}{\partial x^i} \otimes \frac{\partial}{\partial x^j} = T^{ij}\frac{\partial}{\partial x^i} \otimes \frac{\partial}{\partial x^j}$. It is obvious how this generalises to $\mathbb{R}^n$Manifold. For a change of basis mapping from $(x^1, \ldots, x^n)$ to $(u^1, \ldots, u^n)$, given the appropriate Jacobian matrix, we have the $T = \tilde{T}^{kl}\frac{\partial}{\partial u^k} \otimes \frac{\partial}{\partial u^l}$Such that the transformed components are calculated as $\tilde{T}^{kl} = \frac{\partial u^k}{\partial x^i}\frac{\partial u^l}{\partial x^j}T^{ij}$.

3. **(1, 1)-Tensors (Mixed-Rank Covariant-Contravariant Tensor):** $T: T^*M \times TM \to \mathbb{R}$. It takes one vector and a one-form and produces a real number. $T \in TM \times T^*M = span\{\frac{\partial}{\partial x^i} \otimes dx^j | 1 \leq i, j, \leq n\}$. Again, if M is an $\mathbb{R}^2$Manifold, then $T = T_1^1\frac{\partial}{\partial x^1} \otimes dx^1 + T_2^1\frac{\partial}{\partial x^1} \otimes dx^2 + T_1^2\frac{\partial}{\partial x^2} \otimes dx^1 + T_2^2\frac{\partial}{\partial x^2} \otimes dx^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} T_j^i\frac{\partial}{\partial x^i} \otimes dx^j = T_j^i\frac{\partial}{\partial x^i} \otimes dx^j$. It is obvious how this generalises to $\mathbb{R}^n$Manifold. For a change of basis mapping from $(x^1, \ldots, x^n)$ to $(u^1, \ldots, u^n)$, given the appropriate Jacobian matrix, we have the $T = \tilde{T}_l^k\frac{\partial}{\partial u^k} \otimes du^l$Such that the transformed components are calculated as $\tilde{T}_l^k = \frac{\partial u^k}{\partial x^i}\frac{\partial x^j}{\partial u^l}T_j^i$.

Back to the **general rank (r, s) tensors**: $T: T^*M \times \ldots \times T^*M \times TM \times \ldots TM \to \mathbb{R}$, *which is an element of* $T \in TM \otimes \ldots \otimes TM \otimes T^*M \otimes \ldots \otimes T^*M = span\{\frac{\partial}{\partial x^{i_1}} \otimes \ldots \otimes \frac{\partial}{\partial x^{i_r}} \otimes dx^{j_1} \otimes \ldots \otimes dx^{j_s} | 1 \leq i_1, \ldots, i_r,\ j_1, \ldots, j_s\ \leq n\}$, and it is expressed in terms of its components and basis as then $T = T_{j_1,\ldots,j_s}^{i_1,\ldots,i_r}\frac{\partial}{\partial x^{i_1}} \otimes \ldots \otimes \frac{\partial}{\partial x^{i_r}} \otimes dx^{j_1} \otimes \ldots \otimes dx^{j_s}$.Again, an invertible change of basis from $(x^1, \ldots, x^n)$ to $(u^1, \ldots, u^n)$, given the appropriate Jacobian matrix, we have the $T = \tilde{T}_{l_1,\ldots,l_s}^{k_1,\ldots,k_r}\frac{\partial}{\partial u^k} \otimes \ldots \otimes \frac{\partial}{\partial u^{k_r}} \otimes du^{l_1} \otimes \ldots \otimes du^{l_s}$, such that the transformed components are calculated as $\tilde{T}_{l_1,\ldots,l_s}^{k_1,\ldots,k_r} = \frac{\partial u^{k_1}}{\partial x^{i_1}} \ldots \frac{\partial u^{k_r}}{\partial x^{i_r}}\frac{\partial x^{j_1}}{\partial u^{l_1}} \ldots \frac{\partial x^{j_s}}{\partial u^{l_s}}T_{j_1,\ldots,j_s}^{i_1,\ldots,i_r}$. Given a mapping φ : M →M, the pullback of a rank (0, t)-tensor T at the point p is defined similarly to the pullback of differential forms as: $(\varphi^*T_{\varphi(p)})_p(v_{1_p}, \ldots, v_{t_p}) = T_{\varphi(p)}(\varphi * v_{1_p}, \ldots, \varphi * v_{t_p})$. For any tensors T and S, we have $\varphi^*(T \otimes S) = \varphi^*T \otimes \varphi^*S$, which means that pullbacks distribute over tensor products, $\varphi^*(T + S) = \varphi^*T + \varphi^*S$, which means that pullbacks distribute over addition, $\varphi^*(T \wedge S) = \varphi^*T \wedge \varphi^*S$, which means that pullbacks distribute over wedge products.

The Euclidean metric is the dot product, and earlier, the Minkowski metric on $\mathbb{R}^4$ was defined. Both are (2, 0) tensors. The following definitions are needed to define a metric tensor g:

- Like smooth vector fields discussed previously, **smooth tensors** need to be infinitely differentiable in the arguments.
- **Symmetric tensor:** given tensor g and two vector fields v and w, the tensor g is symmetric if $g(v,w) \ = \ g(w,v)$.
- **Non-degenerate tensor:** Tensor g is called non-degenerate at point p, if $g_p(v_p, w_p) \ = \ 0$. g is a non-degenerate tensor if it is non-degenerate at every point p ∈ M.
- A manifold with such a tensor g is called a **pseudo-Riemannian manifold,** and the tensor g is called the metric or sometimes the **pseudo-Riemannian metric**. If the metric *g* also has one additional property, that *g(v,w)* ≥ 0 for all vector fields *v* and *w,* then it is called a **Riemannian metric,** and the manifold is called a **Riemannian manifold**.

A metric on the manifold M is a **smooth, symmetric, non-degenerate, rank-two covariant** tensor g, which we can write as a matrix. Metric tensors are generally denoted with a lowercase g. The metric tensor g gives an inner product on every vector space TpM in the tangent bundle of M. The inner product of $v_p, w_p \ \in TpM$ is given by $g(v_p, w_p)$. Most often, the inner product of two vectors is denoted with <·, ·> where $< v_p, w_p > \equiv \ g(v_p, w_p)$. For basis vectors $< \frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j} > \equiv \ g\left(\frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j}\right) = g_{ij}$, which is a (1,1)-tensor. Being (1,1)-tensor enables defining a map L from a given space to its dual, such as $\mathbb{R}^n \rightarrow (\mathbb{R}^n)^*$. The tensor metric can be expressed as a matrix:

$$g(v,w) = [v^1 \quad \dots \quad v^n] \begin{bmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nn} \end{bmatrix} \begin{bmatrix} w^1 \\ \vdots \\ w^n \end{bmatrix}$$

When $g_{ij} = \delta_{ij}$, which is the Kronecker delta, then g is the Euclidean metric, which is the Euclidean inner product, and matrix $[g_{ij}\ ]$ is none other than the identity matrix.

The length/norm of a vector v_p is calculated as: $\|v_p\| \equiv \sqrt{|g(v_p, w_p)|}$

Given two points *p* and *q* in the same coordinate patch of *M* that are connected by a curve $\gamma: [a,b] \subset \mathbb{R} \rightarrow M$ where $\gamma(a) = p$ and $\gamma(b) = q$. The curve $\gamma(t)$ has tangent velocity vectors $\dot{\gamma}(t)$ along the curve. To ensure the tangent velocity vectors actually exist at the endpoints, the curve needs to be extended a tiny amount ϵ to $(a - \epsilon, b + \epsilon) \subset \mathbb{R}$. The **length of the curve $\boldsymbol{\gamma}$** from *p* to *q* is defined to be $L(\gamma) \ = \int_a^b \sqrt{|g(\dot{\gamma}(t), \dot{\gamma}(t))|} \ dt$.

The distance between points *p* and *q* is defined in terms of the minimum piecewise continuous curve connecting them as $d(p,q) = \inf_{\gamma} L(\gamma)$, where inf is the infimum operator for the lower limit of lengths of curves in this instance.
If the Manifold is defined with a metric on it, it should be used. If it is not defined with a metric, then distances are not a valid measure.

Given a Riemannian Manifold M and a tensor metric g defined on the tangent space TpM at each point p, this tensor g is the Riemannian metric for this Manifold.

Back to the example from Physics, given an Inertia matrix, a rigid body with origin at O, time-dependent body fixed axis $K = \{\hat{x}(t), \hat{y}(t), \hat{z}(t)\}$ in $\mathbb{R}^3$, an ith particle in the rigid body has mass m_i and position vector r_i with $[r_i]_K = (x_i, y_i, z_i)$ relative to O, and let $r_i^2 \equiv g(r_i, r_i)$, then the (2,0)-moment of Inertia tensor is $I_{(2,0)} = \sum_i m_i \,(r_i^2\, g - L(r_i) \otimes L(r_i))$. The (1,1)-moment of Inertia tensor is $I_{(1,1)} = \sum_i m_i \left(r_i^2\, I - L(r_i) \otimes r_i\right)$. Some common components are defined as $I_{xx} = \sum_i m_i \left(y_i^2 + z_i^2\right)$, and $I_{xy} = -\sum_i m_i x_i y_i$ as seen from the 3x3 Inertia matrix earlier in this section. This matrix showed the entanglement caused by having a spin around one axis affecting the spin around the others. A 2-particle system would create a tensor of order 6 to maintain the $\mathbb{R}^3$ position vector of each particle. Rotation is usually in the $\mathbb{C}^3$ space. Combining rotation of 2 particle system would create a nine-dimensional Hilbert space of Complex space $\mathbb{C}^3 \otimes \mathbb{C}^3$, with the required basis. For complete derivation, check the book (Jeevanjee, 2011). More on Hilbert spaces will be presented in chapter five.

The Lie derivative applies to all forms of tensors, while the global (coordinate-free) exterior differentiation applies to differential forms only as subsets of tensors.

To introduce the Lie derivative, we need to revisit integral curves. Integral curves γ are curves on the manifold M that are considered as a family of mappings from each time step to the next in the same manifold M to itself. We begin by fixing some time t_0 as our zero time at point p. Then, for each time t, we have a mapping $\gamma(t)$ that sends $\gamma(t_0)$ to $\gamma(t_0 + t)$, such that $p = \gamma(t_0) \to \gamma_t(p) = \gamma(t_0 + t)$. It can also be between Manifolds or to another copy of M: $M \to M$, which would be a push-forward defined as: $T_p\gamma_t = \gamma_{t*}\colon T_pM \to T_{\gamma_t(p)}M$. A pull-back would be $T_p^*\gamma_t = \gamma_t\, {*}\colon T_{\gamma_t(p)}^*M \to T_p^*M$. Given that at some point p, the mapping takes us to point q, $\gamma_t(p) = q$, we can define the inverse mapping $\gamma_t^{-1}(q) = p$, which is sometimes denoted $\gamma_{-t}(q) = p$, which makes the push forwards defined as:
$T_p\gamma_t^{-1} = T_p\gamma_{-t} = \gamma_{-t*}\colon T_qM \to T_{\gamma_{-t}(q)}M$

Lie derivative of a vector field w in the direction of v at the point p: $(\mathcal{L}_v w)_p = \lim_{t\to 0} \frac{T_p\gamma_{-t} \cdot w_{\gamma_t(p)} - w_p}{t} = \frac{d}{dt}((\gamma_{-t})^* \cdot w_{\gamma_t(p)})\Big|_0$.

Lie derivatives of one-forms α, in the direction of v at point p : $(\mathcal{L}_v \alpha)_p = \lim_{t\to 0} \frac{T^*\gamma_{-t} \cdot \alpha_{\gamma_t(p)} - \alpha_p}{t} = \frac{d}{dt}((\gamma_t)^* \cdot \alpha_{\gamma_t(p)})\Big|_0$.

Lie derivative of functions as zero-forms $(\mathcal{L}_v f)_p = \lim_{t\to 0} \frac{T^*\gamma_{-t} \cdot f_{\gamma_t(p)} - f_p}{t} = \frac{\partial f}{\partial \gamma^i}\Big|_{\gamma(t_0)} \frac{\partial \gamma^i}{\partial t}\Big|_{t_0} = v_p[f]$.

Lie derivative of tensors: Given the rank (r, s) tensor T defined above and that the tensor is in a single coordinate chart on a manifold, it is possible to patch coordinate charts together. The tensor pull back by γ_t is defined as: $(\gamma_t^* T)_p(\alpha_1, \dots, \alpha_r, v_1, \dots, v_s) = T_{\gamma_{t(p)}}(\gamma_{-t}^* \alpha_1, \dots, \gamma_{-t}^* \alpha_r, \gamma_{-t}^* v_1, \dots, \gamma_{-t}^* v_s)$. The Lie derivative of a tensor is then defined as: $(\mathcal{L}_v T)_p = \lim_{t\to 0} \frac{\gamma_t^* T_{\gamma_{t(p)}} - T_p}{t} = \frac{d}{dt}(\gamma_t^* T_{\gamma_{t(p)}})\Big|_0$. This leads to several identities with other operators that facilitate the computation and provide valuable properties. For more details, refer to Appendix A in (Fortney, 2018).

Summary

This chapter should have enabled the reader to see that tensors are elementary mathematical objects that transform in a coordinate-free approach. For example, tensor fields in a vector space or on a curved manifold undergo linear transformations under changes in the space coordinates. The Jacobian matrix of the mapping functions is used to transform the coefficients in one coordinate system/basis to another. Differentiation is defined for the different object types. Going through this chapter while executing the code in the ch3.ipynb, editing it to try new examples, and checking the help of the functions for different options of the parameters, should make the material easier to visualise and manipulate for various applications.

3.5 Multilinear Subspace Learning (MSL)

As we discussed in chapter two, finding a lower-dimensional structure in a given dataset reduces computational requirements turning an intractable solution into a tractable one, reducing noise, and explaining the data dynamics or interactions. In the higher dimensions, the dimensionality curse is the main obstacle and the need for finding an approximate structure that preserves the non-linear dynamics is even more important. Figure 25 illustrates the computational requirements of the covariance matrix of a vectorised 3-

dimensional video dataset, with the first two dimensions being spatial rows and columns of 128 x 88 dimensionality and a time third dimension of 20 frames. The LSL vectorisation in (a) results in a large covariance matrix of 189 GB memory fingerprint and the resulting processing time. The MSL tensor-based analysis of three smaller covariance matrices results in 95.8KB of memory fingerprint and reduced processing time (Lu, Plataniotis and Venetsanopoulos, 2011).

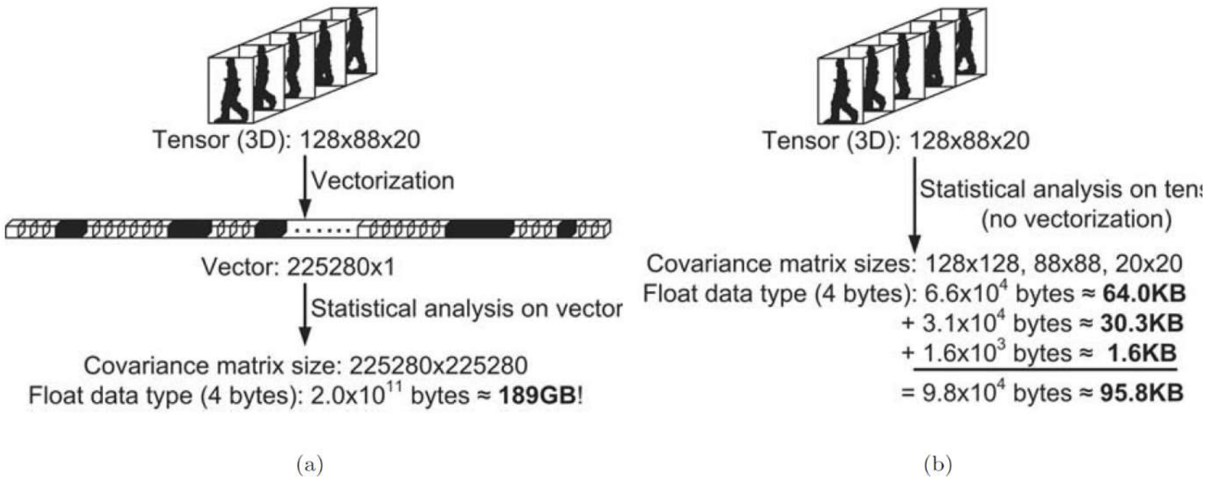

Tensor (3D): 128x88x20
Vectorization
Vector: 225280x1
Statistical analysis on vector
Covariance matrix size: 225280x225280
Float data type (4 bytes): 2.0x10^11 bytes ≈ 189GB!
(a)
Tensor (3D): 128x88x20
Statistical analysis on ten
(no vectorization)
Covariance matrix sizes: 128x128, 88x88, 20x20
Float data type (4 bytes): 6.6x10^4 bytes ≈ 64.0KB
+ 3.1x10^4 bytes ≈ 30.3KB
+ 1.6x10^3 bytes ≈ 1.6KB
= 9.8x10^4 bytes ≈ 95.8KB
(b)

Figure 25: Vector-based analysis in (a) versus tensor-based analysis (b) of a 3D video object covariance matrix (Lu, Plataniotis and Venetsanopoulos, 2011).

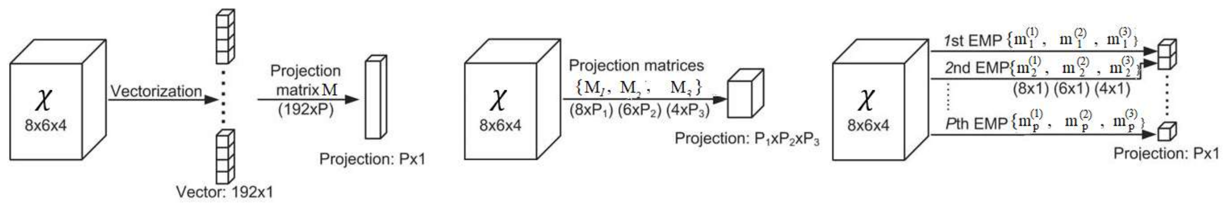
As shown in Figure 25, linear subspace learning (LSL) vectorisation is performed by the product of the number of dimensions in each mode. The multilinear subspace learning (MLS) tensor-based analysis is performed by the sum of dimensions in each mode. This reduces the degree of freedom and creates a sparse/compact grid-like structure that preserves the multi-way interactions across the different modes. Consequently, this reduces or solves the small sample size (SSS) problem containing many features, which makes LSL ill-posed (Lu, Plataniotis and Venetsanopoulos, 2014).

3.5.1 Vector-Vector Projection vs Tensor Projection Performance

In chapter one, matrices were defined as performing linear transformations on vectors. These linear transformations are considered projections because it maps a vector $\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{I_1}}$ to another vector $\boldsymbol{u} \in \mathbb{R}^{\boldsymbol{I_2}}$ that could be in a higher or lower-dimensional space using a projection Matrix M $\in \mathbb{R}^{I_1 \times I_2}$such that $u = M^T v = v \times_1 M^T$ (Lu, Plataniotis and Venetsanopoulos, 2011).

Tensor Projections are based on the Tucker decomposition (explained below) and is a generalisation of the vector projections in the higher dimensions. It takes as input an N-dimensional (N-way) tensor object $\chi\epsilon\,\mathbb{R}^{I_1 "\times" I_2 "\times" \ldots "\times" I_N}$ and project it to another dimension such as $\chi\epsilon\,\mathbb{R}^{P_1 "\times" P_2 "\times" \ldots "\times" P_N}$, where $P_j \leq I_j$ for j = 1, ..., N, using N projection matrices. For example, a matrix (2-way tensor) $\chi\epsilon\,\mathbb{R}^{I_1 "\times" I_2}$ is projected to a lower dimension $\chi\epsilon\,\mathbb{R}^{P_1 "\times" P_2}$ using two projection matrices $M_1 \in \mathbb{R}^{I_1 \times P_1}, M_2 \in \mathbb{R}^{I_2 \times P_2}$: $U = \chi \times_1 M_1^T \times_2 M_2^T = M_1^T V M_2$. In the higher order, this is generalised to $M_j \in \mathbb{R}^{I_j \times P_j}\ for\ j = 1, \ldots N, U = \chi \times_1 M_1^T \times_2 M_2^T \times_3 \ldots \times_N M_N^T$. Figure 26 shows example $\chi\epsilon\,\mathbb{R}^{8"\times" 6"\times" 4}$ vectorised in (a) and using a vector to vector projection to lower dimension, then as a tensor to tensor projection in (b), then as a tensor to vector projection in (c), where EMP stands for elementary multilinear projection.

The tensor-to-vector projections are based on the CANDECOMP/PARAFAC model (explained below). It is a special case of the tensor to tensor projections in which the lower dimension shape vector is $P_j = 1$ for j=1…N. For example, a 2-way tensor (matrix) can be projected to a vector of scalars using two projection matrices (or unit vectors since the number of the columns is 1) as $u = \chi \times_1 m_1^T \times_2 m_2^T = m_1^T \chi m_2$. Unit vectors mean their norm $\|m_j\|$ is equal to 1. Figure 27 shows a $\chi\epsilon\,\mathbb{R}^{8"\times" 6"\times" 4}$ projection to vector using EMP. In the higher dimension, it is considered the inner product between χ with the result of the outer product of the projection vectors: $U = \langle \chi, m_1^T \circ m_2^T \circ \ldots \circ m_N^T \rangle = \langle \chi, M \rangle\ for\ M = m_1^T \circ m_2^T \circ \ldots \circ m_N^T$.


χ
8x6x4
Vectorization
Vector: 192x1
Projection matrix M (192xP)
Projection: Px1
χ
8x6x4
Projection matrices {M1, M2, M3}
(8xP1) (6xP2) (4xP3)
Projection: P1xP2xP3
χ
8x6x4
1st EMP {m1(1), m1(2), m1(3)}
2nd EMP {m2(1), m2(2), m2(3)}
(8x1) (6x1) (4x1)
Pth EMP {mP(1), mP(2), mP(3)}
Projection: Px1

(a) (b) (c)

Figure 26: Illustration of (a) vector-to-vector projection, (b) tensor-to-tensor projection, (c) tensor-to-vector projection (Lu, Plataniotis and Venetsanopoulos, 2011).

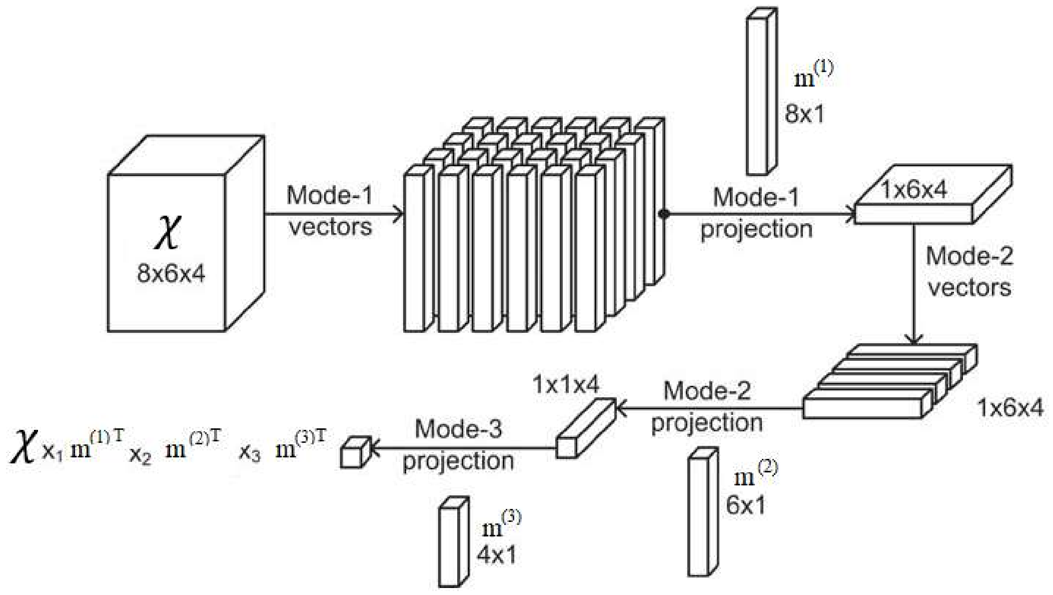


Figure 27: Illustration of an elementary multilinear projection (Lu, Plataniotis and Venetsanopoulos, 2011).

From the examples above, you can see that the vector to vector projections (VVP) vectorises a tensor to create a one-dimensional vector of elements $P = \prod_{j=1}^{N} P_j$ different elements/scalars (all elements of the tensor causing the dimensionality curse). That is then linearly mapped into the output vector using one of the linear subspace learning methods in chapter two to reduce the dimensionality by giving a weight (parameter) to estimate to every element. However, in the tensor-to-tensor projection (TTP) (and the special case of tensor-to-vector projections - TVP), each element in χ is projected by each column in the projection matrices M_j, such that there are shared columns in the projection matrices by estimating $P = \sum_{j=1}^{N} I_j$ parameters. This reduces the model's number of parameters, hence reducing its complexity. Table 0-1 compares the different projection methods on different input and output sizes and shows that tensor-to-tensor projections (TTP) require the fewest number of parameters to estimate than both tensor-to-vector (TVP) and the high dimensional vector-to-vector projections (VVP). In TTP, a tensor object A is projected to a smaller tensor of size P1 × P2 × P3. This multilinear projection can be carried out through P_N mode-n multiplications Table 2 summarises the comparison of LSL and VVP methods and MSL using TVP and TTP methods on various criteria such as representation, accuracy and complexity.

Table 0-1: Number of parameters to be estimated by three multilinear projections for N=2D, I_N = 10, Pn = 3. (Lu, Plataniotis and Venetsanopoulos, 2011)

Input	**Output**	**VVP**	**TVP**	**TTP**

$\prod_{n=1}^{N} I_N$	P	$P\prod_{n=1}^{N} I_N$	$P\sum_{n=1}^{N} I_N$	$P\sum_{n=1}^{N} P_N \times I_N$
10×10	4	400	80	40 (Pn = 2)
100×100	4	40,000	800	400 (Pn = 2)
$100 \times 100 \times 100$	8	8,000,000	2400	600 (Pn = 2)
$\prod_{n=1}^{4} 100$	16	1,600,000,000	6400	800 (Pn = 2)

Table 2: Linear versus multilinear subspace learning.

Comparison	**Linear subspace learning**	**Multilinear subspace learning**
Representation	Reshape into vectors	Natural tensorial representation
Structure	Break natural structure	Preserve natural structure
Parameter	Estimate a large number of parameters	Estimate fewer parameters
SSS problem	More severe SSS problem	Less SSS problem
Massive data	Hardly applicable to massive data	Able to handle massive data
Optimization (in most cases)	Closed-form solution	Suboptimal, iterative solution

Python notebook ch3.ipynb shows examples of the different projection methods discussed in this section and the parameters' numbers for each.

3.5.2 Scatter Matrices in the higher dimensions

As we have seen in chapter two, finding a lower-dimensional structure required pair-wise covariance or scatter matrices. A tensor object $\chi \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ is a dataset when it contains M samples, each described with N features with shape vector (I_1, I_2, … I_N). The scatter matrix in the higher-dimensional tensor objects suitable for TTP is defined per mode as follows:

$S_{T_\chi}^{(n)} = \sum_{m=1}^{M} \left(\chi_{m(n)} - \bar{\chi}_{(n)}\right)\left(\chi_{m(n)} - \bar{\chi}_{(n)}\right)^T$, where $\bar{\chi} = \frac{1}{M}\sum_{m=1}^{M} \chi_m$, and $\chi_{m(n)}, \bar{\chi}_{(n)}$ are the mode-n unfolding of χ_m and $\bar{\chi}$, respectively. For a prelabeled dataset, the between-class scatter is defined as: $S_{B_\chi}^{(n)} = \sum_{c=1}^{C} M_c \left(\bar{\chi}_{c(n)} - \bar{\chi}_{(n)}\right)\left(\bar{\chi}_{c(n)} - \bar{\chi}_{(n)}\right)^T$, where C is the number of classes, M_c is the number of samples for class c, c_m is the class label for the mth sample, $\chi_{c(n)}$ is the n-mode of χ_c samples in class c, $\bar{\chi}_{(n)}$is the n-mode of $\bar{\chi}$ the tensor means, and the class mean tensor is defined as: $\bar{\chi}_c = \frac{1}{M_c}\sum_{m=1,c_m=c}^{M} \chi_m$. The within-class scatter matrix is defined as: $S_{W_\chi}^{(n)} = \sum_{m=1}^{M} \left(\chi_{m(n)} - \bar{\chi}_{c_m(n)}\right)\left(\chi_{m(n)} - \bar{\chi}_{c_m(n)}\right)^T$, where $\bar{\chi}_{c_m(n)}$ is the mode-n unfolding of $\bar{\chi}_{c_m}$which is the mean of the class of sample m.

For TVP, scalar-based scatters are defined as degenerate equations similar to those above, using vectors instead of matrices and scalars instead of vectors.

References

Bader, B., Harshman, R.A. and Kolda, T.G. (2007) 'Temporal Analysis of Semantic Graphs Using ASALSAN', in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Omaha, NE, USA: IEEE, pp. 33–42. Available at: https://doi.org/10.1109/ICDM.2007.54.

Carter, T.A. (1995) *Linear Algebra, An Introduction to Linear Algebra for Pre-Calculus Students*. Rice University.

Chahal, J.S. (2018) *Fundamentals of Linear Algebra: With Applications in Computer Science, Economics, Engineering, Mathematics, and Physics*. Boca Raton: CRC Press, Taylor and Francis Group.

Charles Van Loan *et al.* (2009) *Future Directions in Tensor-Based Computation and Modeling*. Arlington, Virginia at the National Science Foundation,. Available at: http://www.cs.cornell.edu/cv/TenWork/Home.htm.

Cichocki, A. *et al.* (2016) 'Low-Rank Tensor Networks for Dimensionality Reduction and Large-Scale Optimization Problems: Perspectives and Challenges PART 1', *Foundations and Trends® in Machine Learning*, 9(4–5), pp. 249–429. Available at: https://doi.org/10.1561/2200000059.

Debals, O. (2017) *Tensorization and Applications in Blind Source Separation*. Available at: https://lirias.kuleuven.be/retrieve/464228 (Accessed: 27 August 2022).

Fortney, J.P. (2018) *A Visual Introduction to Differential Forms and Calculus on Manifolds*. Cham: Springer International Publishing. Available at: https://doi.org/10.1007/978-3-319-96992-3.

Haarmann, J. *et al.* (2014) 'Homotopy equivalence of finite digital images'. Available at: https://doi.org/10.1007/s10851-015-0578-8.

Harshman, R.A. (1970) 'Foundations of the PARAFAC procedure: Models and conditions for an explanatory multi-modal factor analysis', *UCLA Working Papers in Phonetics*, 16, pp. 1–84.

Hou, M. (2017) *Tensor-based Regression Models and Applications*. PhD Dissertation. Universite Laval.

Huang, F. *et al.* (2021) 'Tensor decomposition with relational constraints for predicting multiple types of microRNA-disease associations', *Briefings in Bioinformatics*, 22(3), p. bbaa140. Available at: https://doi.org/10.1093/bib/bbaa140.

Jeevanjee, N. (2011) *An introduction to tensors and group theory for physicists*. New York: Birkhäuser.

Ji, Y. *et al.* (2019) 'A Survey on Tensor Techniques and Applications in Machine Learning', *IEEE Access*, 7, pp. 162950–162990. Available at: https://doi.org/10.1109/ACCESS.2019.2949814.

Kiers, H.A.L. and Mechelen, I.V. (2001) 'Three-way component analysis: Principles and illustrative application.', *Psychological Methods*, 6(1), pp. 84–110. Available at: https://doi.org/10.1037/1082-989X.6.1.84.

Kolda, T.G. and Bader, B.W. (2009) 'Tensor Decompositions and Applications', *SIAM Review*, 51(3), pp. 455–500. Available at: https://doi.org/10.1137/07070111X.

Li, X., Zhou, H. and Li, L. (2013) 'Tucker Tensor Regression and Neuroimaging Analysis'. arXiv. Available at: https://doi.org/10.48550/arXiv.1304.5637.

Lu, H., Plataniotis, K.N. and Venetsanopoulos, A.N. (2011) 'A survey of multilinear subspace learning for tensor data', *Pattern Recognition*, 44(7), pp. 1540–1551. Available at: https://doi.org/10.1016/j.patcog.2011.01.004.

Lu, H., Plataniotis, K.N. and Venetsanopoulos, A.N. (2014) *Multilinear subspace learning: dimensionality reduction of multidimensional data*. Boca Raton, Florida: CRC Press/Taylor & Francis Group (Chapman & Hall/CRC machine learning & pattern recognition series).

Mangan, T.C. (2008) 'A Gentle Introduction to Tensors', p. 12.

Marmin, A., Castella, M. and Pesquet, J.-C. (2020) 'Globally optimizing owing to tensor decomposition', in *EUSIPCO 2020 - 28th European Signal Processing Conference*. Amsterdam, Netherlands, pp. 990–994. Available at: https://doi.org/10.23919/Eusipco47968.2020.9287511.

Miwakeichi, F. *et al.* (2004) ‘Decomposing EEG data into space–time–frequency components using Parallel Factor Analysis’, *NeuroImage*, 22(3), pp. 1035–1045. Available at: https://doi.org/10.1016/j.neuroimage.2004.03.039.

Smilde, A., Bro, R. and Geladi, P. (2004) *Multi-Way Analysis with Applications in the Chemical Sciences*. Chichester, UK: John Wiley & Sons, Ltd. Available at: https://doi.org/10.1002/0470012110.

Stegeman, A. (2007) ‘Comparing Independent Component Analysis and the Parafac model for artificial multi-subject fMRI data’, p. 38.

Vasilescu, M.A.O. and Terzopoulos, D. (2002) ‘Multilinear Analysis of Image Ensembles: TensorFaces’, in A. Heyden et al. (eds) *Computer Vision — ECCV 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 447–460. Available at: https://doi.org/10.1007/3-540-47969-4_30.

Zhang, J., Li, S.Z. and Wang, J. (2005) ‘Manifold Learning and Applications in Recognition’, in Y.-P. Tan, K.H. Yap, and L. Wang (eds) *Intelligent Multimedia Processing with Soft Computing*. Springer Berlin Heidelberg (Studies in Fuzziness and Soft Computing), pp. 281–300. Available at: https://doi.org/10.1007/3-540-32367-8_13.