Unit 01 Numerical

np.array([1, 2, 3])

creates a 1-dimensional array with elements 1, 2, and 3

np.zeros((2, 3))

creates a 2x3 array filled with zeros

np.ones((3, 3), dtype=np.float32)

creates a 3x3 array filled with ones of dtype float32

array.shape

returns the shape of an array, e.g., (3, 4)

array.ndim

returns the number of dimensions of an array, e.g., 3

np.array([1, 2, 3]).dtype

returns the datatype of elements, e.g., float64

array[:, 1]

selects the second column of a 2D array

array[1, -1]

selects the last element of the second row in a 2D array

array[array > 5]

filters elements greater than 5 in an array

np.concatenate((array1, array2), axis=0)

concatenates arrays vertically

np.stack((array1, array2), axis=1)

stacks arrays horizontally with explicit axis

np.tile(array, (2, 3))

tiles the array into a 2x3 grid

result = array1 + array2

element-wise addition of two arrays

result = (array1 >= 0) & (array2 < 10)

element-wise logical AND operation

result = np.broadcast_to(array[:, np.newaxis], (3, 4))

broadcasts array to shape (3, 4)

np.sum(array, axis=1)

sums elements along rows in a 2D array

np.mean(array, axis=0)

calculates mean along columns in a 2D array

np.max(array, axis=(0, 1))

finds maximum element across all dimensions

np.cumsum(array)

computes cumulative sum of elements along a flattened array

np.diff(array)

calculates the differences between consecutive elements in an array

np.cumprod(array)

computes cumulative product of elements along a flattened array

np.sort(array, axis=1)

sorts elements along rows of a 2D array

np.argsort(array)

returns indices that would sort an array

np.argmax(array, axis=0)

finds indices of maximum elements along columns

Array Element

A single value in an array, accessible by its indices.

Vectorization

The process of applying operations element-wise to entire arrays, avoiding explicit loops.

SIMD (Single Instruction, Multiple Data)

A parallel processing technique where a single instruction is applied to multiple data elements simultaneously.

Broadcasting

A set of rules for performing element-wise operations on arrays of different shapes during vectorized computations.

Element-wise Operation

An operation performed independently on each element of an array.

Corresponding element

In a binary elementwise operation on A and B, the element of A at a given index is combined with the element of B at the same index.

Dot Product

A mathematical operation that calculates the sum of products of corresponding elements of two arrays.

Vector

A one-dimensional array representing a list of scalar values.

Matrix

A two-dimensional array with rows and columns used for mathematical operations.

Tensor

A multi-dimensional array with more than two dimensions, essential in deep learning.

Tensor rank

The number of dimensions of a tensor.

Indexing

Accessing specific elements in an array using integer or boolean indices. Reduces rank.

Slicing

Extracting specific portions of an array to create a new array. Preserves rank.

Flattening

Converting a multi-dimensional array into a one-dimensional array.

Vectorized Function

A function optimized for array input, performing element-wise operations.

Element-wise Comparison

Comparing corresponding elements of two arrays element by element.

Data Type (dtype)

Specifies the type of elements in an array, such as int, float, or complex.

Shape

Describes the size of each dimension in an array, e.g., (3, 4) for a 2D array.

np.zeros((2, 3), dtype=np.float64)

creates a 2x3 array filled with zeros of dtype float64

np.ones((3, 3), dtype=np.int32)

creates a 3x3 array filled with ones of dtype int32

np.full((2, 2), 7.5)

creates a 2x2 array filled with 7.5

np.arange(0, 10, 2)

creates an array from 0 to 10 with a step of 2, [0, 2, 4, 6, 8]

np.linspace(0, 1, 5)

creates an array of 5 evenly spaced values between 0 and 1, [0.0, 0.25, 0.5, 0.75, 1.0]

np.eye(3)

creates a 3x3 identity matrix, where diagonal elements are 1 and others are 0

np.random.rand(2, 2)

creates a 2x2 array with random values between 0 and 1

np.empty((3, 3))

creates an uninitialized 3x3 array, often filled with garbage values

np.arange(12).reshape(3, 4)

creates a 3x4 array and reshapes it, [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

Scalar Broadcasting

Broadcasting a scalar to an array, applying the scalar to every element.

Shape Compatibility

Arrays are broadcastable if their shapes are compatible or if one of them is a scalar.

Broadcasting compatibility rule

A+B is okay if; B is scalar; A.shape==B.shape; or A.shape[:-n]==B.shape[:-n] for some n.

Broadcastable Dimensions

Dimensions are broadcastable if they are equal or one of them is 1.

Broadcastable array shape

(3, 1, 4) and (1, 4) are broadcastable, resulting in shape (3, 1, 4) after broadcasting.

Broadcastable array shape

(2, 3) and (3,) are broadcastable, resulting in shape (2, 3) after broadcasting.

Broadcastable array shape

(2, 1, 3) and (2,3,1) are broadcastable, resulting in shape (2, 3, 3) after broadcasting.

Unbroadcastable array shape

(3, 3) and (2, 4) are not broadcastable; broadcasting fails due to incompatible dimensions.

Array Slicing

Extracting a portion of an array by specifying a range of indices or dimensions. Example: array[1:3, :].

Array Indexing

Accessing specific elements in an array using integer or boolean indices. Example: array[0, 4].

Array Mapping

Applying a function to each element of an array, creating a new array. Example: np.sin(array).

Array Filtering

Selecting elements from an array based on a condition, creating a filtered array. Example: array[array > 5].

Array Reduction

Combining elements of an array to produce a single result (e.g., sum, mean). Example: np.sum(array).

Array Scans

Performing a cumulative operation on elements, building an array of partial results. Example: np.cumsum(array).

Array Searching

Finding the index or position of a specific value or condition in an array. Example: np.argmax(array).

Array Sorting

Rearranging elements in ascending or descending order within an array. Example: np.sort(array).

Arg*

Functions that return the indices of elements that satisfy a condition, like np.argmax and np.argmin.

np.transpose(array)

Swaps the dimensions of an array, effectively rotating it.

np.concatenate((array1, array2), axis=0)

Joins arrays along an existing axis, creating a new array.

np.max(array, axis=None)

Returns the maximum value along the specified axis or the flattened array.

np.min(array, axis=None)

Returns the minimum value along the specified axis or the flattened array.

np.mean(array, axis=None)

Computes the arithmetic mean along the specified axis or the flattened array.

np.sum(array, axis=None)

Calculates the sum of array elements along the specified axis or the flattened array.

np.std(array, axis=None)

Computes the standard deviation along the specified axis or the flattened array.

np.argmin(array, axis=None)

Returns the indices of the minimum values along the specified axis or flattened array.

np.argmax(array, axis=None)

Returns the indices of the maximum values along the specified axis or flattened array.

np.argsort(array, axis=-1)

Returns the indices that would sort the array along the specified axis.

np.where(condition, [x, y])

Returns elements chosen from x or y depending on the condition.

np.unique(array)

Finds the unique elements of an array and returns them in sorted order.

np.clip(array, a_min, a_max)

Clips the values in an array to be within a specified range.

np.exp(array)

Computes the exponential of each element in the input array.

np.log(array)

Computes the natural logarithm of each element in the input array.

np.sqrt(array)

Computes the square root of each element in the input array.

np.sin(array), np.cos(array), np.tan(array)

Computes trigonometric functions element-wise.

np.logical_and(array1, array2)

Computes the element-wise logical AND of two arrays.

np.logical_or(array1, array2)

Computes the element-wise logical OR of two arrays.

np.logical_not(array)

Computes the element-wise logical NOT of the input array.

np.isnan(array)

Detects NaN (Not a Number) elements in an array.

np.nonzero(array)

Returns the indices of the elements that are non-zero (or True).

np.unravel_index(indices, shape)

Converts a flat index or array of flat indices into a tuple of coordinate arrays, e.g. from np.argmax.

np.isinf(array)

Detects infinite elements in an array.

np.floor(array)

Rounds down the elements of an array to the nearest integer.

np.ceil(array)

Rounds up the elements of an array to the nearest integer.

np.random.randint(low, high, size=None)

Generates random integers from low (inclusive) to high (exclusive).

np.random.normal(mean, std_dev, size=None)

Generates random samples from a normal distribution with specified mean and standard deviation.

np.random.choice(array, size=None, replace=True)

Generates a random sample from a given 1-D array.

np.random.DefaultRNG(seed=None)

Creates a random number generator with a specified seed.

rng.uniform(low=0.0, high=1.0, size=None)

Generates random numbers from a uniform distribution from a pre-created generator.

Unit 02 Floats And Strided

sign bit

the leftmost bit in a floating-point number, indicating its positive (0) or negative (1) value

mantissa (or significand)

the bits in a floating-point number representing the precise fractional part, excluding the implicit leading 1 bit

exponent

the bits in a floating-point number determining the scale of the number (e.g., 2^exponent)

exponent bias

an offset value applied to the exponent to allow for representation of both positive and negative exponents in a biased manner

strided array

a way of laying out data in memory where the elements are stored in a flat array, but the strides allow for (apparent) multidimensional indexing.

strides in NumPy

describe the number of bytes to jump to get to the next element in each dimension of an array

C ordering (row-major)

arrays are stored in memory "row by row", with the rightmost index changing fastest

Fortran ordering (column-major)

arrays are stored in memory "column by column", with the leftmost index changing fastest

dope vector

contains information about an array's shape, strides, data type, and other metadata

advantages of C ordering

efficient for row-wise traversal and matches how most programming languages access arrays

advantages of Fortran ordering

efficient for column-wise traversal, suits mathematical and engineering computations

dope vector for a 3D array

example: shape=(3, 4, 5), strides=(20, 5, 1), indicating how to move in memory for each dimension

effect of changing strides on array iteration

changing strides can modify the order in which array elements are accessed in memory

Fortran ordering in nested arrays

innermost arrays follow Fortran ordering, beneficial for numerical simulations requiring column-wise computations

Illiffe vector

alternative to dope vectors, representing shape and strides as nested lists

IEEE 754 exceptions

special conditions that occur during floating-point arithmetic operations

overflow

occurs when a result is too large to be represented within the finite range of a floating-point format

underflow

occurs when a result is too small to be represented as a normalized floating-point number

division by zero

occurs when attempting to divide a number by zero, resulting in an undefined mathematical operation

invalid operation

occurs for operations like square root of a negative number, generating a NaN (Not a Number) result

trapping exceptions

IEEE 754 allows exceptions to be trapped, enabling custom handling of exceptional conditions

setting traps in IEEE 754

specifies whether exceptions like overflow, underflow, or invalid operation should trigger an interrupt or be ignored

handling NaN (Not a Number)

NaN is a special value representing undefined or unrepresentable results, propagating through most operations

handling Inf (Infinity)

Inf represents positive or negative infinity, occurring as a result of overflow or division by zero

detecting and avoiding NaN

isnan() function checks if a value is NaN, helping identify and handle undefined computations

using Inf in computations

Inf can be used in mathematical operations, providing a placeholder for extremely large or small values

NaN (Not a Number)

a special floating-point value representing undefined or unrepresentable results

propagation of NaN

any arithmetic operation with NaN as an operand results in NaN, making it propagate through calculations

NaN in comparisons

any comparison involving NaN, including equality checks, results in false, ensuring consistency in comparisons

isnan() function

checks if a value is NaN, returning True if it's NaN and False otherwise

NaN as a placeholder

NaN can be used to indicate missing or undefined data in computations and data analysis

handling NaN in data cleaning

NaN values in datasets can be replaced, removed, or interpolated to prepare data for analysis

IEEE 754 standard for NaN

defines the bit pattern used to represent NaN, ensuring consistent handling across different systems

NaN in mathematical functions

many functions return NaN for undefined inputs, allowing continued execution of programs

NaN in statistical calculations

NaN values in datasets can be skipped during statistical calculations to obtain meaningful results

NaN (Not a Number) bit pattern

In 64-bit float: [sign][all one exponent] [non-zero mantissa]

Inf (Infinity) bit pattern

In 64-bit float: [sign][all one exponent] [all zero mantissa]

Positive zero bit pattern

In 64-bit float: [0][all zeros]

Negative zero bit pattern

In 64-bit float: [1][all-zeros]

Signed zero issue

Arises due to the distinction between positive and negative zero, impacting certain calculations

Reshape for Promotion

Use np.reshape() to change the shape of an array, promoting or demoting dimensions as needed.

Squeeze for Reduction

Use np.squeeze() to remove (all) single-dimensional entries from the shape of an array, reducing its dimensions. E.g. a (3,1,4,1) array becomes a (3,4) array.

Newaxis for Promotion

Use np.newaxis or None to add new dimensions, promoting arrays to higher dimensions. x[:, np.newaxis] promotes x from 1D to 2D.

None for Promotion

Using None in slicing/indexing adds a new axis, promoting arrays to higher dimensions. x[:, None] promotes x from 1D to 2D.

Ravel for Flattening

Use np.ravel() to flatten multi-dimensional arrays into one-dimensional arrays.

Flatten for Flattening

Use np.ndarray.flatten() to flatten arrays into one-dimensional arrays (returns a copy).

Einsum Function

np.einsum() allows for efficient contraction and summation of multidimensional arrays using Einstein's summation convention.

Einsum Notation

Uses subscripts to define input and output array dimensions, facilitating complex array operations in a concise manner.

Einsum Example

Example: np.einsum('ijk->jik', A, B) swaps the first two axes of A and B and leaves the last axis unchanged.

Swapaxes Function

np.swapaxes() swaps two axes of an array, changing the shape and layout of the array in memory.

Swapaxes Example

Example: np.swapaxes(arr, axis1, axis2) swaps the specified axes (axis1 and axis2) of the input array.

np.meshgrid()

grids=np.meshgrid(*spacings) creates coordinate matrices from coordinate vectors, facilitating vectorized evaluations of functions on a grid.

Benefits of Einsum

Enables efficient handling of complex array operations, reducing the need for explicit loops and intermediate arrays.

Machine Epsilon

Machine epsilon (eps) is the smallest positive floating-point number such that 1.0 + eps != 1.0.

Double Precision Machine Epsilon

For IEEE 754 double precision (float64), machine epsilon is approximately 2.220446049250313e-16.

Single Precision Machine Epsilon

For IEEE 754 single precision (float32), machine epsilon is approximately 1.19209290e-07.

IEEE 754 Roundoff Guarantees

In IEEE 754 arithmetic, roundoff errors are guaranteed to be less than 0.5 ulps (units in the last place) for basic operations.

Roundoff Guarantee Example

For addition, the result is correctly rounded to the nearest representable floating-point number within 0.5 ulps of the exact result.

IEEE 754 Rounding Modes

IEEE 754 defines different rounding modes: round to nearest, round towards positive infinity, round towards negative infinity, and round towards zero.

ULPs (Units in the Last Place)

ULPs represent the gap between adjacent floating-point numbers in a numerical system.

Significance of ULPs

ULPs indicate the precision of a floating-point number relative to its magnitude.

Floating-Point Numbers and ULPs

Two floating-point numbers that differ by 1 ULP have the smallest possible difference.

np.nextafter Function

In NumPy, np.nextafter(x, y) returns the next representable floating-point number after x in the direction of y.

np.nextafter Example

Example: np.nextafter(1.0, 2.0) returns the next representable floating-point number greater than 1.0.

np.nextafter Usage

Useful for exploring floating-point number representation and analyzing precision in numerical algorithms.

Avoiding Catastrophic Cancellation

By understanding ULPs, programmers can mitigate the effects of catastrophic cancellation in numerical computations.

Comparing Floating-Point Numbers

When comparing floating-point numbers, consider a tolerance in ULPs rather than an absolute tolerance to account for precision differences.

Time complexity of transpose

O(1)

you are

a space cadet

Square Root of a Negative Number

Taking the square root of a negative number in real arithmetic results in NaN.

Zero Divided by Zero

Zero divided by zero is an undefined operation, resulting in NaN.

Infinity Minus Infinity

Performing subtraction between positive and negative infinity results in NaN.

NaN as an Operand

Any arithmetic operation involving NaN as an operand results in NaN.

0.0 * Infinity

Multiplying zero by positive or negative infinity results in NaN.

Infinity / Infinity

Dividing positive or negative infinity by itself results in NaN.

Infinity * Zero

Multiplying positive or negative infinity by zero results in NaN.

Overflow Exception

If untrapped, overflow results in positive or negative infinity (Inf).

Underflow Exception

If untrapped, underflow results in positive/negative zero.

Division by Zero Exception

If untrapped, division by zero results in positive or negative infinity (Inf).

Invalid Operation Exception

If untrapped, invalid operations like square root of a negative number result in NaN.

Inexact Exception

A result that is not exact, typically due to limited precision or rounding, but not exceptional. Results in nearest float.

Unit 03 Visualisation

facet

a way to break down data into subsets and display them separately within the same plot

layer

a distinct component of a plot, such as data points, lines, or shapes, added on top of the base plot

geom

the geometric object representing data points, lines, or shapes in a plot

mapping

the process of associating variables in the dataset to visual properties in the plot, like x and y coordinates

stat

a statistical transformation applied to the data before plotting, such as mean, median, or count

scale

a mapping between data values and aesthetic properties, like color or size

coord

the coordinate system defining the x and y axes and their scales

guide

elements like legends, labels, and color scales that help interpret the plot

Layered Grammar of Graphics

a framework for visualizing data, comprising a set of components that can be combined in layers to create a plot

components of the Layered Grammar of Graphics

data, mapping, stat, coord, geom, scale, guide, facet, layer

Florence Nightingale

a pioneer in data visualization, known for her coxcomb diagram representing causes of mortality in the Crimean War, emphasizing the significance of sanitation in healthcare

William Playfair

credited with inventing statistical graphics, introduced line charts, bar charts, and pie charts, making complex data accessible and understandable to a wider audience

axes

coordinate axes representing data values, comprising x-axis and y-axis, defining the scales and ranges for data display

title

a text element providing a descriptive title for the plot, giving context to the data being visualized

legend

a guide that explains the meaning of colors, shapes, or other visual properties used in the plot, corresponding to different data categories

labels

textual annotations attached to axes, data points, or other plot elements, providing information about what is being represented

facet labels

labels specific to subsets of data in faceted plots, providing context for each subset within the plot

guides

collectively refer to legends, labels, and color scales, aiding in the interpretation of visual properties and mapped data values

theme

a predefined set of visual properties, such as colors, fonts, and background, defining the overall appearance of the plot

shape

the visual appearance of point geoms, defining how individual data points are represented, such as circles, squares, or triangles

colour

the hue or intensity of color used to distinguish between different data points or categories in the plot

size

the physical dimension of point geoms, determining their area or diameter in the plot, often representing a quantitative variable

alpha

the transparency or opacity of geometry, allowing overlapping geoms to be visually distinguished

linetype

the pattern used to represent lines, differentiating between dashed, solid, and dotted lines

colour scales

mappings from data values to colors, aiding in the visualization of information in plots

perceptually uniform monotonic brightness scales

color scales where equal steps in data values result in equal perceptual differences in brightness, ensuring accurate representation

examples of good color scales for unsigned data

viridis, plasma, inferno, magma, cividis

rainbow color scales

vibrant, colourful but perceptually non-uniform color scales, where equal steps in data values do not result in equal perceptual differences in color, leading to inaccurate representation

perceptual linearity

the property of a color scale where equal steps in data values result in equal perceptual differences in color, ensuring accurate representation

monotonic colour scale

a color scale where the perceived brightness consistently increases or decreases with the data values

unsigned colour scale

a color scale representing a unipolar range of values, where all data points are compared to a single reference point, often used for non-negative values

diverging colour scale

a color scale representing data values relative to a central reference point, with distinct colors for positive and negative values, aiding in highlighting both high and low extremes in the data

scatter plot

a type of plot where individual data points are represented as dots on a two-dimensional plane, useful for visualizing the relationship between two continuous variables

line plot

a plot where data points are connected by straight lines, emphasizing the trend or pattern in the data over a continuous range, suitable for time-series or sequential data where data lies on a true continuum

staircase plot

a type of plot where data points are connected by horizontal and vertical lines, creating a step-like appearance, often used to represent discrete or categorical data

bar chart

a graphical representation of categorical data using rectangular bars of varying heights or lengths, with the length of each bar proportional to the value it represents, suitable for comparing discrete categories

Box plot

a statistical plot that displays the distribution of a dataset, indicating the first quartile, median, third quartile, as well as a a "whisker" range, providing insights into the data's spread and identifying potential outliers

polar coord

a coordinate system where data points are plotted on a circular grid, often used to display cyclical or periodic data

log-log coord

a coordinate system where both x-axis and y-axis have logarithmic scales, useful for visualizing data with exponential growth or decay

semilog coord

a coordinate system where only one axis (either x or y) has a logarithmic scale, suitable for data with a wide range of magnitudes

symlog coord

a coordinate system where linear scaling is used for small values, and logarithmic scaling is used for absolute large values, accommodating data with both small and large magnitude over both positive and negative values

aspect ratio

the ratio of the plot's width to its height, influencing the visual perception of the data, where equal aspect ratios maintain proportionality in the plot elements

figure size

the dimensions (width and height) of the entire plotting area, determining the overall size of the plot and its readability, important for adjusting the plot to fit within documents or presentations

histogram

a graphical representation of the distribution of a dataset, showing the frequency of different ranges of values

violin plot

a density plot, providing a more detailed view of the data's distribution, especially in cases of skewness and multimodality

dot plot

a simple chart where each data point is represented by a dot along a single axis, useful for comparing individual values in a dataset

means

the arithmetic average of a set of values, calculated by adding up all values and dividing by the total number of values

medians

the middle value in a sorted dataset, separating the higher half from the lower half of the data points

error bars

graphical representations of the variability or uncertainty of data points in a plot, typically indicating the standard deviation or confidence intervals

confidence intervals

a range of values that is likely to contain an unknown population parameter, providing a measure of the uncertainty associated with a sample estimate. Often used as error bars (e.g. 95% confidence interval)

standard deviation

a measure of the dispersion or spread of a set of values, indicating how much the values deviate from the mean

standard error

an estimate of the standard deviation of a sample population, indicating the variability of the sample mean. Equal to the standard deviation divided by the square root of the sample size

uncertainty representation

the visual depiction of the degree of confidence or reliability associated with data points or statistical estimates in a plot

error bars

horizontal or vertical lines extending from data points or bars, indicating the range within which the true value is likely to lie, providing a measure of uncertainty

confidence intervals

shaded areas or lines around a plot element, representing the range within which a certain percentage of data points or estimates are expected to fall, indicating the precision of the estimate

uncertainty bands

shaded regions surrounding a line or curve in a plot, indicating the variability or uncertainty in the data, commonly used in time series or regression plots

whiskers in Box plots

lines extending from the box in a Box plot, showing the range within which most of the data points lie, with outliers displayed as individual points, illustrating the spread and potential outliers in the dataset

y versus x mappings

the directionality of the relationship between variables in a plot, where the y-axis represents the dependent variable and the x-axis represents the independent variable, indicating the direction of causality

causality in plots

x-axis indicates the cause (independent variable) and y-axis indicates the effect (dependent variable)

chartjunk

unnecessary or distracting visual elements in a plot, such as heavy gridlines, excessive tick marks, or decorative images, that do not add to the message of the plot

data-to-ink ratio

the proportion of ink used to display data in a plot, where the ink used to display data should be maximized, and the ink used for non-data elements should be minimized

Unit 04 Vectors

vectors in R^N

ordered tuples of N real numbers representing points in N-dimensional space

vector addition (v + w)

component-wise addition of corresponding elements of vectors v and w

scalar multiplication (c * v)

multiplication of each element of vector v by scalar constant c

norm

length of a vector -- a defined property

fundamental vector operations

addition, scalar multiplication

topological vector space

a vector space equipped with a norm (distances exist)

inner product vector space

a vector space equipped with an inner product (angles exist)

distance

norm of a difference of two vectors norm(A-B) or norm(A + (-1 * B))

inner product (dot product)

sum of the products of corresponding elements of v and w

real number

a number that can be represented by a point on the number line

$\mathbb{R}^N$

the set of all N-tuples of real numbers

$\mathbb{R}^+$

the set of non-negative real numbers.

$(\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}^+$

a function that takes two vectors of dimension n and returns a non-negative real number (like the norm)

dot product

the inner product for a real vector space

angle between vectors

angle θ between vectors v and w is given by cos(θ) = (v · w) / (||v|| ||w||)

cosine similarity

the cosine of the angle between two vectors, i.e. the dot product of the vectors divided by the product of their norms (not the same as the angle!)

orthogonal vectors

vectors that are perpendicular to each other, i.e. their dot product is zero, at 90 degrees to each other

unit vectors

vectors with norm 1 (in some specified norm)

orthonormal vectors

some vectors $a,b,c,...$ that are (mutually) orthogonal and are all unit vectors

linear interpolation of two vectors

z = αx + (1 - α)y combines vectors x and y by taking a weighted average of their elements

L1 norm of vector v

sum of the absolute values of its elements: ||v||_1 = |v_1| + |v_2| + ... + |v_N|

L2 norm of vector v

square root of the sum of the squares of its elements:

$$||v||_2 = \sqrt{(v_1^2 + v_2^2 + \ldots + v_N^2)}$$

Lp norm of vector v

generalization of L2 norm:

$$||v||_p = (|v_1|^p + |v_2|^p + \ldots + |v_N|^p)^{1/p}$$

L_infinity norm of vector v

maximum absolute value of its elements: ||v||_∞ = max(|v_1|, |v_2|, ..., |v_N|)

L_-infinity norm

minimum absolute value of its elements: ||v||_-∞ = min(|v_1|, |v_2|, ..., |v_N|)

normalisation

process of scaling a vector by dividing by its norm

unit vector

vector with norm 1 (in some specified norm)

mean vector

average of a set of vectors, corresponding to the geometric centroid of the vectors

computation of the mean vector

sum the vectors and divide by the number of vectors (i.e. multiply by 1/N)

lerp

linear interpolation, i.e. weighted average of two vectors

Curse of Dimensionality

phenomenon where high-dimensional spaces exhibit counterintuitive behaviors, especially increasing sparsity

Density in shells

in high dimensions, most of the volume of a hypersphere is concentrated near its surface

Volume of unit ball in high-d space

decreases exponentially with dimension

Edge effects in high-d boxes

almost all points in a high-d box are very close to the edges or corners

Near-orthogonality of all vectors

in (very) high-dimensional spaces, most vectors are nearly orthogonal to each other (at 90 degrees)

Exponential increase in distance

as dimensionality increases, points become exponentially farther apart

Increased sparsity of data

in high-dimensional spaces, data points become sparse, making analysis and visualization challenging

Difficulty in visualizing and interpreting high-d data

high-dimensional data is difficult to represent visually and comprehend intuitively

All the bread is crust

in high-dimensional spaces, most of the volume inside some convex object (like a sphere or a box) is concentrated near its surface

The average is far away

in high-dimensional spaces, the average of a set of points is far away from most of the points

Distances in high-d spaces

in high-dimensional spaces, norms become less useful for measuring distances between points except locally, as almost all points are the same distance from each other

Matrix Addition (A + B)

sum of corresponding elements of matrices A and B

Scalar Multiplication (c * A)

multiplication of each element of matrix A by scalar constant c

Matrix Multiplication (AB)

operation resulting from composing linear maps represented by matrices A and B

Rules for matrix multiplication

the number of columns of A must equal the number of rows of B,

$$R^{MxN}, R^{NxP} -> R^{MxP}$$

Matrix Composition and Function Composition

matrix multiplication corresponds to composition of linear functions

Matrix Dimensions (NxM)

notation where N represents the number of rows and M represents the number of columns in a matrix

Matrix-Vector Multiplication (Av)

result of applying linear transformation represented by matrix A to vector v

Transpose of a Matrix (A^T)

operation flipping rows and columns of matrix A

(AB)^T = B^T A^T

transposition property of the product of matrices

Linear map

any function between two vector spaces that satisfies the requirements of linearity

Linear transform

any linear map from R^N to R^N (i.e. where dimension is unchanged)

Linear map properties

linearity, preservation of origin, lines, and planes

Linear projection

any linear map from R^N to R^M where M < N (i.e. where dimension is reduced)

Properties of Linear Maps

linearity, preservation of origin, lines, and planes are fundamental properties of linear transformations

Combining Linear Maps

all matrix operations (addition, multiplication, etc.) can be seen as combinations of rotations, scales, and translations

Definition of Linearity

linear maps satisfy the properties of additivity and scalar multiplication

Additivity Property of Linear Maps

for any vectors v and w, T(v + w) = T(v) + T(w)

Scalar Multiplication Property of Linear Maps

for any scalar c and vector v, T(cv) = cT(v)

Preservation of Origin

linear maps always map the origin to the origin: T(0) = 0

Preservation of Lines

linear maps transform lines in the input space to lines in the output space

Rotation matrix

matrix representing a rotation in space, i.e. a linear map that rotates vectors by a specified angle without scaling

Scale matrix

matrix representing a scaling in space, i.e. a linear map that scales vectors by a specified factor without rotating; always diagonal

Skew matrix

matrix representing a skew in space, i.e. a linear map that shears vectors

Definition of Matrices in R^(MxN)

matrices are rectangular arrays of real numbers with M rows and N columns

Matrix Multiplication (AB)

operation resulting in a new matrix C, where C[i, j] is the dot product of the ith row of A and jth column of B

Associative Property of Matrix Multiplication

(AB)C = A(BC), allowing the grouping of matrix multiplication operations

Non-Commutative Property of Matrix Multiplication

in general, AB ≠ BA for matrices A and B

Identity Matrix (I)

a square matrix where all elements on the main diagonal are 1 and all other elements are 0; AI = A and IA = A. Has no effect on a vector when multiplied by it

Effect of Multiplying by Identity Matrix

any matrix multiplied by the identity matrix remains unchanged

Zero Matrix (0)

a matrix where all elements are 0; A0 = 0 for any matrix A, maps all vectors to the origin

Matrix Multiplication by Zero Matrix

the result is always the zero matrix: A0 = 0

Diagonal of a Matrix

the elements from the top left to the bottom right of a square matrix, elements A[i,i], (the "main diagonal")

Anti-diagonal of a Matrix

the elements from the top right to the bottom left of a square matrix, elements A[i, N-i+1]

Diagonal Matrix

a square matrix where all non-diagonal elements are zero, a pure scaling operation

Off-diagonal

all elements of a matrix that are not on the main diagonal

Upper Triangular Matrix

a square matrix where all elements below the main diagonal are zero

Lower Triangular Matrix

a square matrix where all elements above the main diagonal are zero

Square Matrix

a matrix with the same number of rows and columns (NxN)

Properties of square matrix

may have an inverse; has an eigenvalue decomposition; has a determinant

Symmetric Matrix

a square matrix that is equal to its transpose: A^T = A. The diagonal elements are symmetric about the main diagonal

Skew-Symmetric (Anti-symmetric) Matrix

a square matrix where the transpose equals the negative of the matrix: A^T = -A

Orthogonal Matrix

a square matrix where the rows and columns are orthonormal unit vectors

Identity Matrix (I)

a diagonal matrix where all diagonal elements are 1

Zero Matrix (0)

a matrix where all elements are 0

Sparse Matrix

a matrix in which most elements are zero

Word Embeddings

vectors used to represent words as numerical vectors in natural language processing tasks

Covariance Matrix

The covariance matrix is a square matrix that summarizes the relationships between multiple variables. For a set of n variables, the covariance matrix C is an n x n matrix, where each entry C[i, j] represents the covariance between variables i and j.

Computing the covariance matrix

The covariance matrix C can be computed using the formula C = (X - μ)(X - μ)^T / N, where X is the data matrix, μ is the mean vector, and N is the number of data points.

Covariance Definition

Covariance between two variables X and Y measures how their values change together. Positive covariance indicates that X and Y increase or decrease together, while negative covariance means they move in opposite directions. Covariance of X and Y is calculated as Cov(X, Y) = Σ[(X_i - μ_X)(Y_i - μ_Y)] / N, where X_i and Y_i are individual data points, μ_X and μ_Y are means of X and Y, and N is the number of data points.

Diagonal Entries of Covariance Matrix

The diagonal entries C[i, i] represent the variance of variable i. Variance measures how much a variable deviates from its mean. A larger diagonal value indicates higher variability in the corresponding variable.

Off-Diagonal Entries of Covariance Matrix

Off-diagonal entries C[i, j] (i ≠ j) represent the covariance between variables i and j. Positive values indicate a positive relationship, while negative values indicate a negative relationship. The magnitude represents the strength of the relationship.

Symmetry of Covariance Matrix

The covariance matrix is symmetric (C[i, j] = C[j, i] for all i and j) because the covariance between variables i and j is the same as the covariance between j and i.

Unit 05 Linear

sparse matrix

a matrix in which most of the elements are zero

dense matrix

a matrix in which most of the elements are non-zero

symmetric matrix

a square matrix that is equal to its transpose (i.e. A = A^T)

representing edge-weighted graphs as matrices

a common method is using an adjacency matrix where each entry represents the weight of the edge between two vertices

spectral methods

representing graphs as matrices

discrete-continuous interchange

the idea that discrete objects can be represented as continuous objects, and vice versa

flow in a graph

it can be represented as a product of the adjacency matrix and a vector representing flow values, giving a new vector representing updated flow values

sink

a vertex with more outgoing edge weight than incoming edge weight

source

a vertex with more incoming edge weight than outgoing edge weight

degree

the count of the edges connected to a vertex

in-degree

the count of the edges entering a vertex

out-degree

the count of the edges leaving a vertex

adjacency matrix

a square matrix where each entry represents the weight of the edge between two vertices, usually sparse.

directed graph

a graph where edges have a direction

undirected graph

a graph where edges have no direction

flow conservation

the idea that the total flow into a vertex must equal the total flow out of a vertex

stochastic matrices

matrices where each row represents a probability distribution, meaning all elements are non-negative and each row sums to 1

row-stochastic

a matrix where each row represents a probability distribution, meaning all elements are non-negative and each row sums to 1

column-stochastic

a matrix where each column represents a probability distribution, meaning all elements are non-negative and each column sums to 1

doubly stochastic matrices

matrices that are both row-stochastic and column-stochastic, where both rows and columns sum to 1. Conserving in and out flow.

matrix exponentiation

raising a matrix to a power like A^5 = A @ A @ A @ A @ A

degree matrix

a diagonal matrix where each entry represents the degree of a vertex `D = diag(sum(A, axis=0))`

Laplacian matrix

a matrix used in spectral graph theory, defined as the degree matrix minus the adjacency matrix `L = D - A`

eigenvalues of the Laplacian matrix

number of zeros in the eigenvalues of the Laplacian matrix is equal to the number of connected components in the graph

eigenvectors of the Laplacian matrix

the eigenvectors of the Laplacian matrix are used to partition the graph into subgraphs

Fiedler value

the second smallest eigenvalue of the Laplacian matrix, indicates how well a graph can be partitioned into two subgraphs

Fiedler vector

the eigenvector corresponding to the Fiedler value, indicates the optimal partitioning of a graph into two subgraphs

integer matrix powers as repeated linear transformations

raising a matrix to an integer power means applying the linear transformation represented by the matrix multiple times

matrix inversion

multiplying a vector by the inverse of a matrix undoes the transformation

Number of eigenvalues

A square matrix of size n has n eigenvalues (though some may be complex).

Number of eigenvectors

A square matrix of size n has n eigenvectors. However, if the matrix is singular, they may not be distinct.

Eigenvalues of a matrix

Eigenvalues are the solutions to the characteristic equation $\det(A - \lambda I) = 0$, where A is the (square) matrix, λ is the eigenvalue, and I is the identity matrix.

Eigenvectors of a matrix

Eigenvectors corresponding to an eigenvalue λ are non-zero vectors v that satisfy the equation $Av = \lambda v$, where A is the (square) matrix and v is the eigenvector.

Interpretation of an eigenvector

Eigenvectors represent the directions along which a linear transformation acts by scaling alone.

Interpretation of an eigenvalue

Eigenvalues represent the scaling factor by which a linear transformation acts on the corresponding eigenvector.

Trace of a matrix

Trace of a square matrix A, denoted as tr(A), is the sum of its diagonal elements *and* the sum of its eigenvalues.

Determinant

The determinant of a matrix A, det(A), is equal to the product of its eigenvalues. $\det(A) = \lambda_1 * \lambda_2 * \ldots * \lambda_n$, where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A.

Geometric interpretation of the determinant

The determinant of a matrix A is equal to the area of the parallelogram spanned by its column vectors. Alternatively, how much space is "squished/expanded" by the transformation represented by A.

Singular

A matrix is singular if $\det(A)=0$; it collapses a dimension and thus cannot be inverted.

Non-singular

A matrix that *can* be inverted, with $\det(A)!=0$. Every dimension is preserved.

Power iteration method

Power iteration is an iterative algorithm used to find the leading eigenvalue and corresponding eigenvector of a matrix. It involves repeated multiplication of the matrix with a vector and normalization to converge towards the dominant eigenpair.

Eigenpair

An eigenpair is a pair of an eigenvalue and its corresponding eigenvector.

Eigendecomposition

Eigendecomposition is the factorization of a matrix into a set of eigenvectors and eigenvalues.

Geometric interpretation of eigenvalues

Eigenvalues represent the scaling factor by which a matrix transforms an eigenvector. If $\lambda > 1$, the eigenvalue stretches the vector; if $0 < \lambda < 1$, it compresses the vector; if $\lambda < 0$, it reflects the vector.

Geometric interpretation of singularity

A matrix is singular if and only if its determinant is zero. Geometrically, this means the matrix collapses the space along at least one dimension, resulting in a flat or lower-dimensional transformation.

Real eigenvalues

Symmetric square matrices always have real eigenvalues and orthogonal eigenvectors.

Complex eigenvalues

Complex eigenvalues in a matrix indicate oscillatory behavior in dynamic systems (e.g. in a flow on a graph). When eigenvalues have imaginary parts, the system exhibits periodic or oscillatory motion.

Relation between trace and eigenvalues

The sum of eigenvalues of a matrix A is equal to its trace: λ1 + λ2 + ... + λn = tr(A), where λ1, λ2, ..., λn are the eigenvalues of A.

Relation between determinant and eigenvalues

The determinant of a matrix A is equal to the product of its eigenvalues: det(A) = λ1 * λ2 * ... * λn, where λ1, λ2, ..., λn are the eigenvalues of A.

Principal Component Analysis (PCA)

PCA is a technique used for dimensionality reduction and data compression. It identifies the principal components (eigenvectors) of a the covariance matrix of a dataset, reducing its dimensions while retaining most of the original variance.

Eigenvalue Decomposition in PCA

PCA involves computing the eigenvalue decomposition of the covariance matrix of the data.

Eigenspectrum

The eigenspectrum represents the distribution of eigenvalues; the eigenvalues ranked by absolute magnitude. Large eigenvalues indicate significant variance in the data, while small eigenvalues represent noise or less important features.

Principal Components

Principal components are the eigenvectors of the covariance matrix. They form a new orthogonal basis for the data, with the first principal component capturing the most variance, the second capturing the second most, and so on.

Variance explained by Principal Components

Each principal component explains a certain percentage of the total variance in the data. The ratio of the eigenvalue of a principal component to the sum of all eigenvalues indicates the proportion of variance it represents.

Dimensionality reduction

The process of finding a low-dimensional representation of a dataset that preserves most of its important features. It is useful for visualisation and speeding up machine learning algorithms.

Dimensionality Reduction using PCA

PCA reduces the dimensionality of data by projecting it onto a lower-dimensional subspace formed by the top-k principal components. This reduces noise and computational complexity while

Limitations of PCA

PCA assumes linearity and orthogonality of principal components, making it sensitive to outliers and nonlinear relationships. It may not perform well in cases where these assumptions are violated, requiring alternative methods like nonlinear PCA.

Choosing the Number of Principal Components

Determining the appropriate number of principal components (k) involves analyzing the cumulative explained variance. A common approach is to select k that captures a significant portion (e.g., 90% or 95%) of the total variance.

Applications of PCA

PCA is widely used in various fields, including image and signal processing, bioinformatics, finance, and natural language processing. It aids in feature selection, noise reduction, and visualization of high-dimensional data.

Left singular vectors

The left singular vectors of a matrix A are the eigenvectors of AA^T. They represent the input space of the matrix transformation. The SVD produces these as the columns of the matrix U.

Right singular vectors

The right singular vectors of a matrix A are the eigenvectors of A^TA. They represent the output space of the matrix transformation. The SVD produces these as the columns of the matrix V.

Singular Value Decomposition (SVD)

SVD is a matrix factorization method that decomposes a matrix A into three separate matrices: A = UΣV^T, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing singular values.

Singular Values in SVD

Singular values in Σ represent the magnitude of scaling applied by the transformation. They are arranged in descending order, indicating the importance of corresponding singular vectors in U and V^T.

Interpreting U and V^T in SVD

Columns of U are left singular vectors, representing the transformation of the input space. Columns of V^T are right singular vectors, indicating the transformation in the output space. They form orthonormal bases for the input and output spaces, respectively.

Pseudo-inverse using SVD

The pseudo-inverse A^+ of a matrix A (A^+ = VΣ^+U^T) can be computed using SVD. Σ^+ is obtained by taking the reciprocal of non-zero singular values and transposing the resulting diagonal matrix.

Pseudo-Inverse

Denoted as A^+, is a generalization of matrix inversion for non-square matrices. In essence, it provides a solution that minimizes the error, making it valuable in applications like least squares regression and solving inconsistent or underdetermined systems.

Low-Rank Approximation using SVD

SVD enables low-rank approximation of a matrix A. By retaining the first k singular values and their corresponding singular vectors (columns of U and V^T), a low-rank approximation A_k is obtained, minimizing the Frobenius norm ||A - A_k||.

Condition Number

The condition number of a matrix A, κ(A), is the ratio of the largest to smallest singular value. A higher condition number indicates a ill-conditioned matrix, making numerical computations sensitive to small changes in input data.

Applications in Recommendation Systems

SVD-based methods are used in recommendation systems, where user-item interaction matrices are factorized to discover latent patterns. Singular values capture the importance of latent features, aiding in personalized recommendations.

Linear Systems

A linear system consists of multiple linear equations with the same variables. It can be represented as Ax = b, where A is the coefficient matrix, x is the column vector of variables, and b is the column vector of constants.

Consistent Linear System

A linear system is consistent if it has at least one solution, meaning the equations have a common intersection point. Consistent systems can be either independent, dependent, or overdetermined.

Independent Linear System

An independent system has a unique solution. The number of equations is equal to the number of unknowns, and the determinant of the coefficient matrix is non-zero.

Dependent Linear System

A dependent system has infinitely many solutions. The number of equations is less than the number of unknowns, leading to multiple solutions that lie along a line, plane, or higher

Solving Linear Systems using Matrix Inversion

For a square matrix A, the system Ax = b can be solved by multiplying both sides of the equation by A^(-1), the inverse of A. This gives the solution x = A^(-1)b, assuming A is invertible.

Matrix Rank

The rank of a matrix A, denoted as rank(A), is the number of dimensions preserved when transforming by A. It represents the dimension of the vector space spanned by the rows or columns of the matrix. Not to be confused with tensor rank.

Full Rank

A matrix which has Rank(A) = number of rows of A, and therefore is non-singular/invertible

Deficient Rank

A matrix with Rank(A) < number of rows of A, and therefore is singular and cannot be inverted

Null rank

A matrix with Rank(A) = 0, and therefore is singular and cannot be inverted (the zero matrix)

Calculation of Rank

The rank of a matrix can be determined using various methods including singular value decomposition (SVD) and counting the non-zero singular values.

Rank and Eigenvalues

The rank of a square matrix A is equal to the number of non-zero eigenvalues.

Rank and singular values

The rank of a matrix A is equal to the number of non-zero singular values.

Adjacency Matrix

An adjacency matrix is a square matrix used to represent a graph. For a graph with n vertices, the adjacency matrix A is an n x n matrix, where A[i, j] equals 1 if there is an edge between vertex i and vertex j, and 0 otherwise.

Directed Graphs

In the adjacency matrix of a directed graph, A[i, j] equals 1 if there is a directed edge from vertex i to vertex j. For an unconnected pair of vertices, A[i, j] equals 0.

Undirected Graphs

In the adjacency matrix of an undirected graph, A[i, j] equals 1 if there is an edge between vertex i and vertex j. Importantly, for undirected graphs, the matrix is symmetric, meaning A[i, j] = A[j, i] for all i and j.

Weighted Graphs

For weighted graphs, the adjacency matrix may contain weights instead of just 1s and 0s. A[i, j] represents the weight of the edge between vertex i and vertex j. If there is no edge, the entry is typically set to a special value like infinity.

Positive Definite Matrix

A symmetric matrix A is positive definite if, for any non-zero vector x, the quadratic form x^T Ax is positive, i.e., x^T Ax > 0. Positive definite matrices have all positive eigenvalues. Never flips the sign of a vector. Always invertible.

Positive Semidefinite Matrix

A symmetric matrix A is positive semidefinite if, for any non-zero vector x, the quadratic form x^T Ax is non-negative, i.e., x^T Ax ≥ 0. Positive semidefinite matrices have non-negative eigenvalues. Never flips the sign of a vector.

Negative Definite Matrix

A symmetric matrix A is negative definite if, for any non-zero vector x, the quadratic form x^T Ax is negative, i.e., x^T Ax < 0. Negative definite matrices have all negative eigenvalues. Always flips the sign of a vector. Always invertible.

Negative Semidefinite Matrix

A symmetric matrix A is negative semidefinite if, for any non-zero vector x, the quadratic form x^T Ax is non-positive, i.e., x^T Ax ≤ 0. Negative semidefinite matrices have non-positive eigenvalues. Always flips the sign of a vector.

Indefinite Matrix

A symmetric matrix A is indefinite if it is neither positive nor negative definite. In other words, there exist vectors x and y such that x^T Ax > 0 and y^T Ay < 0. Indefinite matrices have both positive and negative eigenvalues. Flips the sign of some vectors.

Orthogonal matrix

An orthogonal matrix is a square matrix whose columns and rows are orthogonal unit vectors (orthonormal vectors). The inverse of an orthogonal matrix is equal to its transpose. A pure *rotation* matrix.

Pure rotation matrix

an orthogonal matrix that represents a rotation in space

Pure scaling matrix

a diagonal matrix that represents a scaling in space

Sparse matrix

A sparse matrix is a matrix with a large number of zero values. The opposite of a sparse matrix is a dense matrix, which has few zero values.

Markov Chain

A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. It can be represented as a square matrix called the transition matrix.

Transition Matrix

A transition matrix is a square matrix that describes the probabilities of moving from one state to another in a Markov chain. It is a stochastic matrix, meaning all its entries are non-negative and each column sums to 1.

Low rank approximation

An approximation of a matrix A. It is obtained by retaining the first k singular values and their corresponding singular vectors (columns of U and V^T).

Whitening

a rigorous form of normalisation of a dataset that transforms it into a Gaussian distribution with zero mean and unit covariance. It is a linear transformation that removes redundancy in the data, making it useful in various machine learning algorithms.

Demeaning

a form of normalisation of a dataset that transforms it into a zero mean distribution by subtracting the mean.

Standardisation

a form of normalisation of a dataset that transforms it into a zero mean distribution with unit variance in each dimension by subtracting the mean and dividing by the standard deviation.

Unit 06 Optimisation

parameters in optimization

variables that the optimization algorithm adjusts to find the best solution

objective function

a mathematical function representing the quantity to be optimized (maximized or minimized)

roles of objective function

measures the performance of the solution and guides the optimization process

equality constraints

constraints that require specific conditions to be met exactly in the optimization problem

inequality constraints

constraints that define conditions that must be satisfied in the optimization problem, but not necessarily exactly

penalisation

technique to handle constraints by adding penalty terms to the objective function for violating constraints

penalty function

a mathematical function that quantifies the violation of constraints or undesirable parameter values, influencing the optimization process

approximation form of optimization

optimization problems where the objective function is an approximation, often used in machine learning tasks

penalty parameter

a tuning parameter (usually λ) that controls the strength of penalization, balancing between fitting the data and penalizing the complexity of the model

approximation form of optimization

$$L(\theta) = \sum \|y_i - f(x_i, \theta)\|$$

represents the approximation form of optimization where it minimizes the sum of absolute differences between observed and predicted values

differentiability in optimization

property of a function that allows it to have derivatives, crucial for gradient-based optimization algorithms

local optimum

a solution that is optimal within a specific neighborhood, but not necessarily the best solution globally

global optimum

the best possible solution across the entire feasible region of an optimization problem

convexity in optimization

a property of the objective function where any line segment connecting two points on the graph lies above the graph, equivalent to a single global optimum

saddle point

a point in the objective function where it is neither a local minimum nor a local maximum

linear regression in optimization framework

modeling linear regression as an optimization problem to find the best-fitting line for given data points

convex sums

technique of aggregating multiple objective functions into a single objective function by taking a weighted sum

heuristic optimization

a trial-and-error approach to problem-solving, using rules of thumb and intuition to find approximate solutions. No guarantee of optimality

convergence graphs

graphical representation showing how the optimization algorithm's objective function value against iteration - a key diagnostic tool for optimization

grid search

exhaustive search method that becomes computationally expensive with high-dimensional parameter spaces

hyperparameters vs parameters

parameters are optimized during the learning process, while hyperparameters are set to tune or configure the optimisation process itself

random search in optimization

an optimization technique that explores the search space randomly, suitable for high-dimensional problems

stochastic hill climbing

iterative optimization algorithm that makes small random steps in the search space to find the optimal solution

genetic algorithms in optimization

optimization technique that mimics the process of natural selection and genetic evolution to find optimal solutions

mutation

mutation introduces diversity in genetic algorithms by adding noise (as in stochastic hill-climbing)

crossover

combines genetic material (parameter vectors) from two parents (other solutions) to create offspring

convexity

a property where the objective function forms a convex shape, allowing for efficient optimization. There is a single global optimum.

convex sums in optimization

combining multiple convex objective functions to create a convex overall objective function

convex functions

functions where the line segment between any two points on the graph lies above the graph itself

concave functions

functions where the line segment between any two points on the graph lies below the graph itself

convex combination

a combination of vectors where the weights are non-negative and sum up to 1

continuity

a property of functions where small changes in the input result in small changes in the output

discontinous

a property of functions where arbitarily small changes in the input can result in arbitarily large changes in the output

C^0 continuity

a function is continuous, but not necessarily differentiable (has no gaps or jumps in the graph)

C^1 continuity

a function is continuously differentiable (has a continuous first derivative)

C^2 continuity

a function has a continuous first and second derivative

C^∞ continuity

a function has every derivative continuous

differentiability in optimization

a property of functions where the derivative at a point exists, indicating the rate of change or gradient

continuity and differentiability relationship

all differentiable functions are continuous, but not all continuous functions are differentiable

functions that are continuous but not differentiable

examples include the absolute value function and the square root function

functions that are both continuous and differentiable

examples include linear functions, sine, cosine, x^2

functions that are neither continuous nor differentiable

examples include step functions

discontinuous function example

$$f(x) = 1/x$$

(reciprocal function) -- discontinuous at x = 0

C^0 continuous function example

$$f(x) = |x|$$

(absolute value function) -- derivative is undefined at x = 0

C^1 continuous function example

$$f(x) = x^2$$

(quadratic function)

C^2 continuous function example

$$f(x) = 3x^3 - 2x^2 + 5x - 1$$

(cubic function)

C^∞ (infinitely differentiable) continuous function example

f(x) = e^x (exponential function)

temperature

a metaheuristic, controlling the likelihood of accepting worse solutions to escape local optima

memory

a metaheuristic where an optimisation algorithm remembers past solutions and use them to guide the search process

stigmergy

a communication mechanism where agents indirectly interact with each other through modifications of their environment (population + memory)

population

a group of potential solutions or individuals in algorithms like genetic algorithms and particle swarm optimization

genetic algorithms

optimization algorithms inspired by the process of natural selection, involving crossover and mutation operations. Hill climbing + population.

simulated annealing

probabilistic optimization algorithm inspired by the annealing process in metallurgy, allowing for exploration of the solution space. Hill climbing + temperature.

temperature schedule

a function that decreases the temperature over time, controlling the annealing process in simulated annealing.

hyperparameters of grid search

grid spacing, the range of values for each parameter

hyperparameters of stochastic hill climbing

step distribution (or step size)

hyperparameters of genetic algorithms

population size, crossover probability, mutation probability

general objective function in optimization

f(x, θ) represents the function to be minimized or maximized, where x is the input and θ are the parameters

approximation form of optimization

L(θ) = Σ |y_i - f(x_i, θ)| represents the objective function as the sum of absolute differences between observed (y_i) and predicted (f(x_i, θ)) values

parameter vector

an element in a vector space representing a set of parameters, often used in mathematical modeling and machine learning

Euclidean space R^n

a vector space representing n-dimensional real-valued vectors, commonly used to represent parameter vectors in various contexts

Unit 07 Gradient Descent

Jacobian matrix

matrix of all first-order partial derivatives of a vector-valued function

Jacobian determinant

scalar value indicating how a small volume in input space is scaled to output space

Gradient vector

vector containing partial derivatives representing the rate of change of a function. First row of the Jacobian matrix

Nabla

symbol (∇) denoting vector differential operator used to represent gradients or derivatives. ∇f(x) = (∂f/∂x, ∂f/∂y, ∂f/∂z, ...) if f is a scalar function of a vector variable.

Gradient Descent

an iterative optimization algorithm used to minimize a function by adjusting its parameters in the direction of the negative gradient

Gradient Descent Equation

$\theta = \theta - \alpha \nabla J(\theta)$, where θ represents parameters, α is the learning rate, $J(\theta)$ is the cost function, and $\nabla J(\theta)$ is the gradient of the cost function at θ

Learning Rate or Step Size

scalar value determining the size of steps taken during each iteration of gradient descent

Stochastic Gradient Descent

updates parameters for each example or sub-batch, suitable for large datasets, more noisy but can escape local optima

Convergence Criteria

stopping condition indicating when to stop gradient descent, commonly based on small changes in the cost function or fixed number of iterations

Gradient Descent with Momentum

utilizes a moving average of past gradients to accelerate convergence, prevents oscillation in shallow ravines

Learning Rate Schedules

techniques like step decay or exponential decay that gradually reduce the learning rate during training to fine-tune the model

Critical point

point where the gradient is zero, indicating a potential minimum, maximum, or saddle point

Local Minimum

critical point where the function has lower values than its surrounding points, indicating a potential lowest point in the local vicinity

Local Maximum

critical point where the function has higher values than its surrounding points, indicating a potential highest point in the local vicinity

Saddle Point

critical point where the function is neither a minimum nor a maximum, but the gradient is zero; can be a peak in one direction and a valley in another

Ridge Point

critical point where the function has higher values in multiple directions and lower values in orthogonal directions

Plateau Point

critical point where the function has a flat region, making it challenging to determine the nature of the critical point

Global Minimum

critical point where the function has the absolute lowest value over its entire domain, indicating the overall minimum point

Global Maximum

critical point where the function has the absolute highest value over its entire domain, indicating the overall maximum point

Hessian Matrix

matrix of second-order partial derivatives representing the curvature of a multivariable function. Used to classify critical points.

Curvature

measure of how fast a gradient is changing at a given point, indicating the shape of the curve

Eigenvalues of the Hessian Matrix

values that indicate the direction and magnitude of the curvature at a critical point, positive for minima, negative for maxima, mixed for saddle points

Positive Definite Hessian Matrix

all eigenvalues are positive, indicating a local minimum at the critical point

Negative Definite Hessian Matrix

all eigenvalues are negative, indicating a local maximum at the critical point

Indefinite Hessian Matrix

contains both positive and negative eigenvalues, indicating a saddle point at the critical point

Positive Semi-Definite Hessian Matrix

all eigenvalues are positive or zero, indicating a local minimum or plateau point at the critical point

Negative Semi-Definite Hessian Matrix

all eigenvalues are negative or zero, indicating a local maximum or plateau point at the critical point

Lipschitz Continuity

property of a function where there exists a Lipschitz constant L such that $|f(x) - f(y)| \leq L|x - y|$ for all x, y in the domain

Lipschitz Constant

the smallest positive real number L satisfying the Lipschitz condition, quantifying the function's rate of change

Lipschitz Continuous Function

function where the rate of change between any two points is bounded by a constant L, ensuring smoothness and control over oscillations

Examples of Lipschitz Continuous Functions

linear functions, functions with bounded derivatives, functions defined on closed intervals with continuous derivatives

Functions Not Lipschitz Continuous

functions with unbounded slopes, functions with infinite oscillations, functions with vertical asymptotes

Importance of Lipschitz continuity

Lipschitz continuity ensures the stability of optimization algorithms, allowing predictable convergence rates and preventing overshooting

Automatic Differentiation

technique for efficiently and accurately evaluating derivatives of mathematical expressions in code, used in machine learning and scientific computing

Forward Mode Automatic Differentiation

computes derivatives by applying the chain rule from the inside out, suitable for functions with few inputs and many outputs

Backward Mode Automatic Differentiation

computes derivatives by applying the chain rule from the outside in, efficient for functions with many inputs and few outputs (common in deep learning)

Computational Complexity of Automatic Differentiation

forward mode complexity is linear in the number of inputs, backward mode complexity is linear in the number of outputs

`autograd` Package in Python

Python library providing automatic differentiation for NumPy-like operations, simplifying gradient computation for machine learning models

grad()

an autograd function taking a (vector -> real) function as input and returning a (vector -> vector) function that computes the gradient of the input function

Tape-based AD

a method of automatic differentiation where intermediate values and operations are recorded on a tape, enabling efficient calculation of gradients

Static vs. Dynamic Computation Graphs

static graphs (common in TensorFlow) define the computation graph before runtime, dynamic graphs (common in PyTorch) allow defining the graph as operations are executed

Issues with Numerical Differentiation

problems like choosing an appropriate step size (leading to numerical instability or precision issues), round-off errors, and the curse of dimensionality in high-dimensional spaces

Numerical Differentiation

approximating derivatives by finite differences, involving small changes in input to calculate the rate of change of a function

Symbolic Differentiation

computing exact derivatives using algebraic techniques, providing precise symbolic expressions for derivatives

Automatic Differentiation

efficiently and accurately evaluating derivatives of mathematical expressions, utilized in machine learning and scientific computing for gradient-based optimization

Gradient Descent Equation

$$\theta = \theta - \partial \nabla L(\theta)$$

Gradient Vector Equation (for a scalar function f(x, y, z))

$$\nabla f(x, y, z) = (\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)$$

Unit 08 Probability

probability

a measure of the how like an event is to occur, ranging from 0 to 1 inclusive

sample space

the set of all possible outcomes of a random variable (often denoted as S)

frequency

the number of times an event occurs in a given number of trials or observations

outcomes

the possible results of an experiment or random process

samples

a subset of observations or measurements from a larger population, consisting of outcomes

events

subsets of the sample space, representing specific occurrences of interest. Every outcome is an event, but not every event is an outcome.

example of event

in a dice throw, getting an even number {2, 4, 6}

example of an outcome

in a dice throw, getting a 2

distributions

mathematical functions describing the likelihood of each possible outcome in a sample space

P(X=x) notation

probability of the random variable X taking a specific value x

P(A) notation

probability of event A occurring

P(A|B) notation

probability of event A occurring given that event B has occurred

P(A ∩ B) notation

probability of both events A and B occurring

P(A ∪ B) notation

probability of either event A or event B occurring

random variable

a variable that can take on different values, each with a certain probability

discrete random variable

a random variable that can only take distinct, separate values, often countable

continuous random variable

a random variable that can take any value over a continuous domain

probability mass function (PMF)

a function that describes the probabilities of discrete random variables taking specific values

probability density function (PDF)

a function that describes the probabilities of continuous random variables falling within specific intervals via densities

density

a non-negative value that describes the relative likelihood of a continuous random variable taking on a given value. Not a probability. May be greater than 1.

axioms of probability

fundamental rules governing probability theory, including unitarity, non-negativity, and additivity (sum rule)

axiom of unitarity

the total probability of all possible outcomes in a sample space is equal to 1

axiom of non-negativity

probabilities are never negative and can only range from 0 to 1

axiom of additivity (sum rule)

the probability of the union of mutually exclusive events is the sum of their individual probabilities

conditional probability

the probability of an event occurring given that another event has already occurred, denoted as P(A|B) where A and B are events

conditional probability from a PMF

P(X=x | Y=y) = P(X=x ∩ Y=y) / P(Y=y), where X and Y are discrete random variables, and P(X=x | Y=y) represents the probability of X taking a specific value x given that Y has taken a specific value y. P(X=x ∩ Y=y) represents the joint probability of X and Y both taking specific values, and P(Y=y) represents the probability of Y taking the value y.

forward probability

the likelihood of observing certain outcomes given specific parameters or conditions, often used in predictive modeling and simulations

inverse probability

the probability of specific parameters or conditions given observed outcomes, used in tasks like parameter estimation and hypothesis testing

Bayesian interpretation of probability

a perspective that treats probability as a measure of belief or uncertainty, incorporating prior knowledge and updating beliefs based on new evidence

frequentist interpretation of probability

a perspective that defines probability as the limit of relative frequency in a large number of trials, focusing on long-term behavior and objective outcomes

Bayes' Rule equation

P(A|B) = (P(B|A) * P(A)) / P(B), where P(A|B) is the posterior probability of event A given event B, P(B|A) is the likelihood of event B given event A, P(A) is the prior probability of event A, and P(B) is the probability of event B.

Bayes' Rule interpretation

a mathematical formula that updates the probability of an event (A) based on new evidence (B), balancing prior beliefs with the likelihood of the evidence given those beliefs

Posterior

(P(A|B)) the updated probability of event A given new evidence B

Likelihood

(P(B|A))the probability of observing evidence B given that event A has occurred

Prior

(P(A)) the initial probability of event A before considering the new evidence

Evidence

the probability of observing evidence B, also known as the marginal likelihood or total probability of B across all possible events

Marginal Likelihood

(P(B)) the probability of observing evidence B, also known as the evidence of B across all possible events

Integration over the evidence in Bayes' Rule

the process of summing or integrating the likelihood (P(B|A)) times the prior (P(A)) for all possible values of event A to obtain the marginal likelihood/evidence (P(B))

joint distribution

a probability distribution that describes the probabilities of all possible combinations of values for two or more random variables

marginal distribution

the probability distribution of a subset of random variables obtained by summing or integrating over the other variables in the joint distribution

conditional distribution

the probability distribution of one or more random variables given specific values or ranges of values for other variables, obtained from the joint distribution

marginalization formula

to find the marginal probability of a subset of random variables, sum or integrate the joint probability distribution over the other variables. For two discrete variables X and Y: P(X) = ∑(P(X, Y=y)) for all possible y values.

conditioning formula

to find the conditional probability of variable X given variable Y, divide the joint probability of X and Y by the marginal probability of Y. For discrete variables: $P(X|Y) = P(X, Y) / P(Y)$.

bigrams

sequences of two adjacent elements or items, often used in text analysis and natural language processing

n-gram models

statistical language models that predict the likelihood of a word or sequence of words based on the previous n-1 words

likelihood

the probability of observed data (usually given specific values of parameters in a statistical model), often denoted as $L(x)$ or $L(x|\theta)$ where θ represents parameters and x represents observed data

likelihood function

a function that represents the likelihood of parameters in a statistical model given observed data

likelihood of multiple independent events

in the case of independent events, the overall likelihood is the product of the likelihoods of individual events, often denoted as $L(x1, x2, ..., xn) = L(x1) * L(x2) * ... * L(xn)$

log-likelihood

the natural logarithm of the likelihood function, often used for computational convenience as it transforms products into sums, reducing underflow problems: $\text{log-likelihood} = \log(L(x))$

expectation

a measure of the center or average value of a random variable, representing the long-term average of its outcomes

intuitive meaning of expectation

the expected value represents what we would anticipate on average if a random experiment were repeated a large number of times

expected value

the expected value of a random variable X is denoted as $E(X)$ or μ, calculated as the sum of each possible value x multiplied by its probability $P(X=x)$ for discrete random variables or the integral of each possible value x multiplied by its probability density $f(x)$ for continuous random variables

expectation formula (for a discrete random variable)

$E(X) = \Sigma(x * P(X=x))$ for all possible values x, where X is the random variable and $P(X=x)$ is its probability mass function

expectation formula (for a continuous random variable)

$E(X) = \int(x * f(x))\, dx$ over the entire range of possible values, where X is the random variable and $f(x)$ is its probability density function

expectation of a function of a random variable

$E[g(X)] = \Sigma(g(x) * P(X=x))$ for discrete random variables or $E[g(X)] = \int(g(x) * f(x))\, dx$ for continuous random variables, where $g(x)$ is a function of the random variable X

entropy

a measure of uncertainty or disorder in a random variable, indicating the average amount of information needed to describe its outcomes

entropy equation (for discrete random variables)

$H(X) = -\Sigma(P(X=x) * \log_2(P(X=x)))$ for all possible values x, where $H(X)$ represents the entropy of the random variable X

entropy interpretation

entropy quantifies the uncertainty associated with a random variable. Higher entropy values indicate greater uncertainty, while lower values indicate more predictability and less uncertainty.

information (in bits)

a unit used to measure the amount of uncertainty or surprise associated with an event or outcome

bit

a binary digit, representing a choice between two mutually exclusive options (0 or 1)

entropy as average information

entropy $H(X) = -\Sigma(P(X=x) * \log_2(P(X=x)))$ for all possible values x, representing the average amount of information required to specify an outcome in a random variable X

entropy as expected log probability

entropy $H(X) = -E[\log_2(P(X=x))]$ for all possible values x, representing the expected log probability of an outcome in a random variable X

information theory

a branch of mathematics and computer science that studies the quantification, storage, and communication of information, exploring concepts such as entropy, mutual information, and data compression

logits

the natural logarithm of the odds, representing the relationship between the probability of an event and its complement, often used in logistic regression models. -ve means less likely than even, +ve means more likely than even

odds

the ratio of the probability of an event occurring to the probability of it not occurring, calculated as P(event) / P(not event)

log-probabilities

natural logarithm of probabilities, useful for numerical stability and simplifying calculations, often employed in machine learning algorithms, including softmax functions and neural networks

log-odds

the natural logarithm of the odds, often used in logistic regression models. Also called logits.

logistic function

a function that maps real values in (-inf, inf) to probabilities between 0 and 1, often used in logistic regression models. Also called the sigmoid function.

empirical distribution

a probability distribution derived from observed data, representing the frequencies of different outcomes in the sample

sampling from distributions

generating random samples from a given probability distribution, often used in simulations and statistical analysis to approximate real-world scenarios

Monte Carlo approximation to expectation

using random sampling to estimate the expectation of a function, calculated as the average of function evaluations over a large number of random samples, providing an approximate solution to complex problems

empirical distribution

$P(X = x) = \frac{\text{Number of occurrences of } x}{\text{Total number of observations}}$, where X represents the random variable and x represents a specific outcome in the sample.

sampling from distributions

$X \sim F(x)$, where $F(x)$ is the probability distribution function. Random samples x are drawn from $F(x)$ to approximate the behavior of the distribution.

Monte Carlo approximation to expectation

$E(g(X)) \approx \frac{1}{N} \sum_{i=1}^{N} g(x_i)$, where N is the number of random samples drawn from the distribution X, x_i represents each sample, and $g(x_i)$ is the function to be evaluated.

Monte Carlo error

the difference between the estimated value obtained through Monte Carlo simulations and the true value, decreasing as the number of samples (N) increases at a rate of $O(\frac{1}{\sqrt{N}})$

convergence of Monte Carlo methods

the property where the estimates obtained through Monte Carlo simulations approach the true values as the number of samples increases

Unit 09 Inference

support

the range of values that a random variable can take -- can be compact (finite) or infinite

compact support

values within a specific, finite range

infinite support

values that extend infinitely, without any upper or lower bounds

semi-infinite support

values that extend infinitely in one direction, but are bounded in the other direction (e.g. 0 to infinity)

probability density function (pdf)

a function that describes the likelihood of a continuous random variable taking a specific value -- represents densities, not probabilities

meaning of density versus probability

pdf values indicate relative concentration of probabilities, whereas probabilities represent the integral of the pdf over a range of values

cumulative distribution function (cdf)

a function that gives the probability that a random variable takes a value less than or equal to a given value

continuous random variables

variables that can take any real value within a given range

issues with continuous random variables

the inability to assign probabilities to individual outcomes due to infinite possibilities

uniform distribution

probability distribution where every outcome is equally likely

multivariate distributions

probability distributions over vector spaces

univariate normal

the normal distribution for a single random variable with mean (μ) and variance (σ^2)

properties of the normal distribution

bell-shaped, symmetric, characterized by mean and variance

multivariate normal

probability distribution over a real vector space, represented by a mean vector and covariance matrix

mean vector

the expected values for each variable in a multivariate normal distribution

covariance matrix

measures the strength and direction of linear relationships between variables in a normal distribution

central limit theorem (CLT)

a fundamental theorem in probability theory stating that the distribution of the sum (or average) of a large number of independent, identically distributed random variables approaches a normal distribution, regardless of the original distribution of the variables

conditions for CLT

1. The random variables must be independent. 2. The random variables must be identically distributed. 3. The sample size must be sufficiently large.

sampling distribution

the distribution of a statistic (e.g., mean or sum) calculated from a sample as the sample size increases, following the CLT

population distribution

the distribution of values in the entire population from which a sample is drawn

standard error

the standard deviation of a sampling distribution, representing the variability of sample means around the population mean

implications of CLT

1. Allows the use of normal distribution approximations in various statistical analyses. 2. Justifies hypothesis testing and confidence interval construction for various sample statistics.

importance in inferential statistics

enables making inferences about population parameters based on sample statistics, assuming the sample size is large enough

limitations of CLT

may not apply if the sample size is small or the underlying population distribution has heavy tails or is not well-behaved

population

the entire set of individuals, objects, or measurements of interest to a particular study

sample

a subset of individuals, objects, or measurements selected from a population for the purpose of making inferences about the population

random sampling

the process of selecting a sample from a population in such a way that every possible sample of a given size has an equal chance of being chosen

sampling bias

occurs when the method of selecting a sample causes the sample to be unrepresentative of the population, leading to skewed or inaccurate results

representative sample

a sample that accurately reflects the characteristics of the population from which it is drawn

sampling error

the difference between a sample statistic and the corresponding population parameter caused by chance variation in the selection of the sample

population parameter

a numerical measure that describes an aspect of a population, such as a mean or proportion

sample statistic

a numerical measure that describes an aspect of a sample, such as a sample mean or sample proportion

inference

the process of drawing conclusions or making predictions about a population based on information obtained from a sample

generalizability

the extent to which the findings from a sample can be applied to the larger population

varieties of inference

direct, maximum likelihood estimation, and Bayesian

direct inference

making conclusions about population parameters based on observed sample statistics without formal statistical methods

maximum likelihood estimation (MLE)

a method of estimating the parameters of a statistical model by maximizing the likelihood function, representing the probability of observed data given the parameters

likelihood function

the probability of observing the given data for different values of the parameters in a statistical model

Bayesian inference

a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available

prior probability

the initial probability assigned to a hypothesis before considering the observed data in Bayesian inference

likelihood

the probability of observing the given data for a specific set of parameters in Bayesian inference

posterior probability

the updated probability of a hypothesis after considering the observed data in Bayesian inference, obtained by combining the prior probability and the likelihood

Bayes' theorem

a mathematical formula that describes the relationship between the prior probability, likelihood, and posterior probability in Bayesian inference

advantages of Bayesian inference

incorporates prior knowledge, provides a coherent framework for decision-making, and handles small sample sizes well

disadvantages of Bayesian inference

requires specifying a prior distribution, can be computationally intensive, and interpretation of subjective priors can be challenging

probabilistic programming

a programming paradigm that allows the incorporation of probability distributions and Bayesian inference into computer programs, enabling modeling of complex, uncertain systems

Markov Chain Monte Carlo (MCMC)

a computational method used for sampling from complex probability distributions, especially in Bayesian inference

MCMC algorithm steps

1. Start with an initial state. 2. Propose a new state based on a transition probability. 3. Accept or reject the new state based on acceptance probability. 4. Repeat steps 2-3 to generate a chain of states.

Metropolis-Hastings algorithm

an MCMC algorithm that generalizes the basic MCMC approach by using a proposal distribution and an acceptance probability to generate a Markov chain

Burn-in period

initial phase of an MCMC simulation where the chain stabilizes and reaches the target distribution, discarding the samples generated during this phase

thinning

the process of discarding some of the samples generated by an MCMC simulation to improve convergence

convergence diagnostics

methods used to assess whether an MCMC chain has reached the target distribution, ensuring accurate inference

probabilistic programming language

a programming language designed specifically for expressing probabilistic models, making it easier to define and manipulate complex probability distributions

advantages of probabilistic programming

facilitates rapid prototyping of Bayesian models, enables easy model modification, and supports complex, hierarchical models

limitations of MCMC

computationally intensive, sensitive to the choice of initial conditions, and might require a large number of iterations to achieve convergence

posterior distribution

the probability distribution of the parameters of a statistical model given observed data, obtained by applying Bayes' theorem

posterior predictive

the probability distribution of future observations given observed data, obtained by integrating over the posterior distribution of the parameters

prior predictive

the probability distribution of future observations given a prior distribution of the parameters, obtained by integrating over the prior distribution

linear regression by direct estimation

a method of estimating the parameters of a linear regression model by minimizing the sum of squared errors between the observed and predicted values, e.g. via pseudo-inverse

linear regression by maximum likelihood estimation

a method of estimating the parameters of a linear regression model by maximizing the likelihood function, representing the probability of observed data given the parameters (gradient, offset, and variance)

linear regression by Bayesian inference

a method of estimating the parameters of a linear regression model by using Bayes' theorem to update the prior probability of the parameters based on the observed data

Unit 10 Signals

evenly spaced samples

samples taken at regular intervals from a continuous signal

consistent quantization

uniform assignment of digital values to analog signal levels

quantization levels

number of distinct amplitude values in a digital signal

quantization noise

error introduced due to the finite number of quantization levels

sampling rate

number of samples taken per unit of time from a continuous signal

Nyquist rate

minimum sampling rate required to accurately represent a signal without aliasing

aliasing

artifacts caused by undersampling, creating false frequencies in the reconstructed signal

aliasing effects

misinterpretation of high-frequency signals as low-frequency signals in the sampled data

aliasing mitigation

preventing aliasing by using a sampling rate higher than the Nyquist rate

SNR (Signal-to-Noise Ratio) formula

SNR = 20 * log10 (Amplitude of Signal / RMS Noise Amplitude)

Frequency of aliased components formula

Aliased Frequency = |k * Sampling Frequency - Original Frequency|

Quantization SNR formula

Quantization SNR = 6.02 * (Number of Bits) + 1.76 dB

interpolation of irregular data

estimating unknown data points within a dataset with irregularly spaced points

common interpolation methods

linear, polynomial, sinc, and nearest-neighbor interpolation

linear interpolation

estimating unknown values by connecting adjacent known data points with straight lines

polynomial interpolation

approximating data points using a polynomial equation fitted to neighboring points

nearest-neighbor interpolation

estimating unknown values based on the nearest known data point

resampling

process of changing the sampling rate of a signal, altering the number of samples per unit time

sliding window technique

a method to analyze data within a fixed-size window that slides over the dataset

running mean

average computed over a fixed-size window as it moves through the data

moving average

series of averages calculated from different subsets of the dataset, useful for smoothing

linear filtering

process of modifying an input signal using a linear operation (a weighted sum), producing an output signal

weighted averages

calculations that assign different weights to values in a window when computing a moving average

convolution

mathematical operation combining two functions to produce a third, by multiplying and summing

convolution in signal processing

process of combining input signal and filter kernel to create an output signal

linear filtering using convolution

applying a filter kernel to input signal via convolution to obtain filtered output

convolution formula

$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) * g(t - \tau)\, d\tau$, where * denotes convolution

convolution properties

commutativity, associativity, distributivity, and identity element exist in convolution

common types of filters

low-pass, high-pass, and band-pass filters

low-pass filter

allows low-frequency components to pass through while attenuating high-frequency components

high-pass filter

allows high-frequency components to pass through while attenuating low-frequency components

band-pass filter

allows a specific range of frequencies to pass through, attenuating others

kernel

array of coefficients used in linear filtering to modify the input signal. We convolve a signal with a kernel specifying a filter.

filter kernel coefficients

weights assigned to each element of the kernel, affecting the filter's behavior

filter design

process of selecting kernel size and coefficients to achieve desired filtering effect

impulse response

the output of a system when an impulse signal (Dirac delta function) is applied as input

impulse response function

representation of a system's output in response to an impulse signal

impulse response recovery

process of determining a system's behavior by analyzing its impulse response

impulse response recovery methods

deconvolution techniques, Fourier transform, and inverse filtering

deconvolution

mathematical operation used to recover the original signal from its convolution with an impulse response

impulse response recovery challenges

noise interference, finite duration of signals, and ill-conditioned systems

Dirac delta function

mathematical function representing an idealized impulse at a single point in space or time ("spike" function)

properties of Dirac delta function

integral over the entire real line is 1, and it is zero everywhere except at 0

Dirac delta in convolution

Dirac delta function is the identity element in convolution, i.e. $f * \delta = f$

nonlinear filtering

filtering technique where the computation is anything other than a weighted sum

median filtering

nonlinear filtering method where the output is the median value of the neighboring elements in the window

advantages of median filtering

eliminates spike noise, preserves edges, and is computationally efficient

pure frequencies

signals consisting of a single frequency without any harmonics or overtones

sinusoids

signals characterized by a sinusoidal waveform, represented as $A * \sin(2\pi ft + \varphi)$, where A is amplitude, f is frequency, t is time, and φ is phase angle

amplitude of a sinusoid

maximum displacement of the waveform from its equilibrium position

frequency of a sinusoid

number of complete cycles occurring in one unit of time, measured in Hertz (Hz)

phase angle of a sinusoid

offset in the horizontal direction, indicating the starting point of the waveform

Fourier Transform

mathematical technique that transforms a time-domain signal into its frequency components

continuous Fourier Transform equation

F(ω) = ∫[from -∞ to ∞] f(t) * e^(-jωt) dt, where F(ω) represents the frequency spectrum of f(t)

Discrete Fourier Transform (DFT)

discretized version of the Fourier Transform, applied to sampled, discrete-time signals

DFT equation

X(k) = Σ[n=0 to N-1] x(n) * e^(-j(2π/N)kn), where X(k) represents the frequency component at index k. X(k) is a complex number with magnitude and phase.

Complex number

a number that can be expressed in the form a + bi, where a and b are real numbers, and i is the imaginary unit, satisfying i^2 = -1

magnitude and phase angle

two components of a complex number, representing its amplitude and phase angle. Amplitude = sqrt(a^2 + b^2), Phase = arctan(b/a)

Argand diagram

a graphical representation of complex numbers as points in the complex plane, with the real part on the x-axis and the imaginary part on the y-axis

Fast Fourier Transform (FFT)

algorithm for efficiently computing the Discrete Fourier Transform, reducing time complexity from O(N^2) to O(N log N)

time complexity of DFT

O(N^2), where N is the number of samples in the input signal

time complexity of FFT

O(N log N), significantly faster than the naive DFT calculation

Inverse Discrete Fourier Transform (IDFT)

transforms frequency-domain representation back to time domain

IDFT equation

x(n) = (1/N) * Σ[k=0 to N-1] X(k) * e^(j(2π/N)kn), where x(n) is the reconstructed time-domain signal

spatial domain

representation of a signal in terms of its amplitude at various points in space or time

frequency domain

representation of a signal in terms of its frequency components and their amplitudes

convolution theorem

states that convolution in the time domain is equivalent to multiplication in the frequency domain

implications of the convolution theorem

filtering in the frequency domain is computationally efficient, and convolution in time domain can be avoided

filter design with DFT

designing filters by manipulating their frequency response in the frequency domain

steps for filter design using DFT

1. Determine desired frequency response 2. Compute inverse DFT to get filter coefficients 3. Apply these coefficients for convolution

frequency response

representation of a filter's behavior as a function of frequency, showing how it affects different frequencies

sinc interpolation

interpolation technique that uses sinc function as the interpolation kernel

sinc function

mathematical function defined as sinc(x) = sin(πx) / (πx), with a central peak at x = 0 and zeros at other integers

ideal interpolation kernel

the sinc function, providing perfect reconstruction of the original signal without distortion

sinc interpolation process

multiply input samples with sinc function, summing the results to obtain interpolated values

limitations of sinc interpolation

computational intensity due to infinite support, requires truncation or windowing for practical implementation

windowing functions

mathematical functions used to taper the edges of a signal before applying Fourier analysis or other algorithms

use of windowing functions

minimizing spectral leakage, improving frequency resolution, and reducing artifacts in Fourier analysis

application of windowing functions

applied to time-domain signals before performing Fourier Transform or other frequency analysis methods

lobes

distinct sections in the frequency spectrum of a signal or a windowing function

main lobe

central and largest lobe in the frequency spectrum, containing the primary frequency content of the signal

sidelobes

smaller lobes on both sides of the main lobe in the frequency spectrum

sidelobe level

amplitude of the sidelobes relative to the main lobe, indicating the extent of spectral leakage

spectral leakage

phenomenon where signal energy spreads into adjacent frequency bins, causing inaccuracies in frequency analysis

windowing to reduce sidelobes

applying window functions helps reduce sidelobe levels, minimizing spectral leakage and improving frequency analysis accuracy

spectrum

representation of a signal's frequency content, showing the amplitudes and phases of each frequency component

magnitude spectrum

computing the absolute values of the Fourier Transform coefficients

phase spectrum

computing the angles (phases) of the Fourier Transform coefficients

importance of magnitude spectrum

determines the amplitudes of frequency components, vital in applications like filtering and modulation

importance of phase spectrum

determines the relative positions/offsets of frequency components, vital in applications like signal reconstruction

X Formulae

Notation for a vector

$$\mathbf{x}$$

Notation for a matrix

$$A$$

Relative precision of floating point representation

$$\epsilon = \frac{|\text{float}(x) - x|}{|x|}$$

IEEE754 guarantee for relative precision (machine precision ϵ) for a t bit mantissa floating point number

$$\epsilon = \frac{1}{2}2^{-t} = 2^{-t-1}$$

Addition of two vectors

$$\mathbf{x} + \mathbf{y} = [x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n]$$

Scalar multiplication of a vector

$$c\mathbf{x} = [cx_1, cx_2, \ldots, cx_n]$$

Linear interpolation of two vectors

$$\text{lerp}(\mathbf{x}, \mathbf{y}, \alpha) = \alpha\mathbf{x} + (1 - \alpha)\mathbf{y}$$

Cosine of angle between two vectors in terms of normalized dot product

$$\cos\theta = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||}$$

Dot product / inner product

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$$

Mean vector

$$\text{mean}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = \frac{1}{N} \sum_i \mathbf{x_i}$$

Definition of linearity (for a linear function f and equivalent matrix A)

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) = A(\mathbf{x} + \mathbf{y}) = A\mathbf{x} + A\mathbf{y}$$
$$f(c\mathbf{x}) = cf(\mathbf{x}) = A(c\mathbf{x}) = cA\mathbf{x}$$

Matrix addition

$$A + B = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \ldots & a_{1,m} + b_{1,m} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \ldots & a_{2,m} + b_{2,m} \\ \ldots & & & \\ a_{n,1} + b_{n,1} & a_{n,2} + b_{n,2} & \ldots & a_{n,m} + b_{n,m} \end{bmatrix}$$

Scalar matrix multiplication

$$cA = \begin{bmatrix} ca_{1,1} & ca_{1,2} & \ldots & ca_{1,m} \\ ca_{2,1} & ca_{2,2} & \ldots & ca_{2,m} \\ \ldots & & & \\ ca_{n,1} & ca_{n,2} & \ldots & ca_{n,m} \end{bmatrix}$$

Matrix multiplication

$$C_{ij} = \sum_k a_{ik} b_{kj}$$

Outer product (matrix version)

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{x}^T \mathbf{y}$$

Inner product (matrix version)

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}\mathbf{y}^\mathbf{T}$$

Power iteration for leading eigenvalue

$$x_n = \frac{Ax_{n-1}}{\|Ax_{n-1}\|_\infty}$$

Definition of eigenvalue and eigenvector

$$A\mathbf{x_i} = \lambda_i \mathbf{x_i}$$

Trace of a matrix

$$\mathrm{Tr}(A) = \sum_{i=1}^{n} \lambda_i$$

Determinant of a matrix

$$\det(A) = \prod_{i=1}^{n} \lambda_i$$

Linear system $Ax = y$

$$A\mathbf{x} = \mathbf{y}$$

SVD definition

$$A = U\Sigma V$$

(pseudo-)inverse by SVD

$$A^{-1} = V^T \Sigma^{\dagger} U^T$$

Definition of optimization

$$\theta^* = \operatorname{argmin} \theta \in \Theta L(\theta)$$

Objective function for approximation

$$L(\theta) = \|y' - y\| = \|f(\mathbf{x}; \theta) - y\|$$

Equality constraints for optimization

$$\theta^t = \operatorname{argmin} \theta \in \Theta L(\theta) \text{ subject to } c(\theta) = 0$$

Inequality constraints for optimization

$$\theta^* = \operatorname{argmin} \theta \in \Theta L(\theta) \text{ subject to } c(\theta) \leq 0$$

Gradient of a function

$$\nabla L(\theta) = \left[\frac{\partial L(\theta)}{\partial \theta_1}, \frac{\partial L(\theta)}{\partial \theta_2}, \ldots, \frac{\partial L(\theta)}{\partial \theta_n} \right]$$

Gradient descent algorithm

$$\theta^{(\mathrm{i}+1)} = \theta^{(\mathrm{i})} - \delta \nabla L(\theta^{(\mathrm{i})})$$

Definition of differentiation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

Hessian matrix

$$\nabla \nabla L(\theta) = \nabla^2 L(\theta) = \begin{bmatrix} \frac{\partial^2 L(\theta)}{\partial \theta_1^2} & \frac{\partial^2 L(\theta)}{\partial \theta_1 \partial \theta_2} & \frac{\partial^2 L(\theta)}{\partial \theta_1 \partial \theta_3} & \cdots & \frac{\partial^2 L(\theta)}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 L(\theta)}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 L(\theta)}{\partial \theta_2^2} & \frac{\partial^2 L(\theta)}{\partial \theta_2 \partial \theta_3} & \cdots & \frac{\partial^2 L(\theta)}{\partial \theta_2 \partial \theta_n} \\ \cdots & & & & \\ \frac{\partial^2 L(\theta)}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 L(\theta)}{\partial \theta_n \partial \theta_2} & \frac{\partial^2 L(\theta)}{\partial \theta_n \partial \theta_3} & \cdots & \frac{\partial^2 L(\theta)}{\partial \theta_n^2} \end{bmatrix}$$

Convex sum of sub-objective functions

$$L(\theta) = \lambda_1 L_1(\theta) + \lambda_2 L_2(\theta) + \cdots + \lambda_n L_n(\theta)$$

Expectation of a random variable

$$\mathbb{E}\left[X\right] = \int_x x \; f_X(x) dx$$

Expectation of a function of a random variable

$$\mathbb{E}\left[g(X)\right] = \int_x f_X(x) g(x) dx$$

Definition of Nyquist limit

$$f_n = \frac{f_s}{2}$$

Signal to noise ratio

$$SNR = \frac{S}{N}$$

SNR in dB

$$SNR_{dB} = 10 \log_{10} \left(\frac{S}{N} \right)$$

Exponential smoothing

$$y[t] = \alpha y[t-1] + (1-\alpha)x[t]$$

Convolution of sampled signals

$$(x * y)[n] = \sum_{m=-M}^{M} x[n]y[n-m]$$

Convolution of sampled signals

$$(x * y)[n] = \sum_{m=-M}^{M} x[n]y[n-m]$$

Discrete Fourier Transform (DFT) equation

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi i}{N} kn}$$

X Numpy

@ (Matrix Multiplication Operator)

Performs matrix multiplication between two arrays. Example: result = x @ y computes the matrix multiplication of arrays x and y.

& (Bitwise AND operator)

Performs element-wise logical AND between two arrays. Example: x & y computes the element-wise logical AND between arrays x and y.

| (Bitwise OR operator)

Performs element-wise logical OR between two arrays. Example: x | y computes the element-wise logical OR between arrays x and y.

~ (Bitwise NOT operator)

Performs element-wise logical NOT on an array. Example: ~x computes the element-wise logical NOT of array x.

anp.grad(fn)

with autograd takes a function fn and returns a function that computes the gradient of fn with respect to its inputs. Example: grad_fn = grad(fn) returns a function grad_fn that computes the gradient of fn with respect to its inputs.

array.astype(dtype)

Casts the array to the specified data type. Example: x.astype(np.float64) converts the array x to the data type float64.

array.dtype

Returns the data type of the array elements. Example: x.dtype returns the data type of elements in array x.

array.ndim

Returns the number of array dimensions. Example: x.ndim returns the number of dimensions of array x.

array.ravel() or array.flatten()

Returns a flattened array containing the same elements as the input array. Example: x.ravel() returns a flattened version of array x.

array.shape

Returns a tuple representing the dimensions of the array. Example: x.shape returns the dimensions of array x.

array.strides

Returns a tuple of bytes to step in each dimension when traversing an array. Example: x.strides returns the strides of array x.

array.T

Returns the transposed array. Example: x.T transposes the array x. For higher-dimensional arrays, reverses the order of the axes.

array[:, np.newaxis] or array[:, None]: Adds a new axis to an array. Example: x[:, np.newaxis] adds a new (singleton) axis at the end of array x.

array[..., 0]

The ellipsis (...) represents as many colons as needed to produce a complete indexing tuple. Example: x[..., 0] returns the first element of the last axis of array x.

array[condition]

Returns an array containing elements from the input array where the condition is True. Example: x[x > 0] returns an array containing all positive elements of x.

array[index_array]

Returns an array containing elements from the input array at the specified indices. Example: x[[0, 2, 4]] returns an array containing the elements at indices 0, 2, and 4 of x.

array[start:stop:step]

Slice elements from index start to stop-1 with a step size of step. Negative indices are interpreted as counting from the end of the array. Example: x[1:5:2] returns every other element from index 1 to 4 in array x.

array.real and array.imag

return the real and imaginary parts of a complex array. Example: x.real returns the real part of array x.

evals, evecs = np.linalg.eigh(a)

Computes the eigenvalues and eigenvectors of a real symmetric matrix. Example: np.linalg.eigh(a) computes eigenvalues and eigenvectors of a.

np.all(condition, [axis=])

Returns True if all elements of the input array satisfy the condition. Optionally, you can specify the axis along which to operate. Example: np.all(x > 0, axis=1) checks if all elements in each row of x are greater than 0.

np.allclose(a, b, [atol=, rtol=])

Returns True if all elements of two arrays are approximately equal. atol and rtol are optional parameters specifying the absolute and relative tolerances, respectively. Example: np.allclose(x, y, atol=1e-10, rtol=1e-6) checks if arrays x and y are approximately equal within the specified tolerances.

np.any(condition, [axis=])

Returns True if any element of the input array satisfies the condition. Optionally, you can specify the axis along which to operate. Example: np.any(x > 0, axis=0) checks if any element in each column of x is greater than 0.

np.argmax(x, [axis=])

Returns the indices of the maximum values along an axis. Optionally, you can specify the axis. Example: np.argmax(x, axis=0) gives the indices of the maximum values along each column.

np.argmin(x, [axis=])

Returns the indices of the minimum values along an axis. Optionally, you can specify the axis. Example: np.argmin(x, axis=1) gives the indices of the minimum values along each row.

np.argsort(x, [axis=])

Returns the indices that would sort an array. Optionally, you can specify the axis. Example: np.argsort(x) gives the indices that would sort each row of the array.

np.clip(x, a_min, a_max)

Clips (limits) the values in an array. Example: np.clip(x, 0, 1) clips the values in x to be between 0 and 1.

np.concatenate((a1, a2, ...), axis=)

Join a sequence of arrays along an existing axis. Example: np.concatenate((x, y), axis=0) concatenates arrays x and y along the first axis.

np.count_nonzero(x, [axis=])

Counts the number of non-zero values in the array. Optionally, you can specify the axis along which to operate. Example: np.count_nonzero(x, axis=0) counts the number of non-zero values in each column of x.

np.cov(x, [rowvar=False])

Estimates the covariance matrix of a dataset. x and y are the variables to be considered. rowvar specifies whether each row represents a variable (True) or a sample (False). Example: np.cov(x, rowvar=False) computes the covariance matrix of x arranged as columns.

np.cumprod(x, [axis=])

Returns the cumulative product of the elements along a given axis. Example: np.cumprod(x) returns the cumulative product of elements in array x.

np.cumsum(x, [axis=])

Returns the cumulative sum of the elements along a given axis. Example: np.cumsum(x) returns the cumulative sum of elements in array x.

np.diag(x)

Extracts the diagonal of a matrix *or* expands a vector into a diagonal matrix. Example: np.diag(x) returns the diagonal of matrix x if x is a matrix. np.diag(np.diag(x)) returns a matrix with the diagonal elements of x on the diagonal.

np.diff(x, [n=, axis=])

Calculates the n-th discrete difference along the given axis. Example: np.diff(x, n=2) returns the second discrete difference of elements in array x.

np.digitize(x, bins)

Returns the indices of the bins to which each value in input array belongs. Example: np.digitize(x, bins) returns the indices of the bins to which each value in x belongs.

np.einsum(subscripts, *operands)

Evaluates the Einstein summation convention on the operands. Example: np.einsum('ij -> ji', a) transposes a matrix a. Letters are used to label axes.

np.extract(condition, x)

Returns the elements of an array that satisfy a condition. Example: np.extract(x > 0, x) returns the elements of array x that are greater than 0. Same as x[x > 0].

np.flatten(x)

Returns a flattened array containing the same elements as the input array. Example: np.flatten(x) returns a flattened version of array x.

np.fft(x)

Computes the one-dimensional discrete Fourier Transform. Example: np.fft(x) computes the one-dimensional discrete Fourier Transform of array x.

np.fftfreq(x) and np.rfftfreq(x)

Computes the sample frequencies for a discrete Fourier Transform. Example: np.fftfreq(x) computes the sample frequencies for a discrete Fourier Transform of array x.

np.floor(x) or np.ceil(x)

Returns the floor or ceiling (nearest whole number lesser/greater) of each element in the input array. Example: np.floor(x) returns the floor of each element in array x.

np.full(shape, fill_value, [dtype=])

Returns a new array of given shape and type, filled with a constant value. Example: np.full((3, 3), 5.0) creates a 3x3 array filled with the value 5.0.

np.gradient(x, [edge_order=])

Returns the gradient of an N-dimensional array. edge_order specifies the order of the edge used to calculate the gradient. Example: np.gradient(x, edge_order=2) returns the gradient of array x using a second-order edge.

np.histogram(x, [bins=])

Computes the histogram of a set of data. bins specifies the number of bins to use. Example: np.histogram(x, bins=10) computes the histogram of array x using 10 bins.

np.ifft(x)

Computes the one-dimensional inverse discrete Fourier Transform. Example: np.ifft(x) computes the one-dimensional inverse discrete Fourier Transform of array x.

np.is[finite, inf, nan](x)

Returns a boolean array indicating which elements are finite (not infinity or NaN), infinite, or Nan. Example: np.isfinite(x) returns a boolean array indicating which elements of x are finite. np.isnan(x) returns a boolean array indicating which elements of x are NaN.

np.linspace(start, stop, steps)

Returns evenly spaced numbers over a specified interval. Example: np.linspace(0, 1, 11) returns 11 evenly spaced numbers between 0 and 1.

np.linalg.cond(a, [p=])

Computes the condition number of a matrix. p is an optional parameter specifying the norm used in the calculation (default is 2-norm). Example: np.linalg.cond(a, p=2) computes the 2-norm condition number of matrix a.

np.linalg.det(a)

Computes the determinant of a square matrix. Example: np.linalg.det(a) computes the determinant of matrix a.

np.linalg.inv(a)

Computes the multiplicative inverse of a matrix. Example: np.linalg.inv(a) computes the inverse of matrix a if it exists. Raises an error if the matrix is singular or not square.

np.linalg.norm(x, [ord=, axis=])

Computes the norm of a vector or matrix. ord specifies the type of norm to compute (default is L2 norm). axis specifies the axis that is reduced over. Example: np.linalg.norm(x, ord=2, axis=1) computes L2 norms of rows in a matrix.

np.linalg.pinv(a)

Computes the (Moore-Penrose) pseudo-inverse of a matrix. rcond is an optional parameter specifying the singular values below which singular values are considered zero. Example: np.linalg.pinv(a, rcond=1e-15) computes the pseudo-inverse of matrix a with a specified tolerance for singular values.

np.loadtxt(fname)

Loads data from a text file. fname is the name of the file to load. Example: np.loadtxt('data.txt') loads data from the file data.txt.

np.max(x, [axis=])

Returns the maximum value along an axis. Optionally, you can specify the axis. Example: np.max(x, axis=0) computes the maximum values along each column.

np.mean(x, [axis=])

Reduction; computes the mean over an array. Optionally only reduces over the specified axes. Example: np.mean(x, axis=1)

np.meshgrid(*xi, **kwargs)

Returns coordinate matrices from coordinate vectors. Example: X, Y = np.meshgrid(x, y) creates 2D grids of coordinates X and Y from 1D arrays x and y, respectively.

np.min(x, [axis=])

Returns the minimum value along an axis. Optionally, you can specify the axis. Example: np.min(x, axis=1) computes the minimum values along each row.

np.nan[min, max, sum, prod, mean, cumsum, cumprod, argmin, argmax, std, median]

Returns the minimum, maximum, etc. of an array ignoring NaN values. Example: np.nanmean(x) returns the mean of array x ignoring NaN values.

np.nextafter(x1, x2)

Returns the next floating-point value after x1 towards x2. Example: np.nextafter(1.0, 2.0) returns the next floating-point value after 1.0 towards 2.0.

np.nonzero(a)

Return the indices of the elements that are non-zero. Example: np.nonzero(x) returns a tuple of arrays representing the indices of non-zero elements in x.

np.ones(shape, [dtype=])

Returns a new array of given shape and type, filled with ones. Example: np.ones((3, 2), dtype=float) creates a 3x2 array filled with ones.

np.ravel(x)

Returns a flattened array containing the same elements as the input array. Example: np.ravel(x) returns a flattened version of array x.

np.repeat(a, repeats, [axis=])

Repeats elements of an array. The repeats argument specifies the number of repetitions for each element. Optionally, you can specify the axis along which to repeat values. Example: np.repeat(x, 3, axis=1) repeats each column of x three times.

np.reshape(x, newshape)

Gives a new shape to an array without changing its data. Example: np.reshape(x, (2, 3)) reshapes array x to a 2x3 matrix. Cannot change the total number of elements.

np.rfft(x) and np.irfft(x)

Computes the one-dimensional discrete Fourier Transform for real input. Example: np.rfft(x) computes the one-dimensional discrete Fourier Transform of real array x.

np.savetxt(fname, x)

Saves an array to a text file. fname is the name of the file to save to. Example: np.savetxt('data.txt', x) saves array x to the file data.txt.

np.searchsorted(a, v, [side=])

Finds indices where elements should be inserted to maintain order. a is the input array and v is the values to insert. side specifies whether to return the first index (left) or last index (right) if the value is repeated. Example: np.searchsorted(a, v, side='left') returns the indices where values v should be inserted into array a to maintain order.

np.seterr()

change trapping of floating point exceptions. Example: np.seterr(divide='ignore') ignores division by zero errors.

np.sign(x)

Returns an array of the same shape as x with elements indicating the sign of each element in the input array. Positive elements are represented as 1, negative elements as -1, and zeros as 0. Example: np.sign(x) returns an array with the signs of elements in x.

np.sort(x, [axis=])

Returns a sorted copy of an array. Optionally, you can specify the axis. Example: np.sort(x, axis=0) sorts each column of the array.

np.squeeze(x, [axis=])

Removes single-dimensional entries from the shape of an array. Optionally, you can specify the axis to remove particular dimensions. Example: np.squeeze(x, axis=1) removes the second dimension if it has size 1.

np.stack(arrays, axis=)

Join arrays *of the same shapealong a new axis. Example: np.stack((x, y)) stacks arrays x and y and adds a new dimension at the front.

np.std(x, [axis=])

Computes the standard deviation along the specified axis. Example: np.std(x, axis=0) computes the standard deviation of each column of x.

np.sum(x, [axis=])

Reduction; computes the sum over an array. Optionally only reduces over the specified axes. Example: np.sum(x, axis=0)

np.swapaxes(x, axis1, axis2)

Interchanges two axes of an array. Example: np.swapaxes(x, 0, 1) swaps the first and second axes of array x.

np.tile(x, reps)

Construct an array by repeating x the number of times given by reps. Example: np.tile(x, (2, 3)) creates a 2x3 tiling of x.

np.trace(a, [offset=, axis1=, axis2=])

Computes the sum along diagonals of a square matrix. offset is an optional parameter specifying the diagonal in question (0 for the main diagonal). axis1 and axis2 specify the axes along which to compute the trace for higher-dimensional arrays. Example: np.trace(a, offset=1) computes the sum of elements above the main diagonal of the matrix a.

np.transpose(x)

Reverses the dimensions of an array

np.unravel_index(indices, shape)

Converts a flat index or array of flat indices into a tuple of coordinate arrays. Example: np.unravel_index(22, (3, 4)) converts index 22 in a 3x4 array to coordinates (5, 2). Useful for converting argmin/argmax results to coordinates.

np.where(condition, [x=, y=])

Return elements chosen from x or y depending on the condition. If only condition is provided, it returns the indices where condition is True. If x and y are provided, it returns elements from x where condition is True, and from y where condition is False. Example: np.where(x > 0, x, 0) replaces negative elements in x with 0.

np.zeros(shape, [dtype=])

Returns a new array of given shape and type, filled with zeros. Example: np.zeros((2, 3), dtype=int) creates a 2x3 array filled with zeros.

rng = np.random.default_rng()

Creates a random number generator. Example: rng = np.random.default_rng() creates a random number generator rng.

rng.normal(loc=, scale=, size=)

Returns an array of random samples from a normal distribution. loc specifies the mean, scale specifies the standard deviation, and size specifies the shape of the output array. Example: rng.normal(loc=0.0, scale=1.0, size=(2, 3)) returns a 2x3 array of samples from a standard normal distribution.

rng.permutation(x)

Randomly permutes a sequence, or returns a permuted range. Example: rng.permutation(x) returns a randomly permuted version of array x.

rng.randint(low=, high=, size=)

Returns an array of random integers from low (inclusive) to high (exclusive). size specifies the shape of the output array. Example: rng.randint(low=0, high=10, size=(2, 3)) returns a 2x3 array of random integers from 0 to 9.

rng.uniform(low=, high=, size=)

Returns an array of random samples from a uniform distribution. low specifies the lower bound, high specifies the upper bound, and size specifies the shape of the output array. Example: rng.uniform(low=-1.0, high=1.0, size=(2, 3)) returns a 2x3 array of samples from a uniform distribution over [-1, 1).

u, s, vt = np.linalg.svd(a)

Computes the singular value decomposition of a matrix. Example: np.linalg.svd(a) returns the singular value decomposition of matrix a.