

PyAIML Notes V3

With **Python 3.x** implementations



Rohnak Agarwal

-  : rrka79wal@gmail.com
-  : +91-9658600961
-  : +91-9658600961/ +91-7978928490

PREFACE & DISCLAIMER

1. The minimum level of implementation uses NumPy and Python's built-in libraries, rather than building everything from scratch. Such implementations are commonly found in Python textbooks and instructional resources.
2. Some algorithms are implemented at a slightly lower level than highly optimized external libraries (e.g., PyTorch, TensorFlow) to help illustrate the underlying mechanics and core ideas more transparently.
3. All necessary citations have been provided to the best of our knowledge. Any missing references are unintentional and will be promptly addressed upon notification at rka79wal@gmail.com.
4. The wordings/ phrasing of the citations may have been improvised to fit the context/ time of this book.

- Rohnak Agarwal

SUBJECT LIST

(According to BITS Pilani Syllabus)

STATUS

Semester 1

1. [ACI] Artificial and Computational Intelligence

- (a) ARTIFICIAL INTELLIGENCE: BOOK: Artificial Intelligence: A Modern Approach 3e
SERIES: Prentice Hall Series
AUTHOR(S): Stuart Russell and Peter Norvig
PUBLISHER: Pearson
YEAR: 2010
URL: <https://aima.cs.berkeley.edu/> WIP (155/1153) CH (4/27)

2. [ISM] Introduction to Statistical Methods

- (a) STATISTICS: BOOK: Statistics for Data Scientists: An Introduction to Probability, Statistics, and Data Analysis
SERIES: Undergraduate Topics in Computer Science
AUTHOR(S): Maurits Kaptein and Edwin van den Heuvel
PUBLISHER: Springer
YEAR: 2022
URL: <https://doi.org/10.1007/978-3-030-10531-0> WIP (262/341) CH (7/8)

3. [MFML] Mathematical Foundations for Machine Learning

- (a) MATHS FOR ML: BOOK: Mathematics For Machine Learning
AUTHOR(S): Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong
PUBLISHER: Cambridge University Press
YEAR: 2020
URL: <https://mml-book.com/> WIP (187/417) CH (6/12)

4. [ML] Machine Learning

- (a) MACHINE LEARNING: BOOK: Pattern Recognition And Machine Learning
SERIES: Information Science and Statistics
AUTHOR(S): Christopher M. Bishop
PUBLISHER: SPRINGER NP EXCLUSIVE (CBS)
YEAR: 2009
URL: <https://www.amazon.in/Pattern-Recognition-Machine-Learning-2009/dp/1493938436> WIP (42/758) CH (1/14)

Semester 2

5. [CYB] AI and ML techniques for Cyber Security
6. [DMML] Data Management for Machine Learning
7. [DNN] Deep Neural Networks
8. [DRL] Deep Reinforcement Learning

9. [FATML] Fair, Accountable, Transparent Machine Learning
10. [IR] Information Retrieval
11. [MLSO] ML System Optimization
12. [NLP] Natural Language Processing
13. [PGM] Probabilistic Graphical Models

Semester 3

14. [ADL] Advanced Deep learning
15. [CAI] Conversational AI
16. [CV] Computer Vision
17. [DML] Distributed Machine Learning
18. [GNN] Graph Neural Networks
19. [MLOps] ML Operations
20. [NLPA] NLP Applications
21. [SMA] Social Media Analytics
22. [VA] Video Analytics

CONTENTS

I FOUNDATION

1	Basic Functions	3
1.1	Absolute Value function/ Modulus Function ($ x $)	3
1.2	Exponential Function ($\exp\{x\}$, $\exp(x)$)	4
1.3	Sign Function ($\text{sgn}(x)$)	4

2	Advanced Functions	5
----------	---------------------------	---

2.1	Gauss error function ($\text{erf}(z)$)	5
2.2	Gamma Function $\Gamma(x)$	5

II DATA

3	Data	9
----------	-------------	---

3.1	Measurement Levels [56]	9
3.1.1	Nominal, Ordinal, Interval and Ratio [56]	9
3.1.2	Continuous vs Discrete numerical data [56]	9
3.1.3	Outliers [56]	9
3.1.4	Unrealistic Values [56]	10
3.2	Describing Data [56]	10
3.2.1	Frequency/ Frequency table [56]	10
3.2.1.1	(Absolute) Frequency/ (Absolute) Frequency table [56]	11
3.2.1.2	Cumulative Frequency/ Cumulative Frequency table [56]	11
3.2.1.3	Relative Frequency/ Relative Frequency table [56]	12
3.2.1.4	Cumulative Relative Frequency/ Cumulative Relative Frequency table [56]	12
3.2.2	Central Tendency [56]	13
3.2.2.1	(Arithmetic) mean/ average (TODO / μ) [56]	13
3.2.2.2	Mode [56]	13
3.2.2.3	Median [56]	13
3.2.2.4	Quantiles (q_i) [56]	14
3.2.2.5	Quartiles (q_i) [56]	14
3.2.2.6	Deciles (q_i) [56]	15
3.2.2.7	Percentiles (q_i) [56]	15
3.2.2.8	Range [56]	15
3.2.2.9	Interquartile Range (IQR) [56]	15
3.2.2.10	Mean Absolute Deviation (MAD) [56]	16
3.2.2.11	Mean Squared Deviation (MSD) [56]	16
3.2.2.12	Variance (s^2 / σ^2) [56]	16
3.2.2.13	Standard Deviation (s / σ) [56]	17
3.2.2.14	Skewness (g_1) [56]	17

3.2.2.15	Kurtosis (g_2) [56]	18
3.2.3	Aggregated Data [56]	19
3.2.3.1	Mean [56]	19
3.2.3.2	Variance [56]	20

4 Visualizing Data

4.1	Box and Whiskers Plot/ Box Plot [14, 16] -----	22
4.2	Count Plot [17] -----	23
4.3	Density Plot [18] -----	23
4.4	Histogram [19] -----	24
4.5	Pairwise Plot [20, 56] -----	24
4.6	Pie Chart [15] -----	25
4.7	Scatter Plot [21, 22] -----	26

III MATHEMATICS: STATISTICS

5 Sampling Plans

5.1	Generic Formulation [56] -----	30
5.2	Measures of closeness -----	33
5.2.1	Bias	33
5.2.2	Mean Square Error (MSE)	33
5.2.3	Root Mean Square Error (RMSE)	34
5.2.4	Standard Error (SE)	34
5.3	Types of Samplings -----	34
5.4	Convenience Sampling [56] -----	36
5.5	Haphazard Sampling [56] -----	36
5.6	Purposive Sampling/ Judgmental Sampling [56] -----	36
5.7	Simple Random Sampling [56] -----	37
5.7.1	Estimation for population mean	37
5.7.2	Estimation of the sample MSE	38
5.7.3	Sample Statistics (T_n)	38
5.8	Systematic Sampling [56] -----	39
5.8.1	Estimation for population mean	40
5.8.2	Estimation of the MSE	40
5.9	Stratified Sampling [56] -----	41
5.9.1	population stratum	42
5.9.2	population	42
5.9.2.1	Estimation for population mean	42
5.9.3	sample stratum	42
5.9.3.1	Estimation for sample stratum SE	43
5.9.4	sample	43
5.9.4.1	Estimation of the sample MSE	43
5.9.4.2	Estimation of the sample SE	43

5.10 Cluster Sampling [56]	44
5.10.1 Single-Stage Cluster Sampling	44
5.10.1.1 cluster sizes are not all equal	44
5.10.1.2 cluster sizes are all equal	44
5.10.2 Two-Stage Cluster Sampling	45
5.10.2.1 cluster sizes are not all equal	45
5.10.2.2 cluster sizes are all equal	45
5.10.3 Multi-Stage Cluster Sampling	45
5.11 Cross-Sectional Study (population-based)	46
5.12 Cohort Study (exposure-based)	46
5.13 Case-Control Study (disease-based)	46
5.14 Important Notes	48
5.14.1 Haphazard Sampling VS Random Sampling [7]	48
6 Probability Theory	49
6.1 Definitions	49
6.2 Event Axioms	50
6.3 Event Rules	51
6.4 Conditional Probability ($P(A B)$)	51
6.5 Bayesian Probabilities/ Bayesian Statistics	52
6.5.1 Bayes' Law for Statistical Models	54
6.5.2 The Fundamentals of Bayesian Data Analysis	55
6.5.3 Bayesian Decision-Making in Uncertainty	56
6.5.3.1 Providing Point Estimates of Parameters	56
6.5.4 Frequentist AND/VS Bayesian	56
6.6 Expectations and covariances	57
6.7 Bootstrap	57
6.8 Association Measures/ Measures of Risk	57
6.8.1 Risk Difference/ Excess Risk ($ER = P(D E) - P(D E^c)$)	57
6.8.2 Relative Risk ($RR = P(D E)/P(D E^c)$)	58
6.8.3 Odds Ratio ($OR = (P(D^c E^c)/P(D^c E)) \times RR$)	58
6.8.4 Relation in ER, RR & OR	59
6.9 Simpson's Paradox	59
7 Random Variables & Distributions	63
7.1 Intro	63
7.2 Joint Distribution Function ($F_{XY}, F_{X_1 X_2 \dots X_K}$)	64
7.2.1 Independence of Random Variables	64
7.3 Discrete Functions	65
7.3.1 Univariate PMF ($f(x_k) = p_k = P(X = x_k)$)	65
7.3.2 CDF of Univariate PMF ($F(x)$)	65
7.3.3 Bivariate Joint PMF ($f_{XY}(x,y) = P(X = x, Y = y)$)	66
7.3.4 CDF of Bivariate Joint PMF (F_{XY})	66

7.4	Continuous Functions	66
7.4.1	Univariate PDF ($f(x) \neq P(X = x)$)	66
7.4.2	CDF of Univariate PDF ($F(X)$)	67
7.4.3	Bivariate Joint PDF	67
7.4.4	CDF of Bivariate Joint PDF (F_{XY})	67
7.5	Sufficient Statistics	68
7.6	Sample Statistics (T_n), Estimators & Summary Statistics	68
7.6.1	Distributions of Sample Statistic T_n	69
7.6.1.1	Distribution of the Sample Minimum	70
7.6.1.2	Distribution of the Sample Maximum	70
7.6.1.3	Distribution of the Sample Average	70
7.6.1.4	Distribution of the Sample Variance	71
7.6.2	Central Limit Theorem (CLT)	72
7.6.2.1	Central Limit Theorem Applied to Variances	72
7.6.3	Asymptotic Confidence Intervals	73
7.6.4	Methods of Estimation: Method of Moments Estimation (MME)	73
7.6.5	Methods of Estimation: Maximum Likelihood Estimation (MLE)	74
7.6.5.1	Standard Error of MLE	74
7.7	Expected Values ($E(X)$)	75
7.7.1	For PDF (Continuous)	76
7.7.2	For PMF (Discrete)	76
7.7.3	Common for PDF & PMF	76
7.8	Calculation Rules for Random Variables	77
7.8.1	Univariate/ Joint PMF/ PDF/ CDF	77
7.8.1.1	if X and Y are independent	79
7.8.2	Conditonal PMF, PDF & CDF	79
7.8.2.1	If X & Y are independent	79
7.9	Constructing Bivariate Distributions	80
7.9.1	Sums of Random Variables	80
7.9.2	Farlie-Gumbel-Morgenstern (FGM) Family of Distributions	80
7.9.3	Fréchet Family of Distributions	81
7.9.4	Mixtures of Probability Distributions	81
7.9.4.1	Conditionally independent	81
7.10	Measures of Association	82
7.10.1	Pearson's Correlation Coefficient ($CORR(X, Y) = \rho_P$)	82
7.10.1.1	Pearson's Correlation Coefficient estimator (r_P)	83
7.10.2	Kendall's Tau Correlation (concordance) (τ_K)	84
7.10.2.1	Kendall's Tau Correlation estimator (r_K)	84
7.10.3	Spearman's Rho Correlation (ρ_S)	85
7.10.3.1	Spearman's Rho Correlation estimator (r_S)	85
7.10.4	Cohen's Kappa Statistic (κ_C)	86
7.10.4.1	Cohen's Kappa Statistic estimator (hat κ_C)	87

7.10.5	Risk Difference, Relative Risk, and Odds Ratio	88
7.10.5.1	Risk Difference	88
7.10.5.2	Relative Risk	88
7.10.5.3	Odds Ratio	89
7.10.6	Comparison: Pearson's Rho (ρ_P), Spearman's Rho (ρ_S), Kendall's Tau Correlation (τ_K), Cohen's kappa (κ_C)	89
7.10.7	Nominal Association Statistics	89
7.10.7.1	Cohen's kappa statistic	90
7.10.7.2	Pearson's chi-square statistic (χ_P^2)	90
7.10.7.3	Pearson's squared phi-coefficient (ϕ^2)	91
7.10.7.4	Cramér's V (V)	91
7.10.8	Ordinal Association Statistics	91
7.10.8.1	Goodman and Kruskal's gamma (γ)	91
7.10.9	Binary Association Statistics	92
7.10.9.1	Gower and Legendre (S_θ, T_θ)	92
7.10.9.2	Pearson's squared phi-coefficient (ϕ^2)	93
7.10.9.3	L family of similarity indices	93
7.10.9.4	Yule's Q statistic (Q)	93
7.11	Change of Variables	94

8 Discrete Distributions

8.1	Bernoulli distribution ($X \sim B(p)$)	95
8.1.1	Distribution of the Sample Statistic (T_n)	95
8.1.1.1	Distribution of the Sample Average	95
8.1.2	Maximum Likelihood Estimation (MLE)	95
8.1.3	Estimating the Parameter using Bayesian Methods	96
8.1.4	Bernoulli as Exponential Family	97
8.1.5	Summary	97
8.2	Binomial distribution ($X \sim B(n, p)$)	99
8.2.1	PMF	99
8.2.2	Summary	99
8.3	Multinomial distribution	100
8.3.1	Summary	100
8.4	Negative binomial distribution ($NB(r, p)$)	101
8.4.1	PMF	101
8.4.2	Summary	101
8.5	Poisson distribution	103
8.5.1	Summary	103
8.5.2	Standard Error of MLE	104
8.5.3	Sums of Random Variables	104

9 Continuous Distributions

9.1	Normal Distribution/ Gaussian Distribution ($N(\mu, \sigma^2), N(x \mu, \sigma^2)$)	105
9.1.1	PDF ($f_{\mu, \sigma}(x)$ or $f(x \mu, \sigma)$)	105
9.1.2	Maximum Likelihood	106

9.1.3	Normally Distributed Populations	106
9.1.3.1	Confidence Intervals for Normal Populations	106
9.1.4	Distribution of the Sample Statistic (T_n)	107
9.1.4.1	Distribution of the Sample Average	107
9.1.5	Methods of Moments Estimation (MME)	108
9.1.6	Sums of Random Variables	108
9.1.7	The t-Test for a Single Sample ($H_0 : \mu(f) \leq \mu_0$)	108
9.1.8	The t-Test for Two Independent Samples ($H_0 : \mu_1 = \mu_2$)	109
9.1.9	The t-Test for Two Dependent Samples ($H_0 : \mu_D = \mu_1 - \mu_2 \leq 0$)	110
9.1.10	The F-Test for Equal Variances ($H_0 : \sigma_1 = \sigma_2$)	110
9.1.11	Tests for Normality	111
9.1.11.1	Graphical approach (g-g Plot)	111
9.1.11.2	Shapiro-Wilk test	111
9.1.12	Grubbs Test for Outliers/ extreme studentized deviate test/ maximum normed residual test	112
9.1.13	Tukey's Method for Outliers (Using IQR)	112
9.1.14	Estimating the Parameters using Bayesian Methods	112
9.1.14.1	Bayesian Analysis for Normal Populations Based on Single Observation	113
9.1.14.2	Bayesian Analysis for Normal Populations Based on Multiple Observations	114
9.1.14.3	Bayesian Analysis for Normal Populations with Unknown Mean and Variance	114
9.1.15	Gaussian as Exponential Family	115
9.1.16	Summary	115
9.2	Multivariate normal distribution ($N(\mu, \Sigma)$)	117
9.2.1	PDF	117
9.2.2	Bivariate Normal Distributions	117
9.2.3	Marginals and Conditionals	119
9.2.4	Product of Gaussian Densities	119
9.2.5	Farlie-Gumbel-Morgenstern (FGM) Family of Distributions	119
9.2.6	Mixtures of Probability Distributions	119
9.2.6.1	Conditionally independent	120
9.2.7	Correlation Tests for Numerical Variables ($H_0 : \rho_P = 0$)	120
9.2.8	Sampling from Multivariate Gaussian Distributions	121
9.2.9	Summary	122
9.3	Standard Normal Distribution ($N(0, 1)$)	123
9.3.1	PDF ($f(x)$ or $f(x)$ or $\phi(x)$)	123
9.3.2	Bivariate Standard Normal Distribution	123
9.4	Lognormal Distribution (Lognormal(μ, σ^2), $LN(\mu, \sigma^2)$)	124
9.4.1	PDF	124
9.4.2	Bivariate Lognormal Distributions	124
9.4.3	Summary	125
9.4.4	Lognormally Distributed Populations	126
9.4.5	Method of Moments Estimation (MME)	126
9.4.6	Maximum Likelihood Estimation (MLE)	126
9.4.7	Tukey's Method for Outliers (Using IQR)	127
9.5	(Continuous) Uniform Distribution ($U(a, b)$)	128
9.5.1	PDF	128
9.5.2	Summary	128

9.6	Standard (Continuous) Uniform Distribution ($U(0, 1)$)	129
9.6.1	PDF	129
9.7	Exponential Distribution (Exponential(λ))	130
9.7.1	Distribution of the Sample Statistic (T_n)	130
9.7.1.1	Distribution of the Sample Minimum ($T_n = F_{X_{(1)}}(t)$)	130
9.7.2	Farlie–Gumbel–Morgenstern (FGM) Family	130
9.7.3	Exponential Family	131
9.7.4	Summary	131
9.8	Laplace distribution/ Double Exponential Distribution (Laplace(μ, b))	133
9.8.1	PDF	133
9.8.2	Summary	133
9.9	Chi-squared distribution ($\chi^2(k)$ OR χ_k^2)	135
9.9.1	Summary	135
9.10	Student's t-distribution (t_v)	136
9.10.1	Summary	136
9.11	Beta distribution (Beta(α, β))	138
9.11.1	Summary	138
10	Relationships Between Distributions	141
10.1	Binomial-Poisson	141
10.2	Binomial-Normal	141
11	Mix Distributions	143
11.1	X, Y: Binary, Z: Std Norm Dist	143
11.1.1	Mixtures of Probability Distributions	143
11.1.1.1	Conditionally independent	143
11.2	X, Y: Exp/ Lognormal, Z: Poisson	143
11.2.1	Mixtures of Probability Distributions	143
11.2.1.1	Conditionally independent	144
11.3	Beta-Binomial Conjugacy	144
11.4	Beta-Bernoulli Conjugacy	144
12	Decisions in Uncertainty	147
12.1	Bootstrapping	147
12.1.1	Basic Idea of Bootstrap	147
12.1.1.1	Non-Parametric Bootstrap	148
12.1.1.2	Parametric Bootstrap	148
12.2	Hypothesis testing	148
12.2.1	The One-Sided z -Test for a Single Mean ($H_0 : \mu(f) \leq \mu_0$)	151
12.2.2	The Two-Sided z -Test for a Single Mean ($H_0 : \mu(f) = \mu_0$)	153
12.2.3	Non-Parametric Test: Mann–Whitney U Test for Two Independent Samples ($H_0 : P(Y_1 > Y_2) = P(Y_1 < Y_2)$)	154
12.2.4	The Sign Test for Two Related Samples ($H_0 : P(Y_1 > Y_2) = 0.5$)	155
12.2.5	Wilcoxon's Signed Rank Test for Two Related Samples ($H_0 : m_D = 0$)	155
12.2.6	Levene's Test for Equal Variation ($H_0 : \sigma_1^2 = \sigma_2^2 = \dots = \sigma_k^2$)	156

12.2.7	The χ^2 Test for Categorical Variables ($H_0 : P(X = x, Y = y) = P(X = x)P(Y = y)$)	157
12.2.8	Tests for Outliers	157
12.2.8.1	Tukey's Method for Outliers (Using IQR)	158
12.2.9	Equivalence Testing ($H_0 : \mu(f) - \mu_0 > \Delta$)	158
12.2.10	Bayes' Law for Competing Hypotheses	160

IV MATHEMATICS: LINEAR ALGEBRA

13	Group Theory	163
13.1	Groups	163
13.2	Abelian Group	163
13.3	General Linear Group	163
14	Matrices	165
14.1	Matrix Operations	165
14.1.1	Matrix Addition (+) [49]	165
14.1.2	Matrix Multiplication (AB OR @ OR ·) [49]	165
14.1.2.1	Properties	166
14.1.3	Multiplication by a Scalar (λA)	166
14.1.3.1	Properties	167
14.1.4	Hadamard product (\odot OR *) [49]	167
14.1.5	Matrix Transpose (A^\top) [49]	168
14.1.5.1	Properties	168
14.1.6	Matrix Inverse (A^{-1}) [49]	168
14.1.6.1	Matrix Inverse using Gaussian Elimination	169
14.1.6.2	Moore-Penrose pseudo-inverse ($(A^\top A)^{-1} A^\top$)	169
14.1.6.3	Properties	169
14.2	Rank of a matrix	169
14.2.1	Properties of rank	169
14.3	Null Space and Column Space	170
14.4	Inhomogeneous systems of linear equations and affine subspaces	170
14.5	Types of matrices	170
14.5.1	Square matrix	171
14.5.2	Identity Matrix (I_n)	171
14.5.3	Regular/ Invertible/ Non-singular matrix	171
14.5.4	Singular/ Non-invertible matrix	171
14.5.5	Symmetric Matrix	172
14.5.6	Symmetric, Positive Definite (SPD) Matrix	172
14.5.7	Equivalent Matrices	173
14.5.8	Similar Matrices	173
14.5.9	Orthogonal Matrix	173
14.5.10	Triangular matrix (T)	173
14.5.11	Diagonal Matrix (D)	173

14.6	Norms of a Matrix	174
14.7	Describing/summarizing Matrix	174
14.7.1	Determinant ($\det(A)$ or $ A $)	174
14.7.1.1	Properties of Determinant	176
14.7.2	Trace ($\text{tr}(A)$)	176
14.7.2.1	Properties of Trace	177
14.7.3	Characteristic Polynomial ($p_A(\lambda)$)	177
14.7.4	Eigenvalues (λ) & Eigenvectors	177
14.7.4.1	Properties of eigenvalues & eigenvectors	178
14.7.5	Eigenspace (E_λ) & Eigenspectrum	179
14.7.5.1	Properties of Eigenspace & Eigenspectrum	179
14.8	Matrix Decomposition/Factorization	179
14.8.1	Cholesky Decomposition ($A = LL^\top$)	179
14.8.2	Eigendecomposition ($A = PDP^{-1}$)	180
14.8.3	Singular Value Decomposition (SVD) ($A = U\Sigma V^\top$)	181
14.8.3.1	Geometric Intuitions for the SVD	183
14.8.3.2	Construction of the SVD	183
14.8.4	Eigenvalue Decomposition ($A = PDP^{-1}$) vs. Singular Value Decomposition ($A = U\Sigma V^\top$)	185
14.9	Matrix Approximation	185
15	Vector Spaces	189
15.1	Vector Space	189
15.1.1	Properties of a vector Space	189
15.1.2	Dimension of a vector Space	189
15.1.3	Norm	190
15.1.4	General Inner Product ($\Omega(a, b)$)	190
15.1.5	Inner Product ($\langle x, y \rangle$)	190
15.1.5.1	Inner Product Space	191
15.1.5.2	Inner Products of Random Variables	191
15.1.6	Euclidean vector space	191
15.1.7	Outer Product (\otimes)	191
15.1.8	Lengths	191
15.1.9	Distance	192
15.1.10	Metric	192
15.1.11	Inner Product VS Metric	192
15.1.12	Angles	192
15.1.13	Orthogonal Complement (U^\perp)	193
15.2	Vector Subspace/ linear subspace	193
15.2.1	Dimension of a Vector Subspace	194
15.2.2	Basis of a Vector Subspace	194
15.3	Vector	194
15.3.1	Types of vectors	194
15.3.2	Manhattan Norm/ ℓ_1 Norm	195
15.3.3	Euclidean Norm/ ℓ_2 Norm	195

15.3.4 Outer Product (ab^\top)	195
15.3.5 Scalar/dot product ($a^\top b$)	196
15.3.6 Orthogonal ($x \perp y$) & orthonormal vectors	196
15.3.7 Vector as a function	196
15.3.7.1 Inner Product of Functions	196
15.3.8 Collinearity and Codirection	197
15.4 Linear Combination	197
15.5 Affine Subspaces	197
15.6 Linear (In)dependence	198
15.6.1 Properties of Linear (In)dependence	198
15.7 Generating Set	199
15.8 Span	199
15.9 Basis	199
15.9.1 Checking Basis	200
15.9.2 Basis Change	201
15.9.3 Orthonormal Basis (ONB) & Orthogonal Basis (OGB)	202
15.9.3.1 Orthogonalization using Gaussian Elimination	202
15.9.3.2 Gram-Schmidt Orthogonalization	203
15.10 Linear Mappings (Φ)	206
15.10.1 Matrix Representation of Linear Mappings	206
15.10.2 Image/Range & Kernel/Null space	206
15.11 Affine Mapping	207
15.12 Injective, Surjective, Bijective Mappings	207
15.13 Special cases of Linear mappings	208
15.14 Transformation Matrix (A_Φ)	208
15.15 Coordinates	209
15.16 Projections (π)	209
15.16.1 Projection Matrix (P_π)	210
15.16.2 Projection onto One-Dimensional Subspaces (Lines)	210
15.16.3 Projection onto General Subspaces	212
15.16.4 Projection onto Affine Subspaces	215
15.17 Rotations (Φ) & Rotation Matrix ($R(\theta)$)	215
15.17.1 Rotations in R^2	216
15.17.2 Rotations in R^3	216
15.17.3 Rotations in n Dimensions (Givens Rotation)	217
16 Vector Calculus	219
16.1 Function ($f(x)$)	219
16.1.1 Differentiation of Uni-variate Functions	219
16.1.2 Differentiation of multi-variate Functions	219
16.1.2.1 Gradient	220
16.2 Gradients of Vector-Valued Functions ($J = \nabla_x f$)	220
16.3 Higher-Order Derivatives	221
16.4 Gradients of Matrices	221

16.5	Differentiation Rules	222
16.6	Backpropagation: Gradients in DNN	223
16.7	Computational Graph	224
16.7.1	Static Computational Graphs	224
16.7.2	Dynamic Computational Graphs	225
16.8	Automatic Differentiation	225
16.9	Taylor Series & Polynomial	226

17 Systems of Linear Equations

17.1	Types of Solutions	230
17.2	Finding Solutions using Matrix Inverse	230
17.3	Finding Solutions using Elementary Transformations	230
17.3.1	Row-Echelon Form (REF)	231
17.3.2	Reduced Row-Echelon Form (RREF)/ row-reduced echelon form/ row canonical form [49]	231
17.3.3	Particular Solution/ Special solution	231
17.3.4	Finding Solutions to $Ax = 0$: Minus-1 Trick [49]	231
17.3.5	General Solution (= Particular Solution + Solutions to $Ax = 0$)	232
17.4	Approximate Solution	232
17.5	Finding Solutions using stationary iterative methods - TODO	233
17.5.1	Richardson method - TODO	233
17.5.2	Jacobi method - TODO	233
17.5.3	Gauß-Seidel method - TODO	233
17.5.4	successive over-relaxation method - TODO	233
17.6	Finding Solutions using Krylov subspace methods - TODO	233
17.6.1	conjugate gradients - TODO	233
17.6.2	generalized minimal residual - TODO	233
17.6.3	biconjugate gradients - TODO	233

V ARTIFICIAL INTELLIGENCE (AI)

18 AI: Introduction

18.1	Approaches to AI	237
18.1.1	Acting humanly: The Turing Test approach	237
18.1.2	Thinking humanly: The cognitive modeling approach	237
18.1.3	Thinking rationally: The “laws of thought” approach	238
18.1.4	Acting rationally: The rational agent approach	238
18.2	AI: Disciplines	239
18.2.1	Philosophy	239
18.2.2	Mathematics	239
18.2.3	Economics	240
18.2.4	Neuroscience	240
18.2.5	Psychology	240
18.2.6	Computer engineering	240
18.2.7	Control theory and cybernetics	240
18.2.8	Linguistics	240
18.3	AI: History	241

19 AI: Agents	243
19.1 Task Environment/ Problem [8]	244
19.1.1 PEAS: Defining Problem	244
19.1.2 Properties of Task Environments [8]	246
19.1.2.1 Fully observable, partially observable, unobservable	246
19.1.2.2 Single agent, multi-agent	246
19.1.2.3 Deterministic, stochastic, uncertain, nondeterministic	246
19.1.2.4 Episodic, sequential	246
19.1.2.5 Static, dynamic, semidynamic	247
19.1.2.6 Discrete, continuous	247
19.1.2.7 Known, unknown	247
19.2 Agent State Representations	248
19.2.1 Atomic representation	248
19.2.2 Factored representation	249
19.2.3 Structured representation	249
20 AI: Agent Programs	251
20.1 Table Driven Agent	251
20.2 Simple Reflex Agent	252
20.3 Model-based reflex agents	253
20.4 (Model-based) Goal-based Agents	254
20.5 (Model-based Goal-based) Utility-based Agents	255
20.6 Learning agents	256
20.7 Problem-Solving Agent	257
20.7.1 Defining & Formulating Problem	258
20.7.2 Measuring problem-solving performance	260
20.8 Planning Agents	261
21 AI: Searching Solutions	263
21.1 Designing Search Node	263
21.2 General Algorithms & Implementations	265
21.2.1 Child-node	265
21.2.2 Tree Search	265
21.2.3 Graph search	266
21.3 Search Strategies/ Search Algorithms	267
21.3.1 Uninformed Search/ Blind Search	267
21.3.2 Informed Search/ Heuristic Search	267
21.3.3 Local Search	268
21.3.4 Online Search	269
22 AI: Algorithms	271
22.1 Intro	271
Uninformed Search/ Blind Search	274
22.2 Breadth-first search (BFS) [8]	274
22.3 Uniform-cost search (UCS) [8]	276

22.4	Depth-first search (DFS) [8]	278
22.5	Backtracking Search [8]	279
22.6	Depth-limited search (DLS) [8]	280
22.7	Iterative Deepening Search (IDS) [8]	282
22.8	Iterative Lengthening Search (ILS) [8]	283
22.9	Bidirectional search [8]	284
	Informed Search/ Heuristic Search	286
22.10	Best-first search (BestFS) [8]	286
22.11	Greedy best-first search (GBFS) [8]	286
22.12	A* Search [8]	288
22.13	Iterative-Deepening A* (IDA*) search [8]	290
22.14	Recursive best-first search (RBFS) [8]	291
22.15	Memory-Bounded A* (MA*) Search [8]	292
22.16	Simplified Memory-Bounded A* (SMA*) Search [8]	292
	Local Search	294
22.17	(Greedy) Hill Climbing Search [8]	294
22.18	Stochastic hill climbing search [8]	294
22.19	First-choice hill climbing search [8]	294
22.20	Random-restart hill climbing search [8]	295
22.21	Simulated annealing search [8]	295
22.22	Local beam search [8]	296
22.23	Stochastic beam search [8]	296
22.24	Genetic algorithm (GA) [8]	296
22.25	Local Search in Continuous Space	299
22.26	Searching with Non-deterministic Actions	300
22.27	Summary	301
23	AI: Examples	303
23.1	Vacuum Cleaner (Toy problem) [8]	303
23.1.1	As a table driven agent	303
23.1.2	As a simple reflex agent	304
23.1.3	As a problem solving agent	304
23.2	Automated Taxi Driver [8]	308
23.2.1	As a table driven reflex agent	308
23.2.2	As model-based reflex agents	308
23.2.3	As utility-based agents	308
23.2.4	As Learning Agent	308
23.3	Road trip in Romania [8]	309
23.4	8-Puzzle [8]	315
23.4.1	Comparing heuristic functions	315
23.4.1.1	A* [8]	315
23.4.1.2	Comparison [8]	315
23.4.1.3	Generating admissible heuristics from subproblems: Pattern databases	316

23.5	8-Queens problem	317
23.5.1	As a Problem Solving Agent	317
23.5.2	As a Hill Clibing Agent	317
23.5.3	Using Genetic Algorithm	318

VI MACHINE LEARNING (ML) & DEEP LEARNING (DL)

24	★ML Project: Core Components [7]	321
24.1	Problem Definition	321
24.2	Data	321
24.3	Model Architecture	321
24.4	Loss Function (Criterion)	321
24.5	Optimizer	321
24.6	Training Loop	322
24.7	Evaluation Metrics	322
24.8	Validation & Testing	322
24.9	Model Saving & Loading	322
24.10	Inference / Deployment	322
24.11	Experiment Tracking	323
24.12	Reproducibility & Documentation	323
25	★Machine Learning Paradigms [7]	325
26	★Optimizer	329
27	★Loss Function (Criterion)	331
	Regression related Loss	331
27.1	Sum of Errors (SE)	331
27.2	Sum of Absolute Errors (SAE)	331
27.3	Sum of Squared Errors (SSE)	331
27.4	Mean Absolute Error (MAE) / L1 Loss	331
27.5	Mean Square Error (MSE) / L2 Loss	331
27.6	Root Mean Square Error (RMSE) Loss	331
27.7	Huber Loss / Smooth Mean Absolute Error	331
27.8	Mean Bias Error (MBE)	331
	Classification related Loss	332
27.9	Binary Cross-Entropy Loss / Log Loss	332
27.10	Categorical Cross-Entropy Loss	332
27.11	Hinge Loss	332
27.12	Log Loss	332
	Regularization	333
27.13	Early Stopping	333
27.14	Dropout	333
27.15	Lasso/ L1 Regression	333

27.16 Ridge/ L2 Regression	333
27.17 Elastic Net/ L1-L2 Regression	333
28 Machine Learning: Introduction	335
28.1 Intro	335
28.2 Pre-processing: Feature Extraction	335
28.3 Training/ Learning Phase	335
28.4 Testing/ Evaluation Phase	335
28.5 Supervised learning	336
28.5.1 Regression	336
28.5.2 Classification	336
28.6 Unsupervised learning	336
28.6.1 Clustering	336
28.6.2 Density Estimation	336
28.6.3 Visualization	336
28.7 Reinforcement Learning	336
28.8 Model Comparison OR Model Selection	337
28.8.1 Grid Search	337
28.8.2 Random Search	337
28.8.3 Bayesian Optimization	337
28.8.4 Cross-Validation Based Selection	338
28.8.4.1 Holdout Validation	338
28.8.4.2 LOOCV (Leave One Out Cross Validation)	338
28.8.4.3 Stratified Cross-Validation	338
28.8.4.4 K-Fold Cross Validation	339
28.9 Bias-Variance Trade Off	339
28.9.0.1 Bias	339
28.9.0.2 Variance	339
29 Generative AI (GenAI)	341
29.1 How Generative AI Works?	341
29.1.1 Core Mechanism (Training & Inference)	341
29.1.2 By Media Type	341
29.1.3 Agents in Generative AI	342
29.1.4 Training and Fine-Tuning	342
29.1.5 Retrieval-Augmented Generation (RAG)	342
29.2 Types of Generative AI Models	342
29.2.1 Transformers or Autoregressive Models	342
29.2.2 Diffusion Models	343
29.2.3 Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs)	343
29.2.4 Encoder Decoder Models	343
29.3 Evaluation of Generative AI	343
29.4 Applications of Generative AI	344
29.5 Advantages	344
29.6 Disadvantages	344

30	Agentic AI	347
30.1	How agentic AI works	347
30.2	Advantages	348
30.3	Challenges	349
31	Encoder-Decoder Architecture	351
VII SUPERVISED LEARNING		
32	Regression	355
32.1	Polynomial Curve Fitting	355
VIII RECURRENT NEURAL NETWORK (RNN)		
33	Recurrent Neural Networks (RNN) (1982)	359
33.1	Key Components of RNNs	359
33.2	How does RNN work?	360
33.3	Types Of Recurrent Neural Networks	362
33.4	Variants of Recurrent Neural Networks (RNNs)	363
33.5	Advantages	363
33.6	Disadvantages	363
33.7	What fixes it	364
34	Bi-directional RNN (BRNN/ BiRNN) (1997)	365
34.1	Working of Bidirectional Recurrent Neural Networks	365
34.2	Advantages	366
34.3	Disadvantages	367
34.4	What fixes it	367
35	Long Short-Term Memory (LSTM) (1997)	369
35.1	Working of LSTM	369
35.1.1	Forget Gate	371
35.1.2	Input gate	371
35.1.3	Output gate	372
35.2	Advantages	372
35.3	Disadvantages	372
35.4	What fixes it	372
36	Gated Recurrent Units (GRUs) (2014)	373
36.1	Working of GRU	373
36.2	Advantages	375
36.3	Disadvantages	375
36.4	What fixes it	375

IX TIMELINED ML-DL TECHNOLOGIES

37	Connectionist Temporal Classification (CTC) (2006)	379
38	Transducer (2012)	381
38.1	RNN Transducer	381
38.1.1	Prediction Network (G)	382
38.1.2	Transcription Network (F)	382
38.1.3	Output Distribution	383
38.2	LSTM Transducer	383
38.2.1	Prediction Network (G)	383
38.3	Forward-Backward Algorithm	384
38.4	Training	384
38.5	Testing/ Inference	385
39	Attention Mechanism (2014)	387
39.1	Intro: Encoder-decoder & Seq2seq	387
39.2	Probabilistic Approach	388
39.2.1	RNN Encoder-Decoder	388
39.3	Learning (RNN Encoder-Decoder)	389
39.4	How Attention Mechanism Works?	390
39.5	Self-Attention	391
39.6	Scaled Dot-Product Attention (Attention(Q, K, V))	392
39.7	Multi-Head Attention (MultiHead(Q, K, V))	393
40	Transformer (2017)	395
40.1	Model Architecture (encoder-decoder)	395
40.1.1	Encoder and Decoder Stacks	396
40.2	Applications of Attention in Transformer	397
40.3	Position-wise Feed-Forward Networks	397
40.4	Embeddings and Softmax	397
40.5	Positional Encoding	397

X APPLICATIONS

41	Automatic Speech Recognition	401
-----------	-------------------------------------	-----

XI APPENDIX

42	Datasets	405
-----------	-----------------	-----

LIST OF FIGURES

1.1	Absolute Value function/ Modulus Function [11]	3
4.1	Box and Whiskers plot (py-plt) output (face_data.csv)	22
4.2	Box and Whiskers plot (py-sns) output (face_data.csv)	22
4.3	Count plot (py-sns) output (face_data.csv)	23
4.4	Density plot (py-sns) output (face_data.csv)	23
4.5	Histogram (py-sns) output (face_data.csv)	24
4.6	Pairwise plot (py-sns) output (face_data.csv)	24
4.7	Pie chart (py-plt) output (face_data.csv)	25
4.8	Scatter Plot (py-plt) output (face_data.csv)	26
4.9	Scatter Plot (py-sns) output (face_data.csv)	26
5.1	Sampling Plan: Relation between Population and Sample	29
5.2	Visual Representation of Bias, MSE and SE	33
8.1	Binomial Distribution: PDF [25]	99
8.2	Binomial Distribution: CDF [25]	99
8.3	Poisson distribution: PDF [35]	103
8.4	Poisson distribution: CDF [35]	103
9.1	Normal Distribution: PDF [34]	105
9.2	Normal Distribution: CDF [34]	105
9.3	Multivariate normal distribution: PDF [32]	117
9.4	Lognormal Distribution: PDF [30]	124
9.5	Lognormal Distribution: CDF [30]	124
9.6	Uniform Distribution: PDF [27]	128
9.7	Uniform Distribution: CDF [27]	128
9.8	Exponential Distribution: PDF [28]	130
9.9	Exponential Distribution: CDF [28]	130
9.10	Double Exponential/ Laplace Distribution: PDF [29]	133
9.11	Double Exponential/ Laplace Distribution: CDF [29]	133
9.12	Chi-squared Distribution: PDF [26]	135
9.13	Chi-squared Distribution: CDF [26]	135
9.14	Student's t-distribution: PDF [36]	136
9.15	Student's t-distribution: CDF [36]	136
9.16	Beta Distribution: PDF [24]	138
9.17	Beta Distribution: CDF [24]	138
12.1	Types of errors in hypothesis testing [56]	149
12.2	Choosing a Statistical Test: Decision Tree [57]	150
15.1	Rotation of the standard basis in \mathbb{R}^2 by an angle θ . [49]	216
15.2	Rotation of a vector (gray) in \mathbb{R}^3 by an angle θ about the e_3 -axis. The rotated vector is shown in blue. [49]	217
22.1	Genetic Algorithm - flowchart	296
23.1	A vacuum-cleaner world with just two locations	303
23.2	The state space for the vacuum world. Links denote actions: L = Left, R = Right, S= Suck. [8]	303
23.3	A simplified road map of part of Romania. [8]	309

23.4 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines. [8]	309
23.5 Nodes inside a given contour have f-costs less than or equal to the contour value. [8]	314
23.6 A typical instance of the 8-puzzle. The solution is 26 steps long. [8]	315
23.7 A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles. [8]	315
23.8 Almost a solution to the 8-queens problem. the queen in the rightmost column is attacked by the queen at the top left.	317
23.9 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. [8] (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost. [8]	317
23.10 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e). [8]	318
23.11 The 8-queens states corresponding to the first two parents (c) and the first offspring (d). The shaded columns are lost in the crossover step and the unshaded columns are retained. [8]	318
33.1 Recurrent Neural Networks: Unfolding [55]	359
33.2 RNN: Neuron (Recurrent Unit) [55]	360
33.3 RNN: Neuron (Recurrent Unit) [55]	360
33.4 Backpropagation Through Time (BPTT) In RNN	361
35.1 LSTM: unrolling [38]	369
35.2 LSTM Model [38]	370
36.1 GRU Architecture	373
39.1 Scaled Dot-Product Attention [3]	392
39.2 Multi-Head Attention [3]	393

LIST OF TABLES

3.1	Data: Measurement Levels: Nominal, Ordinal, Interval and Ratio [56]	9
6.1	Conditional probabilities in a 2×2 contingency table [56]	51
6.2	2×2 contingency table for exposure-outcome	57
6.3	2×2 contingency table for removal of kidney stones and two surgical treatments by size of kidney stones. [56]	60
6.4	2×2 contingency table for removal of kidney stones and two surgical treatments [56]	61
23.2	Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2 . Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d . [8]	316
31.1	Encoder–Decoder Usage by category [7]	351

LIST OF ALGORITHMS

20.1	The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory. [8]	251
20.2	SIMPLE-REFLEX-AGENT: It acts according to a rule whose condition matches the current state, as defined by the percept. [8]	252
20.3	MODEL-BASED-REFLEX-AGENT: A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent. [8]	254
20.4	A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over [8]	258
21.1	The function CHILD-NODE takes a parent node and an action and returns the resulting child node [8]	265
21.2	An informal description of the general tree-search algorithm. [8]	265
21.3	An informal description of the general graph-search algorithm. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states. [8]	266
22.1	Breadth-first search on a graph. [8]	275
22.2	UNIFORM-COST search on a graph. [8]	277
22.3	A recursive implementation of depth-limited tree search. [8]	281
22.4	The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns failure, meaning that no solution exists. [8]	283
22.5	The algorithm for recursive best-first search. [8]	292
22.6	The hill-climbing search algorithm (steepest-ascent version), which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h . [8]	294
22.7	The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The <i>schedule</i> input determines the value of the temperature T as a function of time. [8]	295
22.8	A genetic algorithm. In this more popular version, each mating of two parents produces only one offspring, not two. [8]	298
23.1	The agent program for a simple reflex agent in the two-state vacuum environment. [8]	304
38.1	Output Sequence Beam Search [6]	385

LIST OF PYTHON SNIPPETS

1.1	Absolute value function	3
3.1	Absolute Frequency - from Scratch	11
3.2	Absolute Frequency - using numPy	11
3.3	Cumulative Frequency - using numPy	11
3.4	Relative Frequency - using numPy	12
3.5	Cumulative Relative Frequency - using numPy	12
3.6	Arithmetic mean - using numPy	13
3.7	Mode - using numPy	13
3.8	Median - using numPy	14
3.9	Quantile - using numPy	14
3.10	Quartile - using numPy	14
3.11	Decile - using numPy	15
3.12	Percentile - using numPy	15
3.13	Range - using numPy	15
3.14	IQR - using numPy	15
3.15	Mean Absolute Deviation (MAD) - using numPy	16
3.16	Mean Squared Deviation (MSD)- using numPy	16
3.17	Variance - using numPy	16
3.18	Standard Deviation - using numPy	17
3.19	Skewness - using numPy	18
3.20	Kurtosis - using numPy	19
3.21	Aggregated Data - Mean - using numPy	20
3.22	Aggregated Data - Variance - using numPy	20
4.1	Box and Whiskers Plot: py-plt: face_data.csv	22
4.2	Box and Whiskers Plot: py-sns: face_data.csv	22
4.3	Count Plot: py-sns: face_data.csv	23
4.4	Density Plot: py-sns: face_data.csv	23
4.5	Histogram: py-sns: face_data.csv	24
4.6	Pairwise Plot: py-sns: face_data.csv	24
4.7	Pie Chart: py-plt: face_data.csv	25
4.8	Scatter Plot: py-plt: face_data.csv	26
4.9	Scatter Plot: py-sns: face_data.csv	26
14.1	Matrix Addition - numPy	165
14.2	Matrix Multiplication - numPy	166
14.3	Multiplication by a Scalar - numPy	167
14.4	Hadamard product - numPy	167
14.5	Matrix Inverse - numPy	168
14.6	Matrix Inverse - numPy	168
14.7	Rank of a matrix - numPy	170
14.8	Square matrix - numPy	171
14.9	Identity Matrix - numPy	171
14.10	Identity Matrix - numPy	172
14.11	Determinant of matrix - numPy	175
14.12	Trace of a matrix - numPy	176
14.13	Eigenvalues & Eigenvectors of a Matrix - numPy	178
14.14	Cholesky Decomposition - numPy	179
14.15	Eigendecomposition - numPy	181
15.1	Dimension of vector space - numPy	189

15.2 General Norm of a vector - numPy	190
15.3 Row & Column vectors - numPy	194
15.4 L1 Norm of a vector - numPy	195
15.5 L2 Norm of a vector - numPy	195
15.6 Outer product - numPy	195
15.7 Dot product - numPy	196
15.8 Check basis sets span same vector space - numPy	200
15.9 Orthogonalization using Gaussian Elimination - numPy	202
15.10 Gram-Schmidt Orthogonalization & Orthonormalization - numPy	204
15.11 Projection onto One-Dimensional Subspaces (Lines) - numPy	211
15.12 Projection onto General Subspaces - numPy	214
19.1 Problem Solving Agent - State skeleton	249
20.1 Problem Solving Agent - Problem Skeleton	259
21.1 Problem Solving Agent - Search Node	264
21.2 Problem Solving Agent - solution	264
21.3 Problem Solving Agents - child_node	265
21.4 Problem Solving Agents - tree_search	265
21.5 Problem Solving Agents - graph_search	266
22.1 Problem Solving Agent - Breadth-first search on a graph	275
22.2 Problem Solving Agent - Uniform cost search on a graph	277
22.3 Problem Solving Agent - Depth first search on a graph	279
22.4 Problem Solving Agent - Backtracking using recursion [7]	280
22.5 Problem Solving Agent - Depth limited search (recursive)	281
22.6 Problem Solving Agent - Iterative Deepening Search	283
22.7 Problem Solving Agent - Iterative Lengthening Search [7]	283
22.8 Problem Solving Agent - (General) Best-first search (BestFS) on a graph	286
22.9 Problem Solving Agent - Greedy Best-first search (GBFS) on a graph	287
22.10 Problem Solving Agent - A* search [7]	289
22.11 Problem Solving Agent - Iterative-Deepening A* (IDA*)	290
33.1 Vanilla RNN from scratch (PyTorch) [7]	361
34.1 BiRNN from scratch (PyTorch) [7]	365
35.1 LSTM from scratch (PyTorch) [7]	370
36.1 GRU from scratch (PyTorch) [7]	374

LIST OF THEOREMS

6.1 Theorem (Bayes Theorem)	52
7.1 Theorem (Fisher-Neyman (sufficient statistics))	68
7.2 Theorem (Central Limit Theorem (CLT) (Lindeberg-Levy))	72
14.1 Theorem (Square Matrix: Invertible)	168
14.2 Theorem (SPD: inner product)	172
14.3 Theorem (SPD: obtaining SPD)	173
14.4 Theorem (Symmetric Matrix: always diagonalizable)	174
14.5 Theorem (Determinant: Laplace Expansion)	175
14.6 Theorem (Determinant: product of eigenvalues)	175
14.7 Theorem (Trace: sum of eigenvalues)	176
14.8 Theorem (Eigenvalue: root of the characteristic polynomial)	177
14.9 Theorem (eigenvectors: linearly independent)	178
14.10 Theorem (Spectral Theorem)	178
14.11 Theorem (Cholesky Decomposition)	179
14.12 Theorem (Eigendecomposition)	180
14.13 Theorem (SVD Theorem)	181
14.14 Theorem (Eckart-Young Theorem)	186
15.1 Theorem (Basis Change)	201
15.2 Theorem (Rank-Nullity Theorem)	207
15.3 Theorem (isomorphic vector spaces)	208

LIST OF DEFINITIONS

6.1	Definition (sample space (Ω))	49
6.2	Definition (event space (\mathcal{A}))	49
6.3	Definition (Observed probabilities)	50
6.4	Definition (joint event/ mutual event ($A \cap B$))	50
6.5	Definition (mutually exclusive events ($P(A \cap B) = P(\emptyset) = 0$))	50
6.6	Definition (independent events ($A \perp B$) ($P(A \cap B) = P(A) \cdot P(B)$))	50
6.7	Definition (associated events/ associated variables)	51
6.8	Definition (law of total probability ($P(A) = P(A \cap B) + P(A \cap B^c)$))	51
6.9	Definition (Conjugate Prior)	54
6.10	Definition (Odds)	58
7.1	Definition (Concordant & Discordant)	84
14.1	Definition (Rank of a matrix)	169
14.2	Definition (Defective (square) matrix)	171
14.3	Definition (Symmetric, Positive Definite (SPD) Matrix)	172
14.4	Definition (Equivalent Matrices)	173
14.5	Definition (Similar Matrices)	173
14.6	Definition (Diagonalizable)	174
14.7	Definition (Spectral Norm of a Matrix)	174
14.8	Definition (Determinant)	174
14.9	Definition (Determinant: Sarrus' Rule)	175
14.10	Definition (Trace)	176
14.11	Definition (Characteristic Polynomial)	177
14.12	Definition (Eigenvalues & Eigenvectors)	177
14.13	Definition (Algebraic Multiplicity)	178
14.14	Definition (Geometric Multiplicity)	178
14.15	Definition (Eigenspace)	179
14.16	Definition (Eigenspectrum)	179
14.17	Definition (Standard SVD/ Full SVD ($A = U\Sigma V^\top$))	182
14.18	Definition (Reduced SVD ($A = U_r\Sigma_rV_r^\top$))	182
14.19	Definition (Compact SVD/ Thin SVD/ Truncated SVD ($A = U\Sigma V$))	182
15.1	Definition (Vector Space)	189
15.2	Definition (Norm)	190
15.3	Definition (Cauchy-Schwarz Inequality)	191
15.4	Definition (Metric)	192
15.5	Definition (Vector Subspace)	193
15.6	Definition (Orthogonal Vectors)	196
15.7	Definition (Codirection)	197
15.8	Definition (Collinearity)	197
15.9	Definition (Linear Combination of vectors)	197
15.10	Definition (Affine Subspaces)	197
15.11	Definition (Linear (In)dependence)	198
15.12	Definition (Generating Set)	199
15.13	Definition (Span)	199
15.14	Definition (Basis)	199
15.15	Definition (Orthonormal Basis (ONB))	202

15.16Definition (Linear Mapping)	206
15.17Definition (Image/ Range)	206
15.18Definition (Kernel/ Null Space)	206
15.19Definition (Affine Mapping)	207
15.20Definition (Injective)	207
15.21Definition (Surjective)	207
15.22Definition (Bijective)	208
15.23Definition (Isomorphism)	208
15.24Definition (Endomorphism)	208
15.25Definition (Automorphism)	208
15.26Definition (identity mapping/ identity automorphism)	208
15.27Definition (Transformation Matrix)	208
15.28Definition (projection)	210
15.29Definition (Projection error/ reconstruction error)	210
15.30Definition (Givens Rotation)	217
16.1 Definition (Univariate function ($f : \mathbb{R} \rightarrow \mathbb{R}$))	219
16.2 Definition (Difference Quotient)	219
16.3 Definition (Power Series)	219
16.4 Definition (multi-variate Functions ($f : \mathbb{R}^n \rightarrow \mathbb{R}$))	219
16.5 Definition (Partial Derivative ($\frac{\partial f}{\partial x_i}$))	219
16.6 Definition (Gradient ($\nabla_x f \in \mathbb{R}^{1 \times n}$))	220
16.7 Definition (vector-valued functions/ vector fields ($\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$))	220
16.8 Definition (Jacobian ($\mathbf{J} = \nabla_{\mathbf{x}} \mathbf{f} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}}$))	220
16.9 Definition (Hessian ($\mathbf{H} = \nabla_{x,y}^2 f(x,y)$))	221
16.10Definition (Tensor)	222
16.11Definition (Flattening (matrix → vector))	222
16.12Definition (Computational Graph)	224
16.13Definition (Automatic differentiation)	225
16.14Definition (Taylor Series ($T_\infty(x)$))	226
16.15Definition (Maclaurin series)	226
16.16Definition (Analytic)	226
16.17Definition (Multivariate Taylor Series)	226
16.18Definition (Taylor Polynomial ($T_n(x)$))	226
16.19Definition (Multivariate Taylor Polynomial ($T_n(\mathbf{x})$))	227
17.1 Definition (REF: pivot)	231

LIST OF SYMBOLS

Common Set Symbols

Symbol	Set Name	Description / Meaning	Typical Elements / Range
\mathbb{N}	Natural Numbers	Positive integers (sometimes includes 0 depending on context)	$\{1, 2, 3, \dots\}$ or $\{0, 1, 2, \dots\}$
\mathbb{Z}	Integers	All whole numbers, positive, negative, and zero	$\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$
\mathbb{Q}	Rational Numbers	Numbers expressible as a fraction of two integers	$\left\{ \frac{a}{b} : a \in \mathbb{Z}, b \in \mathbb{Z}, b \neq 0 \right\}$
\mathbb{R}	Real Numbers	All rational and irrational numbers (all points on the number line)	Includes $\pi, \sqrt{2}, -5, 0, 1.25, \dots$
\mathbb{C}	Complex Numbers	Numbers of the form $a + bi$, where $a, b \in \mathbb{R}$, and $i^2 = -1$	$\{a + bi : a, b \in \mathbb{R}\}$

Common Stats Symbols

Symbol	Description / Meaning
$\mathbb{E}[\cdot]$	Expected Value
$\mathbb{V}[\cdot]$	Variance/ Var/ VAR

I

FOUNDATION

CHAPTER 1

BASIC FUNCTIONS

1.1. Absolute Value function/ Modulus Function ($|x|$)

$$f(x) = |x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

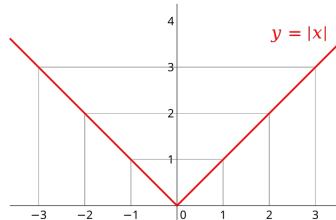


Fig. 1.1: Absolute Value function/ Modulus Function [11]

```
1 def get_absolute_value(val):  
2     if val >= 0:  
3         return val  
4     else:  
5         return -1 * val
```

Python Snippet 1.1: Absolute value function

Properties

Fundamental properties

1. $|a| \geq 0$ Non-negativity
2. $|a| = 0 \iff a = 0$ Positive-definiteness
3. $|ab| = |a||b|$ Multiplicativity
4. $|a+b| \leq |a| + |b|$ Subadditivity, specifically the triangle inequality

Additional useful properties [11]

1. $||a|| = |a|$ Idempotence (the absolute value of the absolute value is the absolute value)
2. $|-a| = |a|$ Evenness (reflection symmetry of the graph)
3. $|a-b| = 0 \iff a = b$ Identity of indiscernibles (equivalent to positive-definiteness)
4. $|a-b| \leq |a-c| + |c-b|$ Triangle inequality (equivalent to subadditivity)
5. $\left|\frac{a}{b}\right| = \frac{|a|}{|b|}$ (if $b \neq 0$) Preservation of division (equivalent to multiplicativity)
6. $|a-b| \geq ||a| - |b||$ Reverse triangle inequality (equivalent to subadditivity)

Inequalities

1. $|a| \leq b \iff -b \leq a \leq b$ [11]
2. $|a| \geq b \iff b \leq a \leq -b$ [11]

Derivative

$$1. \frac{d|x|}{dx} = \frac{x}{|x|} = \begin{cases} -1 & x < 0 \\ 1 & x > 0 \end{cases} \quad [11]$$

$$2. \frac{d}{dx} f(|x|) = \frac{x}{|x|} (f'(|x|)) \quad (\text{discontinuous at } x = 0) \quad [11]$$

$$3. \frac{d}{dx} |f(x)| = \frac{f(x)}{|f(x)|} f'(x) \quad (\text{discontinuous at } f(x) = 0) \quad [11]$$

Anti-derivative/ Integral

$$\int |x| dx = \frac{x|x|}{2} + C \quad [11]$$

1.2. Exponential Function ($\exp\{x\}$, $\exp(x)$)

$$\exp\{x\} = \exp(x) = e^x$$

1.3. Sign Function ($\operatorname{sgn}(x)$)

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \quad [56]$$

CHAPTER 2

ADVANCED FUNCTIONS

2.1. Gauss error function ($\text{erf}(z)$)

$$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt. \quad [12]$$

2.2. Gamma Function $\Gamma(x)$

$$\Gamma(z) = \begin{cases} \int_0^\infty x^{z-1} e^{-x} dx & \text{in general} \\ (k-1)! & \text{when } k \text{ is integer} \end{cases} \quad [56]$$

II

DATA

CHAPTER 3

DATA

3.1. Measurement Levels [56]

3.1.1. Nominal, Ordinal, Interval and Ratio [56]

Levels: [56]

1. Nominal
2. Ordinal
3. Interval
4. Ratio

Levels:	[56]	Categorical Data		Numerical Data	
		Nominal	Ordinal	Interval	Ratio
1. Nominal	[56]	✓	✓	✓	✓
2. Ordinal	[56]	✗	✓	✓	✓
3. Interval	[56]	✗	✗	✓	✓
4. Ratio	[56]	✗	✗	✗	✓

Table 3.1: Data: Measurement Levels: Nominal, Ordinal, Interval and Ratio [56]

Note:

1. Each consecutive measurement level contains as much "information" - in a fairly loose sense of the word - as the previous one and more. [56]

3.1.2. Continuous vs Discrete numerical data [56]

1. Continuous variables can assume any value. [56]
This means that the continuous variable can attain any value between two different values, no matter how close the two values are. [56]
Example: temperature, weight, and age [56]
2. Discrete variables cannot assume any value between 2 values. [56]
Example: number of text messages, accidents, microorganisms, students, etc. [56]

3.1.3. Outliers [56]

1. An outlier is a data point that significantly deviates from other observations in a dataset. [7]
2. Caused by Natural variability in the data or measurement errors. [7]
3. Typically identified using statistical methods like the IQR (Interquartile Range), Z-score, or visualization techniques (e.g., box plots). [7]
4. Outliers are not necessarily incorrect; they may represent rare but valid observations. [7]

Examples:

1. In a dataset of human heights, a person measuring 250 cm might be an outlier but not necessarily unrealistic if it's a rare case of gigantism. [7]

Handling/ Dealing with Outliers:

1. Ignore these abnormalities and go ahead with the data. [56]

-
2. Delete/ remove the suspected records/ entries. [56]
 3. Substitute them, using statistical methods, with a more plausible alternative. (aka **imputation**) [56]

3.1.4. Unrealistic Values [56]

1. An unrealistic value is a data point that is not plausible within the context of the dataset, often due to data entry errors or faulty sensors. [7]
2. Caused by Human error, sensor malfunction, or corruption during data transmission. [7]
3. Typically identified using domain knowledge or logical constraints. [7]
4. Unlike outliers, unrealistic values are generally not useful and need correction or removal. [7]

Examples:

1. A recorded body temperature of 200°C for a human is unrealistic, as it's physically impossible for a person to survive at that temperature. [7]
2. Negative age of a person [7]
3. Missing values [56]
4. Incorrect datatype of value [56]

Handling/ Dealing with Unrealistic values:

1. Delete/ remove the suspected records/ entries. [56]

3.2. Describing Data [56]

Some **descriptive statistics** (or just **descriptives**) that we introduce are often used for data of a certain measurement level. [56]

3.2.1. Frequency/ Frequency table [56]

Measurement levels: Nominal and ordinal data [56]

1. Frequencies are often uninformative for interval or ratio variables. [56]
if there are lots and lots of different possible values, all of them will have a count of just one. [56]
This is often tackled by discretizing (or "**binning**") the variable (which, note, effectively "throws away" some of the information in the data). [56]

```

1 import random
2 import numpy as np
3 import pandas as pd
4 from collections import Counter
5
6 # random seeding
7 random.seed(0)
8 np.random.seed(0)
9
10 # using int instead of float to have 10 unique values only
11 data = np.random.randint(0, 10, size=100) # generate sample data

```

3.2.1.1. (Absolute) Frequency/ (Absolute) Frequency table [56]

1. It refers to the count of occurrences of a particular value or category in a dataset. [7]
2. Simple count, no further processing. [7]
3. **Use Case:** Helpful in creating bar charts (SEE: [\(4.2\) Count Plot \[17\]](#)) or histograms (SEE: [\(4.4\) Histogram \[19\]](#)). [7]

```

1 def get_abs_freq_scratch(data):
2     counter = Counter()
3     for e in data:
4         counter[e] += 1
5
6     return counter

```

Python Snippet 3.1: Absolute Frequency - from Scratch

```

1 def get_abs_freq_np(data):
2     data = np.array(data) # convert data to numpy array
3     counter = np.unique_counts(data)
4     return dict(zip(counter.values, counter.counts))

```

Python Snippet 3.2: Absolute Frequency - using numPy

3.2.1.2. Cumulative Frequency/ Cumulative Frequency table [56]

1. It is the running total of frequencies up to a certain value or class. [7]
2. Each cumulative frequency includes its own frequency plus all previous frequencies. [7]
3. **Use Case:** Useful in percentile calculations and ogive graphs. [7]
4. The cumulative frequency makes more sense for ordinal data than for nominal data, since ordinal data can be ordered in size, which is not possible for nominal data. [56]

$$CF_i = CF_{i-1} + F_i$$

$$= \sum_{k=1}^i F_k$$

CF_i Cumulative Frequency at the current value
 CF_{i-1} Cumulative Frequency at the previous value
 F_i Frequency at the current value

```

1 def get_cum_freq(data):
2     counts = get_abs_freq_np(data)
3     cum_counts = {}
4     _tot = 0
5     for k, v in counts.items():
6         _tot += v
7         cum_counts[k] = _tot
8
9     return cum_counts

```

Python Snippet 3.3: Cumulative Frequency - using numPy

3.2.1.3. Relative Frequency/ Relative Frequency table [56]

1. It shows the proportion of each category relative to the total number of observations. [7]
2. Expressed as a fraction, decimal, or percentage. [7]
3. **Use Case:** Ideal for creating pie charts (SEE: (4.6) Pie Chart [15]) and understanding distribution proportions. [7]

$$RF_i = \frac{F_i}{\sum_{k=1}^N F_k}$$

RF_i	Relative Frequency
F_i	Frequency of the value
N	Total number of observations

```

1 def get_rel_freq(data):
2     abs_freq = get_abs_freq_np(data)
3     rel_freq = {}
4     for k, v in abs_freq.items():
5         rel_freq[k] = v / len(data)
6
7     return rel_freq

```

Python Snippet 3.4: Relative Frequency - using numPy

3.2.1.4. Cumulative Relative Frequency/ Cumulative Relative Frequency table [56]

1. Cumulative relative frequency is the accumulation of the relative frequencies of data points up to a certain value. [7]
2. It indicates the proportion of data points that are less than or equal to a particular value. [7]
3. **Use Cases:**
 - (a) Identifying percentiles and median. [56]
 - (b) Visualizing with a cumulative relative frequency graph (Ogive). [56]
 - (c) Understanding data distribution by determining the proportion of values below a specific threshold. [56]

$$CRF_i = \frac{\sum_{k=1}^i F_k}{N}$$

CRF_i	Cumulative Relative Frequency
F_i	Frequency of the value
N	Total number of observations

```

1 def get_cum_rel_freq(data):
2     cum_freq = get_cum_freq(data)
3     cum_rel_freq = {}
4     for k, v in cum_freq.items():
5         cum_rel_freq[k] = v / len(data)
6
7     return cum_rel_freq

```

Python Snippet 3.5: Cumulative Relative Frequency - using numPy

3.2.2. Central Tendency [56]

- When we work with numerical data, we often want to know something about the "central value" or "middle value" of the variable, also referred to as the **location** of the data. [56]

```

1 import random
2 import numpy as np
3 import pandas as pd
4
5 # random seeding
6 random.seed(0)
7 np.random.seed(0)
8
9 # generate sample data
10 # 'data' is transformed to only 1 decimal number to avoid all unique values
11 low = -100
12 high = 100
13 data = low + (high - low) * np.random.random(size=1000)
14 data = np.round(data, 1)

```

3.2.2.1. (Arithmetic) mean/ average (TODO / μ) [56]

Sample [56]

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Population [56]

$$\mu = \frac{1}{N} \sum_{i=1}^n x_i$$

Notations

μ	Population mean
\bar{x}	Sample mean
x_i	population item
x_i	sample item
N	population size
n	sample size

```

1 print(data.sum() / len(data))
2 # OR, shortcut
3 print(data.mean())

```

Python Snippet 3.6: Arithmetic mean - using numPy

3.2.2.2. Mode [56]

- The mode is merely the most frequently occurring value. [56]
- There might be multiple modes. [56]

```

1 values, counts = np.unique(data, return_counts=True)
2 mode_idx = np.argmax(counts)
3 print(values[mode_idx], counts[mode_idx])

```

Python Snippet 3.7: Mode - using numPy

3.2.2.3. Median [56]

- The median is a value that divides the ordered data from small to large (or large to small) into two equal parts: 50% of the data is below the median and 50% is above. [56]
- The median is not necessarily a value that is present in the data. [56]

Steps:

[56]

1. sort the data
2. choose the middle-most value when n is **odd**
average of the two middle values when n is **even**

```
1 print(np.median(data))
```

Python Snippet 3.8: Median - using numPy

3.2.2.4. Quantiles (q_i) [56]

1. A quantile x_q is a value that splits the ordered data of a variable x into two parts: [56]
 - (a) $q \cdot 100\%$ of the data is below the value x_q
 - (b) $(1 - q) \cdot 100\%$ of the data is above the value x_q
2. The parameter q can take any value in the interval $[0, 1]$. [56]
3. Quantiles can be calculated in different ways, depending on the way we "interpolate" between two values. [56]

We could map the ordered values *equally spaced* on the interval $(0, 1)$, where the i th ordered value of the data is positioned at the level $q_i = i/(n+1)$ in the interval $(0, 1)$, with n being the number of data points. [56]

R uses $q_i = (i-1)/(n-1)$ for quantiles. [56]

Example: [56]

- (a) Data points: $\{2, 5, 6, 4\}$ ($n = 4$)
- (b) Sorted Data points: $\{2, 4, 5, 6\}$ ($n = 4$)
- (c) Quantiles:

i	x_i	$q_i = i/(n+1) = i/5$	$q_i = (i-1)/(n-1) = (i-1)/3$
1	2	$1/5 = 0.2$	0
2	4	$2/5 = 0.4$	$1/3$
3	5	$3/5 = 0.6$	$2/3$
4	6	$4/5 = 0.8$	1

- (d) If $x_i = 3 \Rightarrow q_i = 0.3$

```
1 q = np.round(random.random(), 2)
2 print(q, np.quantile(data, q))
```

Python Snippet 3.9: Quantile - using numPy

3.2.2.5. Quartiles (q_i) [56]

1. When $q = 0.25$, $q = 0.50$, and $q = 0.75$ the quartiles are referred to as the first, second, and third quartiles, respectively. [56]
2. Splits:
 1. $q = 0$ to $q = 0.25$
 2. $q = 0.25$ to $q = 0.5$
 3. $q = 0.5$ to $q = 0.75$
 4. $q = 0.75$ to $q = 1$

```
1 _splits = 5
2 np.percentile(data, np.linspace(0, 1, _splits))
```

Python Snippet 3.10: Quartile - using numPy

3.2.2.6. Deciles (q_i) [56]

1. We call quantiles deciles when q is restricted to the set $\{0.1, 0.2, \dots, 0.9\}$ [56]

2. Splits:

1. $q = 0$ to $q = 0.1$
2. $q = 0.1$ to $q = 0.2$
⋮
9. $q = 0.8$ to $q = 0.9$
10. $q = 0.9$ to $q = 1$

```
1 _splits = 11
2 print(np.percentile(data, np.linspace(0, 1, _splits)))
```

Python Snippet 3.11: Decile - using numPy

3.2.2.7. Percentiles (q_i) [56]

1. We call quantiles percentiles when q is restricted to the set $\{0.01, 0.02, \dots, 0.99\}$ [56]

2. Splits:

1. $q = 0$ to $q = 0.01$
2. $q = 0.01$ to $q = 0.02$
⋮
99. $q = 0.98$ to $q = 0.99$
100. $q = 0.99$ to $q = 1$

```
1 _splits = 101
2 print(np.percentile(data, np.linspace(0, 1, _splits)))
```

Python Snippet 3.12: Percentile - using numPy

3.2.2.8. Range [56]

1. Range is the difference between the maximum and minimum. [56]
2. It quantifies the maximum distance between any two data points. [56]
3. The range is sensitive to outliers. [56]

```
1 print(np.max(data) - np.min(data))
```

Python Snippet 3.13: Range - using numPy

3.2.2.9. Interquartile Range (IQR) [56]

1. The interquartile range (IQR) calculates the difference between the third quartile and the first quartile. [56]
2. It quantifies a range for which 50% of the data falls within. [56]
3. The interquartile range is visualized in the boxplot. [56]

```
1 print(np.percentile(data, 0.75), np.percentile(data, 0.25))
2 print(np.percentile(data, 0.75) - np.percentile(data, 0.25))
```

Python Snippet 3.14: IQR - using numPy

3.2.2.10. Mean Absolute Deviation (MAD) [56]

$$MAD = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}| \quad [56]$$

SEE: (1.1) Absolute Value function/ Modulus Function ($|x|$)

1. average distance that data values are away from the mean. [56]

```
1 x_bar = data.mean()
2 print(np.abs(data - x_bar).mean())
```

Python Snippet 3.15: Mean Absolute Deviation (MAD) - using numPy

3.2.2.11. Mean Squared Deviation (MSD) [56]

$$MSD = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad [56]$$

1. It does the same as MAD, but now it uses squared distances with respect to the mean. [56]

SEE: (3.2.2.10) Mean Absolute Deviation (MAD) [56]

```
1 x_bar = data.mean()
2 print(np.pow(data - x_bar, 2).mean())
```

Python Snippet 3.16: Mean Squared Deviation (MSD)- using numPy

3.2.2.12. Variance (s^2 / σ^2) [56]

Sample [56]

$$\begin{aligned}s^2 &= \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \\ &= MSD \cdot \frac{n}{n-1}\end{aligned}$$

Population [56]

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^n (x_i - \mu)^2$$

Notations

μ	Population mean
\bar{x}	Sample mean
x_i	population item
x_i	sample item
N	population size
n	sample size

Note:

1. The variance is almost identical to the mean squared deviation. [56]
2. For small sample sizes the MSD and variance are not the same, but for large sample sizes they are obviously very similar. [56]
3. The variance is often preferred over the MSD. [56]

```
1 x_bar = data.mean()
2
3 # sample
4 print(np.pow(data - x_bar, 2).sum() / (len(data) - 1))
5
6 # population
7 print(np.pow(data - x_bar, 2).mean())
```

Python Snippet 3.17: Variance - using numPy

3.2.2.13. Standard Deviation (s / σ) [56]

Formulas:

1. Sample:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} = \sqrt{MSD \cdot \frac{n}{n-1}}$$

2. Population:

Note:

1. The standard deviation is on the same scale as the original variable, instead of a squared scale for the variance. [56]

```

1 x_bar = data.mean()
2
3 # sample
4 s2 = np.pow(data - x_bar, 2).sum() / (len(data) - 1)
5 print(np.sqrt(s2))
6
7 # population
8 sig2 = np.pow(data - x_bar, 2).mean()
9 print(np.sqrt(sig2))

```

Python Snippet 3.18: Standard Deviation - using numPy

3.2.2.14. Skewness (g_1) [56]

1. Computed using so-called standardized values z_i . [56]

Standardized values have no unit and the mean and variance of the standardized values are equal to 0 and 1, respectively. [56]

2. Skewness is used to measure the asymmetry in data. [56]

3. Data is considered skewed or asymmetric when the variation on one side of the middle of the data is larger than the variation on the other side. [56]

4. In practice, researchers sometimes compare the mean with the median to get an impression of the skewness, since the median and mean are identical under symmetric data, but this measure is more difficult to interpret than g_1 . [56]

5. Data with skewness values of $|g_1| \leq 0.3$ are considered close to symmetry, [56]

6. g_1 remains **unchanged** when all values $\{x_1, x_2, \dots, x_n\}$ are shifted by a fixed number or when they are multiplied with a fixed number. [56]

This means that shifting the data and/or multiplying the data with a fixed number does not change the “shape” of the data. [56]

Formulas:

1. Sample:
$$z_i = \frac{x_i - \bar{x}}{s} \quad g_1 = \frac{1}{n} \sum_{i=1}^n z_i^3$$

2. Population:

Condition	Conclusions
-----------	-------------

Condition	Conclusions
$g_1 > 0$	<ul style="list-style-type: none"> 1. data is called skewed to the right 2. The values on the right side of the mean are further away from each other than the values on the left side of the mean. 3. In other words, the “tail” on the right is longer than the “tail” on the left.
$g_1 < 0$	data is called skewed to the left and the tail on the left is longer than the tail on the right.
$g_1 = 0$	the data is considered symmetric around its mean.

```

1 x_bar = data.mean()
2
3 # sample
4 s2 = np.pow(data - x_bar, 2).sum() / (len(data) - 1)
5 s = np.sqrt(s2)
6 z = (data - data.mean()) / s
7 g1 = np.pow(z, 3).mean()
8 print(g1)
9
10 # population
11 sig2 = np.pow(data - x_bar, 2).mean()
12 sig = np.sqrt(sig2)
13 z = (data - data.mean()) / sig
14 g1 = np.pow(z, 3).mean()
15 print(g1)

```

Python Snippet 3.19: Skewness - using numPy

3.2.2.15. Kurtosis (g_2) [56]

1. Computed using so-called standardized values z_i . [56]
Standardized values have no unit and the mean and variance of the standardized values are equal to 0 and 1, respectively. [56]
2. Kurtosis is used to measure the “peakedness” of data. [56]
3. It is difficult to demonstrate that data is different from mesokurtic data when g_2 is close to zero, since it requires large sample sizes. [56]
4. Values of g_2 in the asymmetric interval of $[-0.5, 1.5]$ indicate near-mesokurtic data. [56]
5. g_2 remains **unchanged** when all values $\{x_1, x_2, \dots, x_n\}$ are shifted by a fixed number or when they are multiplied with a fixed number. [56]
This means that shifting the data and/or multiplying the data with a fixed number does not change the “shape” of the data. [56]

Formulas:

1. Sample:
$$z_i = \frac{x_i - \bar{x}}{s} \quad g_2 = \frac{1}{n} \sum_{i=1}^n z_i^4 - 3$$

2. Population:

Condition	Condition Name	Conclusions
$g_2 > 0$	leptokurtic	data has long heavy tails and is severely peaked in the middle
$g_2 < 0$	platykurtic	tails of the data are shorter with a flat peak in the middle
$g_2 = 0$	mesokurtic	

```
1 x_bar = data.mean()
2
3 # sample
4 s2 = np.pow(data - x_bar, 2).sum() / (len(data) - 1)
5 s = np.sqrt(s2)
6 z = (data - data.mean()) / s
7 g2 = np.pow(z, 4).mean() - 3
8 print(g2)
9
10 # population
11 sig2 = np.pow(data - x_bar, 2).mean()
12 sig = np.sqrt(sig2)
13 z = (data - data.mean()) / sig
14 g2 = np.pow(z, 4).mean() - 3
15 print(g2)
```

Python Snippet 3.20: Kurtosis - using numPy

3.2.3. Aggregated Data [56]

1. Data is often collected in intervals or groups with a frequency f_j for each group j . ($1 \leq j \leq m$) [56]
2. m : number of groups [56]
3. Measures of central tendency and spread can then still be computed (approximately) based on such grouped data. [56]
4. For each group j we need to determine or set the value x_j as a value that belongs to the group, before we can compute these measures. [56]

```
1 import random
2 import numpy as np
3
4 random.seed(0)
5 np.random.seed(0)
6
7 # generate aggregated data
8 low = -100
9 high = 100
10 bin_size = 5
11 count = 1000
12
13 data = low + (high - low) * np.random.random(size=count)
14 bin_edges = np.arange(low, high + bin_size, bin_size)
15 hist, edges = np.histogram(data, bins=bin_edges)
16
17 data_aggr = {(edges[i], edges[i+1]):hist[i] for i in range(len(hist))} 
18 data_x = {k: np.mean(k) for k in data_aggr}
```

3.2.3.1. Mean [56]

$$\bar{x} = \frac{\sum_{k=1}^m x_k f_k}{\sum_{k=1}^m f_k}$$

```

1 xs = np.array(list(data_x.values()))
2 fs = np.array(list(data_aggr.values()))
3 print((xs * fs).sum() / fs.sum())

```

Python Snippet 3.21: Aggregated Data - Mean - using numPy

3.2.3.2. Variance [56]

$$s^2 = \frac{\sum_{k=1}^m (x_k - \bar{x})^2 f_k}{-1 + \sum_{k=1}^m f_k}$$

```

1 xs = np.array(list(data_x.values()))
2 fs = np.array(list(data_aggr.values()))
3
4 x_bar = xs.mean()
5
6 # sample
7 print((np.pow(xs - x_bar, 2) * fs).sum() / (fs.sum() - 1))
8
9 # population
10 print((np.pow(xs - x_bar, 2) * fs).sum() / fs.sum())

```

Python Snippet 3.22: Aggregated Data - Variance - using numPy

CHAPTER 4

VISUALIZING DATA

1. Visualization, when done well, can make large and even high-dimensional datasets (relatively) easy to interpret. [56]

```
1 import random
2 import faker           # to generate fake data
3 import numpy as np
4 import pandas as pd
5 from tqdm import tqdm      # progress bar
6
7 # plotting libraries
8 import seaborn as sns
9 import matplotlib.pyplot as plt
10
11 random.seed(0)
12 np.random.seed(0)
13 faker.Faker.seed(0)
14
15 fake = faker.Faker()
```

4.1. Box and Whiskers Plot/ Box Plot [14, 16]

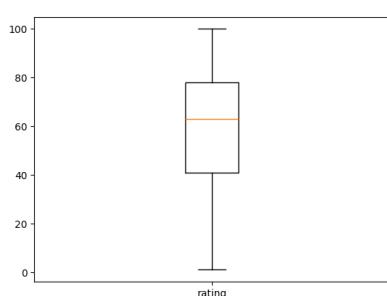
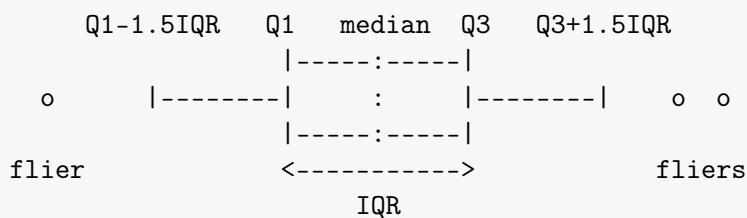


Fig. 4.1: Box and Whiskers plot (py-plt) output (face_data.csv)

```

1 _col = "rating"
2
3 plt.boxplot(df[_col])
4 plt.xticks([1], [_col])
5 plt.show()

```

Python Snippet 4.1: Box and Whiskers Plot: py-plt:
face_data.csv

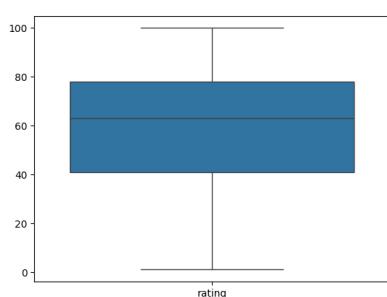


Fig. 4.2: Box and Whiskers plot (py-sns) output (face_data.csv)

```

1 _col = "rating"
2
3 sns.boxplot(df[_col])
4 plt.ylabel("rating")
5 plt.xticks([0], [_col])
6 plt.show()

```

Python Snippet 4.2: Box and Whiskers Plot: py-sns:
face_data.csv

1. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. [16]
2. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range. [16]

4.2. Count Plot [17]

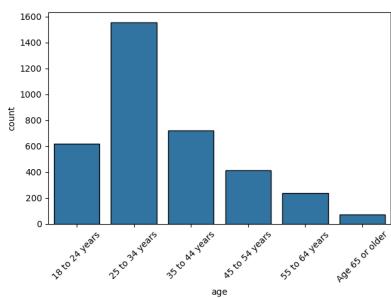


Fig. 4.3: Count plot (py-sns) output (face_data.csv)

```

1 vals = df[df["age"] != " "].copy()
2
3 sns.countplot(
4     x='age',
5     data=vals,
6     order=sorted(vals['age'].unique()),
7     edgecolor='black',
8 )
9
10 plt.xticks(rotation=45)
11 plt.tight_layout()
12 plt.show()

```

Python Snippet 4.3: Count Plot: py-sns: face_data.csv

1. Show the counts of observations in each categorical bin using bars.

4.3. Density Plot [18]

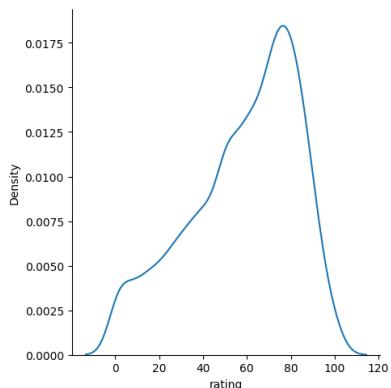


Fig. 4.4: Density plot (py-sns) output (face_data.csv)

```

1 sns.displot(df["rating"], kind="kde")
2
3 plt.show()

```

Python Snippet 4.4: Density Plot: py-sns: face_data.csv

1. A density plot - at least in this setting - can be considered a “continuous approximation” of a histogram. [56]
SEE: (4.4) Histogram [19]
2. It gives per range of values of the continuous variable the probability of observing a value within that range. [56]

4.4. Histogram [19]

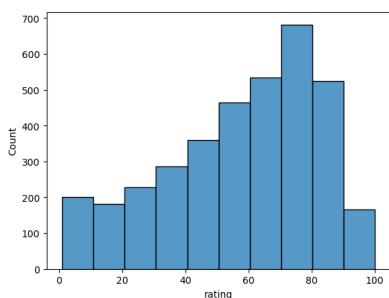


Fig. 4.5: Histogram (py-sns) output (face_data.csv)

```
1 sns.histplot(df["rating"], binwidth=10)
2 plt.show()
```

Python Snippet 4.5: Histogram: py-sns: face_data.csv

1. A histogram is a classic visualization tool that represents the distribution of one or more variables by counting the number of observations that fall within discrete bins. [19]
2. A histogram “bins” the data (discretizes it), and subsequently shows the frequency of occurrence in each bin. [56]
3. It is the continuous variant of the bar chart. [56]
SEE: (4.1) Box and Whiskers Plot/ Box Plot [14, 16]
4. The number of bins selected makes a big difference in the visualization: too few bins obscure the patterns in the data, but too many bins lead to counts of exactly one for each value. [56]

4.5. Pairwise Plot [20, 56]

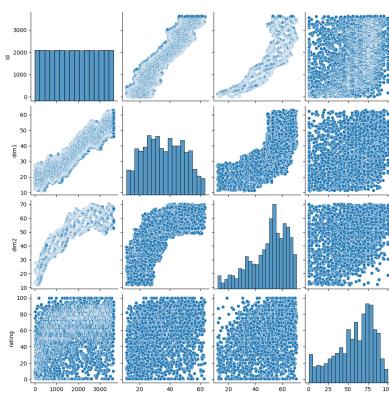


Fig. 4.6: Pairwise plot (py-sns) output (face_data.csv)

```
1 df = pd.read_csv("face_data.csv")
2 sns.pairplot(df)
```

Python Snippet 4.6: Pairwise Plot: py-sns: face_data.csv

1. Plot pairwise relationships in a dataset. [20]
2. By default, this is a grid of Axes such that each numeric variable in data will be shared across the y-axes across a single row and the x-axes across a single column. [20]
3. The diagonal plots are treated differently: a univariate distribution plot is drawn to show the marginal distribution of the data in each column. [20]

4.6. Pie Chart [15]

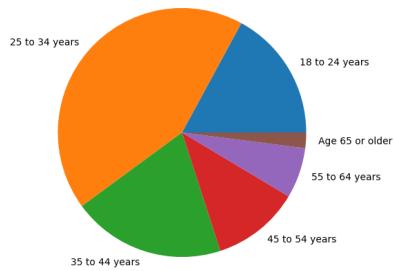


Fig. 4.7: Pie chart (py-plt) output
(face_data.csv)

```
1 vals = df[df["age"] != " "].copy()
2
3 labels, counts = np.unique(
4     vals['age'],
5     return_counts=True
6 )
7
8 plt.pie(counts, labels=labels)
9
10 plt.tight_layout()
11 plt.show()
```

Python Snippet 4.7: Pie Chart: py-plt: face_data.csv

4.7. Scatter Plot [21, 22]

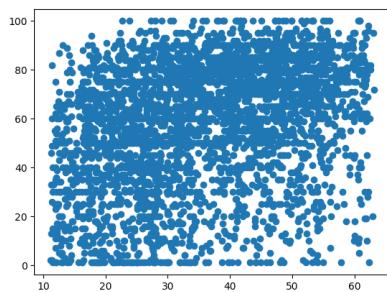


Fig. 4.8: Scatter Plot (py-plt) output
(face_data.csv)

```
1 plt.scatter(df["dim1"], df["rating"])
2
3 plt.show()
```

Python Snippet 4.8: Scatter Plot: py-plt: face_data.csv

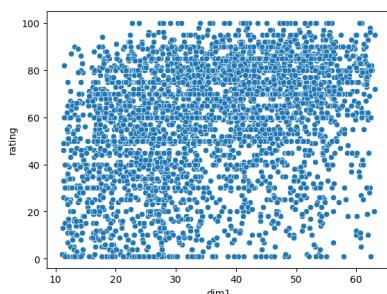


Fig. 4.9: Scatter Plot (py-sns) output
(face_data.csv)

```
1 sns.scatterplot(df, x="dim1", y="rating")
2
3 plt.show()
```

Python Snippet 4.9: Scatter Plot: py-sns: face_data.csv

1. A scatter plot, also called a **scatterplot**, **scatter graph**, **scatter chart**, **scattergram**, or **scatter diagram**, is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. [22]
2. If the points are coded (color/shape/size), one additional variable can be displayed. [22]
3. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. [22]

III

MATHEMATICS:

STATISTICS

CHAPTER 5

SAMPLING PLANS

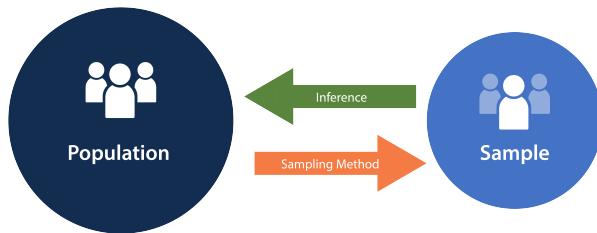


Fig. 5.1: Sampling Plan: Relation between Population and Sample

1. **Statistical Inference:** To extend your conclusions beyond the observed data. The field of **inferential statistics** tries to use the information from a sample to make statements or decisions about the population of interest. It takes into account the uncertainty that the information is coming from sampling and does not perfectly represent the population, since another sample would give different outcomes. An important aspect of inferential statistics is estimation of the population parameters of interest. [56]
2. Sampling procedures are formal *probabilistic approaches* to help collect units from the population for the sample. [56]
3. **Population:** The complete set of units that we would like to say something about is called the (target) population. [56]
In principle we would expect that a population is **always finite**, since an infinite number of units does not exist in real life. However, populations are often treated as infinite. One reason is that populations can be really really large. [56]
It is mathematically often more convenient (as we will see later) to assume that such a population is infinite. [56]
Properly defining or describing a population can be difficult. Furthermore, even if the population is established, measuring all units is often impossible or too elaborate. This means that information about the population can only be obtained by considering a subset of the population. [56]
4. **Sample:** The set of units for which we have obtained data is referred to as the sample. [56]
The sample is typically a subset of the population, although in theory the sample can form the whole population or the sample can contain units that are not from the target population. [56]
5. **Representative Sample:** A representative sample can be intuitively defined as a sample of units that has approximately the same distribution of characteristics as the population from which it was drawn. [56]
Representative sampling is also referred to as random or probability sampling. [56]
6. **Unit:** A unit is usually a concrete or physical thing for which we would like to measure its characteristics. [56]
7. **Estimates:** In terms of statistical inference, the calculations on the sample data are referred to as estimates for the theoretical value in the whole population. [56]
8. **Estimators:** quantities that we compute using the data in our sample to say something about the population. [56]
9. **Realization:** The values in the sample are referred to as a realization from the population. [56]
10. Reasons for sample instead of population:

- (a) In many applications we really can't measure the complete population. For instance, one of the tests applied to aircraft engines is the “frozen bird test”. [56]
- (b) Time, space, or budget restrictions often do not allow us to measure all units from a population. [56]
- (c) Big data itself may be an argument for sampling. If we have a very large sample or we have been able to measure all units from the population, the resulting dataset can be so large that it becomes impossible to analyze the full data at one computer. [56]
11. A non-representative sample implies that we do not know the exact process by which units in the population became part of the sample. [56]
12. If we know which sampling procedure was applied to collect the units for the sample, we would also know how close the calculations or statistics would be to the theoretical value in the whole population. [56]
- Thus the sampling procedure and the choice of calculation on the sample data (Arithmetic) mean/ average (TODO / μ) [56],
 Median [56],
 Quartiles (q_i) [56],
 Standard Deviation (s / σ) [56]
 etc.) would make statistical inference mathematically precise and it would therefore help us when making statements beyond the sample data. [56]

5.1. Generic Formulation [56]

Population Size	N	[56]
Sample Size	n	1. $n \leq N$ [56]
Number of Possible Samples	K	1. exact value of K depends on the sampling plan [56]
		1. $\Omega = \{1, 2, \dots, N\}$ [56]
Population	Ω	2. $\Omega = \bigcup_{k=1}^K S_k$ [56]
		1. Subset of Population [56]
		2. $k \in \{1, 2, \dots, K\}$ [56]
		3. $S_k = \{i_1, i_2, \dots, i_n\}$ ($i_h \in (1, \dots, N)$) [56]
		(n unique units, $h \neq l \Rightarrow i_h \neq i_l$) [56]
Sample	S_k	4. $S_k \subset \Omega \quad \forall k \in (1, \dots, K)$ [56]
		5. Each subset is unique: $k \neq l \Rightarrow S_k \neq S_l$ [56]
		6. Subsets may overlap: $S_k \cap S_l \neq \emptyset$ [56]
Unit	i	[56]
Unit's theoretical value	x_i	[56]

Sample probability	π_k	1. each subset S_k is attached a probability π_k	[56]
		2. $\pi_k > 0 \quad k \in \{1, \dots, K\}$	[56]
		3. $\sum_{k=1}^K \pi_k = 1$	[56]
Unit Probability	p_i	1. Probability of each unit in population	[56]
		2. $p_i > 0 \quad i \in \{1, \dots, N\}$	[56]
		3. $\sum_{i=1}^N p_i \neq 1$ since samples overlap	[56]
		4. the probabilities are not always the same for each unit.	[56]
Population Parameter	θ	1. $\theta \equiv \theta(\mathbf{x})$ where $\mathbf{x} = (x_1, x_2, \dots, x_N)$	[56]
Observations	\mathbf{x}_k^\top	1. $\mathbf{x}_k^\top = (x_{i_1}, x_{i_2}, \dots, x_{i_n})$	[56]
		2. Observed with every sample S_k	[56]
Descriptive Statistic/ Estimate	$\hat{\theta}_k$	1. $\hat{\theta}_k = T(\mathbf{x}_k)$	[56]
		2. Computed based on the observed data.	[56]
		3. used as an <i>estimate</i> for the population parameter θ	[56]
Estimator	$T(\cdot)$	1. It is a function applied to the observed data (i.e., some calculation procedure).	[56]
		2. In many cases the function T is identical to the calculation θ at the population level, but alternative functions may be used depending on the sampling plan.	[56]
		3. Example: For estimating population mean, \bar{x}_k can be used as T	[56]
Expected Population Parameter	$\mathbb{E}[T]$	1. $\mathbb{E}[T] = \sum_{k=1}^K \hat{\theta}_k \pi_k$	[56]
		2. $\mathbb{E}[cT] = c \mathbb{E}[T] \quad \forall c \in \mathbb{R}$	[56]
Weighted Average	$\bar{x}_{w,k}$	1. $\bar{x}_{w,k} = \sum_{i \in S_k} w_{ik} x_i$	[56]
		2. $\sum_{i \in S_k} w_{ik} = 1$	[56]
		3. If every observation has the same weight, we obtain the arithmetic average $\bar{x}_k = \frac{1}{n} \sum_{i \in S_k} x_i$	[56]

1. The set of samples S_1, S_2, \dots, S_K with their probabilities $\pi_1, \pi_2, \pi_3, \dots, \pi_K$ is referred to as a **sampling plan**. [56]
2. The sampling plan contains all the information necessary to analyze the quality of a sampling procedure. [56]

3. **Population Mean:** $\mu = \frac{1}{N} \sum_{i=1}^N x_i$ [56]

4. **Population Variance:** $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$ [56]

5. In general, the value $\hat{\theta}_k$ can be considered an estimate of the population parameter θ when sample S_k would be collected. [56]

6. The estimate $\hat{\theta}_k$ will most likely be different from the population parameter θ , because the sample is just a subset of the population. [56]

5.2. Measures of closeness

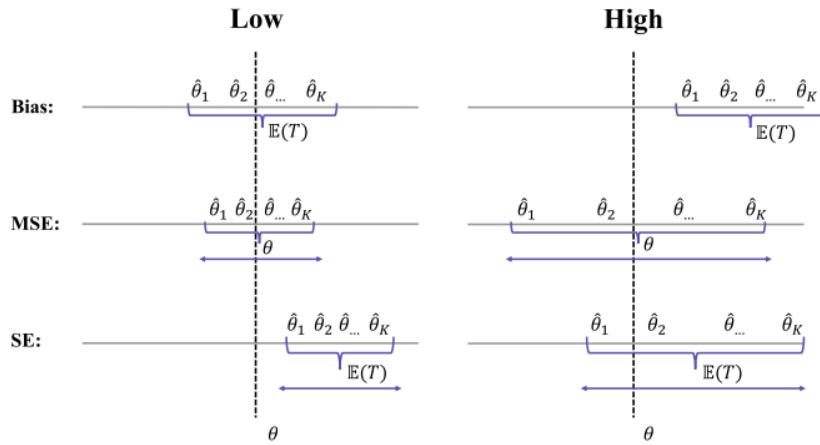


Fig. 5.2: Visual Representation of Bias, MSE and SE

5.2.1. Bias

$$bias = \left(\sum_{k=1}^K \hat{\theta}_k \pi_k \right) - \theta = \mathbb{E}[T] - \theta \quad [56]$$

1. The bias is the difference between the weighted average - over all possible K samples - of the sample estimate $\hat{\theta}_k$'s and the true population parameter θ . [56]
2. The weights in this weighted average are provided by the probabilities π_k . [56]
3. If the bias of an estimator is **zero**, this means that, if we repeatedly take samples using our sampling plan and repeatedly compute our statistic of interest, the average over all of those statistics is equal to the true population parameter. [56]
If the bias is **zero**, the estimator, under the sampling plan that is being evaluated, is said to be **unbiased**. [56]
4. The bias of an estimator is thus the difference between this estimator's expected value and the true population value. [56]
5. A small bias of an estimator under a sampling plan does **not** guarantee that individual sample results $\hat{\theta}_k$ are actually close to the population parameter θ ; it just states that they are close on average, if we were to sample over and over again. [56]
6. If the bias is small, $\mathbb{E}[T]$ is close to the parameter value θ .
If the bias is large, $\mathbb{E}[T]$ is **not** close to θ [56]
7. If the sampling plan is unbiased and thus $\mathbb{E}[T] = \theta$, the RMSE and the SE are identical. [56]

5.2.2. Mean Square Error (MSE)

$$MSE = \sum_{k=1}^K (\hat{\theta}_k - \theta)^2 \pi_k \quad [56]$$

1. To capture the variability in the sample results $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_K$ with respect to the true value θ , we use the so-called mean squared error (MSE). [56]
2. The MSE measures the weighted average squared distance of the sample results $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_K$ from the population parameter θ . [56]
3. The weights are determined by the sampling probabilities. [56]

4. The smaller the MSE the better the sampling plan. [56]
5. If the MSE is small, the variability of the $\hat{\theta}_k$'s around θ is small, while if the MSE is large, the variability around θ is large. [56]

5.2.3. Root Mean Square Error (RMSE)

$$RMSE = \sqrt{MSE} = \sqrt{SE^2 + (\mathbb{E}[T] - \theta)^2} \quad [56]$$

5.2.4. Standard Error (SE)

$$SE = \sqrt{\sum_{k=1}^K (\hat{\theta}_k - \mathbb{E}[T])^2 \pi_k} \quad [56]$$

1. It represents the variability of the sampling plan with respect to the expected population parameter $\mathbb{E}[T]$ instead of using the true population parameter θ . [56]
2. Standard error of an estimator is used as a measure to represent our uncertainty regarding an estimate. [56]
3. If the SE is small, the variability of the $\hat{\theta}_k$'s around $\mathbb{E}[T]$ is small. [56]
4. $SE(cT) = c SE(T) \quad \forall c \in \mathbb{R}$ [56]

5.3. Types of Samplings

1. **Non-representative Sampling** [56]
 - (a) Although these sampling methods are frequently in use, it is strongly recommended not to apply these methods, unless knowledge is available on how to adjust or correct the sample for inferential purposes. [56]
 - (b) These have the risk that some units are much more likely to be included in the sample than others, which can make statistics computed on the sample data bad estimates for the population parameters of interest. [56]
 - (c) With non-representative sampling some units are not only more likely to be included in the sample, we also do not actually know how likely units were included. [56]
 - (d) Even if we wanted to, we could not control for these systematic differences between units. [56]

SEE:

- (a) [\(5.4\) Convenience Sampling](#) [56]
- (b) [\(5.5\) Haphazard Sampling](#) [56]
- (c) [\(5.6\) Purposive Sampling/ Judgmental Sampling](#) [56]

2. **Representative Sampling** [56]
 - (a) We sample units in such a way that we do know how likely units are to be included in the sample (even if they will be different from unit to unit). [56]
 - (b) Random sampling is a sampling method that uses a random mechanism.
 - i. The probability of each unit in the population of becoming part of the sample is both positive and known. [56]

SEE:

- (a) [\(5.7\) Simple Random Sampling](#) [56]

- (b) [\(5.8\) Systematic Sampling \[56\]](#)
- (c) [\(5.9\) Stratified Sampling \[56\]](#)
- (d) [\(5.10\) Cluster Sampling \[56\]](#)

5.4. Convenience Sampling [56]

1. Convenience sampling collects only units from the population that can be easily obtained. [56]
2. This may provide a biased sample, as it represents only one small part or time window of the whole processing window for a batch of products. The term **bias** indicates that we obtain the value of interest with a systematic mistake. [56]
3. Convenience sampling is often justified by using the argument of population homogeneity. This insinuates that either the population units are not truly different or the process produces the population of units in random order. [56]

5.5. Haphazard Sampling [56]

1. Haphazard sampling is often believed to be an excellent way of collecting samples, because it gives a feeling or the impression that each unit was collected completely at random. [56]
2. Despite the feeling of randomness when performing haphazard sampling, often the resulting sample is not truly random. [56]

5.6. Purposive Sampling/ Judgmental Sampling [56]

1. Purposive sampling or judgmental sampling tries to sample units for a specific purpose. [56]
2. This means that the collection of units is focused on one or more particular characteristics and hence it implies that only units that are more alike are sampled. [56]
3. This way of sampling is strongly related to the definition of the population, since deliberately excluding units from the sample is analogous to limiting the population of interest. [56]
4. Purposive sampling may be useful, but it is limited since it does not allow us in general to make statements about the whole population, and at best only about a limited part of the population (although we may not be sure either). [56]
5. It does most likely produce a biased sample with respect to the complete population. [56]

5.7. Simple Random Sampling [56]

N	population size
n	sample size
K	number of possible samples
k	sample index ($k \in (1, \dots, K)$)
S_k	sample

1. Implicitly assume that there is no particular group structure present in the population. [56]
2. Simple random sampling is a way of collecting samples such that each unit from the population has the exact same probability of becoming part of the sample. [56]
3. Simple random sampling is a conceptually easy method of forming random samples but it can prove hard in practice. [56]
4. Simple random sampling is frequently combined with other choices or settings (see stratified and cluster sampling). [56]
5. Number of unique samples = $K = \frac{N!}{n!(N-n)!}$ [56]
6. the probability of collecting sample S_k , using sequential sampling = $\frac{1}{K} = \frac{n!(N-n)!}{N!}$ [56]
7. The probability that a specific unit is part of the sample = $\frac{n}{N}$ [56]
8. The probability that a specific unit is **not** contained in the sample = $1 - \frac{n}{N} = \frac{N-n}{N}$ [56]
9. The number of samples that does **not** contain a specific unit = $\frac{(N-1)!}{n!(N-1-n)!}$ [56]
10. Number of samples that contain certain unit $i = \frac{nK}{N}$ [56]

$$\Rightarrow \sum_{k=1}^K \sum_{i \in S_k} x_i = \frac{nK}{N} \sum_{i=1}^N x_i$$
 [56]
11. Population Variance: $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$ [56]
12. **Disadvantage:** When the numbers of units across these subpopulations are (substantially) different, simple random may not collect units from each subgroup. [56]

Example:

$$N = 20, n = 5$$

$$K = \frac{20!}{5! \cdot 15!} = 15504$$

5.7.1. Estimation for population mean

1. Estimator: $T = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ [56]
 2. Bias: 0 [56]
 3. $MSE(\bar{x}_k) = \frac{\sigma^2}{n} \left(\frac{N}{N-1} \right) \left(1 - \frac{n}{N} \right) = \frac{\sigma^2 (N-n)}{n(N-1)}$ [56]
- MSE shows that it becomes equal to zero when the sample size n becomes equal to the population size

N .

[56]

The MSE will not become zero when the estimator is biased, even if the sample size is equal to the population size.

[56]

$$4. \mathbb{E}[T] = \frac{1}{K} \sum_{k=1}^K \bar{x}_k = \frac{1}{nK} \sum_{k=1}^K \sum_{i \in S_k} x_i = \frac{1}{N} \sum_{i=1}^N x_i = \mu \quad [56]$$

5.7.2. Estimation of the sample MSE

1. unbiased estimator for σ^2 :

$$\hat{\sigma}^2 = \frac{N-1}{N} s_k^2 \quad \left(s_k^2 = \frac{1}{n-1} \sum_{i \in S_k} (x_i - \bar{x}_k)^2 \right) \quad [56]$$

$$2. \hat{MSE}(\bar{x}_k) = \frac{N-n}{Nn} s_k^2 \quad [56]$$

3. This is an unbiased estimator of the MSE of the arithmetic average in $MSE(\bar{x}_k)$ and it may also be used when the observations are binary.

[56]

5.7.3. Sample Statistics (T_n)

1. We can view a sample X_1, X_2, \dots, X_n of size n from a population as a set of random variables (and x_1, x_2, \dots, x_n as the set of realizations) all coming from the same distribution function F . Let X_1, X_2, \dots, X_n be i.i.d. with $X_i \sim F$

[56]

2. A sample statistic $T_n \in \mathbb{R}$ is now defined as any function $T_n \equiv T(X_1, X_2, \dots, X_n)$ that is applied to the sample X_1, X_2, \dots, X_n . As T_n is a function of random variables, it is itself a random variable.

[56]

3. realization for the sample statistic T_n : $t_n = T(x_1, x_2, \dots, x_n)$

[56]

$$4. \text{Sample average: } T_n = \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad [56]$$

$$5. \text{Sample variance: } T_n = S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2 \quad [56]$$

$$6. \text{Sample standard deviation: } T_n = S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2} \quad [56]$$

$$7. \text{Sample skewness: } T_n = b_1 = \frac{1}{nS^3} \sum_{i=1}^n (X_i - \bar{X})^3 \quad [56]$$

$$8. \text{Sample excess kurtosis: } T_n = b_2 = \left(\frac{1}{nS^4} \sum_{i=1}^n (X_i - \bar{X})^4 \right) - 3 \quad [56]$$

9. Sample minimum: $T_n = X_{(1)} = \min \{X_1, X_2, \dots, X_n\}$

[56]

10. Sample maximum: $T_n = X_{(n)} = \max \{X_1, X_2, \dots, X_n\}$

[56]

11. quantile:

- (a) If $np \in \mathbb{N}$ is an integer, the quantile x_p is estimated by the average of two sequential order statistics: $q_p = \frac{[X_{(np)} + X_{(1+np)}]}{2}$.

[56]

- (b) If $np \notin \mathbb{N}$ is not an integer, we take the smallest integer value that is larger than or equal to np , which is denoted by $\lceil np \rceil$.

[56]

5.8. Systematic Sampling [56]

N	population size	$N = nm$
n	number of groups	= sample size
m	number of units in each group	= number of possible samples
k	sample index	$k \in (1, \dots, m)$
S_k	sample	

1. Implicitly assume that there is no particular group structure present in the population. [56]
2. Steps:
 - (a) First the population should be divided into n groups and the order of the units (if Some order exists) should be maintained (or otherwise fix the order).
Each group consists of m units ordered from 1 to m in each group.
 - (b) Collect p th unit ($p \in (1, \dots, m)$) from all n groups with probability of $\frac{1}{m}$
(each unit in the population still has the same probability of being collected)
 $S_k = \{k, k+m, k+2m, \dots, k+(n-1)m\}$ (Total units collected = n)
So, total number of samples = m only
3. The possible samples from systematic sampling are quite different from the set of samples that can be obtained with simple random sampling. [56]
4. population mean: $\mu = \frac{1}{N} \sum_{h=1}^m \sum_{i=1}^n x_{k+m(i-1)}$ [56]
5. population variance: $\sigma^2 = \frac{1}{N} \sum_{k=1}^m \sum_{i=1}^n (x_{k+m(i-1)} - \mu)^2$ [56]
6. sample average for sample S_k : $\bar{x}_k = \frac{1}{n} \sum_{i=1}^n x_{k+m(i-1)}$ [56]
7. Systematic sampling can be more efficient than simple random sampling, in particular when the variance in the systematic samples is larger than the population variance (which is impossible to verify in practice). [56]
8. The most important **advantage** of systematic sampling over simple random sampling is the ease with which the sample may be collected. [56]
9. A clear **disadvantage** of systematic sampling is that the “period” for systematic sampling may coincide with particular patterns in the process or population. [56]
10. **Disadvantage:** When the numbers of units across these subpopulations are (substantially) different, systematic sampling may not collect units from each subgroup. [56]
11. TODO? Systematic Sampling samples (S_1, \dots, S_m) are subset of Simple Random Sampling samples (S_1, \dots, S_K) .

Example:

$$N = 20, n = 5$$

$$m = \frac{20}{5} = 4$$

5.8.1. Estimation for population mean

1. Estimator: $\frac{1}{n} \sum_{i=1}^n x_i$ [56]

if population can be perfectly split up into n groups of m units:

Unbiased Estimator: \bar{x}_k

2. Bias: 0 [56]

3. MSE: $\sigma^2 - \frac{1}{N} \sum_{h=1}^n \sum_{i=1}^m (x_{h+m(i-1)} - \bar{x}_h)^2$ [56]

5.8.2. Estimation of the MSE

1. In the general setting for systematic sampling, an unbiased estimation of the MSE is **not possible**. [56]

2. Given that there is no systematic difference between units based on their position in the groups (and the ratio of sample and population size is an integer), the MSE of the sample average under systematic sampling becomes equal to the MSE of the sample average under simple random sampling. [56]

3. unbiased estimator: $\frac{N-1}{N} s_k^2 \quad s_k^2 = \frac{1}{n-1} \sum_{i=1}^n (x_{k+m(i-1)} - \bar{x}_k)^2$ [56]

5.9. Stratified Sampling [56]

N	total population size	
n	total sample size	
M	number of sub-populations	
N_h	k th sub-population size	$\sum_{h=1}^M N_h = N$
n_h	sample size from k th sub-population	$\sum_{h=1}^M n_h = n$
h	Stratum index	$h \in \{1, \dots, M\}$
k	Stratum index	$k \in \{1, \dots, K_S\}$
K_h	The number of possible samples that can be drawn from stratum h	
K_S	total number of possible samples	
(h, i)	index of unit	$i \in \{1, 2, \dots, N_h\}$ and $h \in \{1, 2, \dots, M\}$
x_{hi}	unit i in stratum h	
w_h	weight of strata h while sampling	$w_h = \frac{N_h}{N}$
f_h	sample fraction in stratum h	$f_h = \frac{n_h}{N_h}$
$S_{h,k}$	<p>$S_{h,k}$ is the collected sample in stratum h</p> <p>The k-th possible sample from stratum h</p> <p>(e.g., $S_{2,3}$ is the 3rd possible sample from stratum 2)</p> <p>focuses on one stratum</p>	
S_k	<p>The k-th possible full stratified sample (i.e., across all strata).</p> <p>Each S_k includes one selected sample from each stratum.</p> <p>focuses on a complete sample (all strata combined)</p>	

1. Stratified sampling is used to accommodate the issue of missing out on sub-populations during sampling by setting the sample size for each subpopulation (often called **strata**) to a fixed percentage of the number of units of the subpopulation. [56]
2. Steps:
 - (a) The population is then divided into n groups such that the order in units is maintained.
 - (b) From k th group/ strata one unit is randomly collected with probability $\frac{n_h}{N_h}$, when each group contains n_h units.
Unlike systematic sampling, random unit is collected instead of p th unit.
3. This form of stratified sampling is not identical to systematic sampling (although this method of stratified sampling is sometimes referred to as systematic sampling). [56]
4. Stratified sampling may lead to samples that are not possible with systematic sampling, but it does not produce all possible samples from simple random sampling. [56]
5. $K_h = \binom{N_h}{n_h} = \frac{N_h!}{n_h!(N_h - n_h)!}$ [56]
6. The number of possible samples: $K_S = \prod_{h=1}^M \frac{N_h!}{n_h!(N_h - n_h)!} = \prod_{h=1}^M K_h$ [56]
7. The probability of collecting any of the K_S samples = $\frac{1}{K_S}$ [56]

8. The probability of collecting a unit now depends on the stratum the unit is part of. [56]
9. If $\frac{n_i}{N_i} = \frac{n_j}{N_j} \quad \forall i, j \in (1, \dots, M)$, then it's called **Proportional Stratified Sampling**. [56]

Example: sample 10% from each sub-population

5.9.1. population stratum

1. population stratum mean: $\mu_h = \frac{1}{N_h} \sum_{i=1}^{N_h} x_{hi}$ [56]
2. population stratum variance: $\sigma_h^2 = \frac{1}{N_h} \sum_{i=1}^{N_h} (x_{hi} - \mu_h)^2$ [56]

5.9.2. population

1. population mean: $\mu = \frac{1}{N} \sum_{h=1}^M \sum_{i=1}^n x_{hi} = \sum_{h=1}^M w_h \mu_h$ [56]
 - (a) $w_h = \frac{N_h}{N}$ [56]
 - (b) $\sum_{h=1}^M w_h = 1$ [56]
2. population variance: $\sigma^2 \equiv \frac{1}{N} \sum_{h=1}^M \sum_{i=1}^{N_h} (x_{hi} - \mu)^2 = \sum_{h=1}^M w_h \sigma_h^2 + \sum_{h=1}^M w_h (\mu_h - \mu)^2$ [56]
 - (a) **within (strata) variances**: the first part represents a weighted mean of the within strata variances
 - (b) **between (strata) variances**: the second part represents a weighted mean of the squared distances of the strata means to the population mean

5.9.2.1. Estimation for population mean

1. Estimator: $\sum_{h=1}^M \frac{N_h}{N} \left(\frac{1}{n_h} \sum_{i=1}^{n_h} x_{hi} \right) = \sum_{h=1}^M w_h \bar{x}_{h,k}$ [56]
2. Bias: 0 [56]
3. MSE: $\sum_{h=1}^M \left[\left(\frac{N_h^2 (N_h - n_h)}{n_h (N_h - 1) N^2} \right) \sigma_h^2 \right]$ [56]

5.9.3. sample stratum

1. sample stratum mean: $\bar{x}_{h,k} = \frac{1}{n_h} \sum_{i \in S_{h,k}} x_{hi}$ [56]
2. sample stratum variance: $s_{h,k}^2 = \frac{1}{n_h - 1} \sum_{i \in S_{h,k}} (x_{hi} - \bar{x}_{h,k})^2$ [56]
3. MSE for unbiased estimator: $MSE(\bar{x}_{h,k}) = \mathbb{V}[\bar{x}_{h,k}] = SE^2(\bar{x}_{h,k})$ [7]
4. The bias and standard error within each stratum h now follow the theory of simple random sampling. [56]
5. bias of $\bar{x}_{h,k} = 0$ in stratum h is zero for the stratum mean μ_h

5.9.3.1. Estimation for sample stratum SE

$$1. \hat{SE}(\bar{x}_{h,k}) = \sqrt{\frac{1-f_h}{n_h}} s_{h,k} \quad f_h = \frac{n_h}{N_h} \quad [56]$$

5.9.4. sample

$$1. \text{ Sample average: } \bar{x}_k = \sum_{h=1}^M w_h \bar{x}_{h,k} \quad [56]$$

2. the *simple random samples* from the different strata are completely **unrelated**, which implies that the sum of the squared standard errors of $w_h \bar{x}_{h,k}$ form the squared standard error of the sample average \bar{x}_k .
[56]

$$\mathbb{V}[\bar{x}_k] = \sum_{h=1}^M w_h^2 \cdot \mathbb{V}[\bar{x}_{h,k}] \quad [7]$$

$$3. \text{ squared standard error of } \bar{x}_k: SE^2(\bar{x}_k) = \sum_{h=1}^M w_h^2 \cdot SE^2(\bar{x}_{h,k}) \quad [7]$$

$$4. MSE(\bar{x}_k) = \sum_{h=1}^M w_h^2 \cdot MSE(\bar{x}_{h,k}) = \sum_{h=1}^M \frac{N_h}{(N_h-1)n_h} (1-f_h) w_h^2 \sigma_h^2 \quad [56]$$

5.9.4.1. Estimation of the sample MSE

$$MSE(\bar{x}_k) = \sum_{h=1}^M \frac{1-f_h}{n_h} w_h^2 s_{h,k}^2. \quad [56]$$

SEE: <https://chatgpt.com/share/680251af-c100-800a-a533-c26a0001b651>

5.9.4.2. Estimation of the sample SE

An estimate of the standard error of \bar{x}_k is now obtained by taking the square root of the estimated MSE:
[56]

$$\hat{SE}(\bar{x}_k) = \sqrt{MSE(\bar{x}_k)}$$

5.10. Cluster Sampling [56]

1. Cluster sampling involves random sampling of groups or clusters of units in the population. [56]
2. Cluster sampling can be less representative than sampling units directly. [56]
3. Cluster sampling introduces a specific structure in the sample which should also be addressed when the data is being analyzed. [56]
4. The cluster structure introduces two sources of variation in the data being collected. These sources of variation need to be quantified to make proper statements on the population of interest. [56]
 - (a) **Within-Cluster Variation:** [7]
 - i. This refers to the differences among units within the same cluster. [7]
 - ii. This variation is often smaller compared to between-cluster variation because individuals within a cluster tend to be more similar. [7]
 - (b) **Between-Cluster Variation:** [7]
 - i. This refers to the differences between clusters. [7]
 - ii. This variation is often larger in cluster sampling because different clusters may have distinct characteristics. [7]
5. The sampling units for the *first stage* are referred to as **primary cluster units**. [56]
6. Sampling these different levels of clusters can be performed using simple random sampling, systematic sampling, or even stratified sampling, if certain cluster are put together on certain criteria. [56]
7. Cluster sampling is in a way related to stratified sampling (clusters may be viewed as strata). [56]
8. Since we deal with multiple levels of hierarchical clusters, the calculation of the probability of collecting one unit from the population and the probability of collecting one of the many sample sets are complex. [56]

5.10.1. Single-Stage Cluster Sampling

A single-stage cluster sample uses a random sample of the clusters and then all units from these clusters are selected. [56]

5.10.1.1. cluster sizes are not all equal

$$\text{Estimator for population mean: } \sum_{h=1}^m \left(\frac{N_h}{\sum_{h=1}^m N_h} \right) \bar{x}_h \quad [56]$$

1. Bias: ≥ 0 (bias is close to zero but positive) [56]

$$2. \text{ MSE: } \sim \frac{M^2}{m(M-1)N^2} \left(1 - \frac{m}{M} \right) \sum_{h=1}^M N_h^2 (\mu_h - \mu)^2 \quad [56]$$

5.10.1.2. cluster sizes are all equal

$$\text{Estimator for population mean: } \frac{M}{mN} \sum_{h=1}^m N_h \bar{x}_h \quad [56]$$

1. Bias: 0 [56]

$$2. \text{ MSE: } \frac{M^2}{m(M-1)N^2} \left(1 - \frac{m}{M} \right) \sum_{h=1}^M \left(N_h \mu_h - \frac{N\mu}{M} \right)^2 \quad [56]$$

5.10.2. Two-Stage Cluster Sampling

In a two-stage cluster sample, the units from the sampled clusters are also randomly sampled instead of taking all units from the cluster. [56]

5.10.2.1. cluster sizes are not all equal

$$\text{Estimator for population mean: } \sum_{h=1}^m \left(\frac{N_h}{\sum_{h=1}^m N_h} \right) \bar{x}_h \quad [56]$$

1. Bias: ≥ 0 (bias is close to zero but positive) [56]

$$2. \text{ MSE: } \sim \frac{M}{m N^2} \left[\left(1 - \frac{m}{M} \right) \frac{1}{M-1} \sum_{h=1}^M N_h^2 (\mu_h - \mu)^2 + \sum_{h=1}^M \left(\frac{N_h^2 (N_h - n_h)}{n_h (N_h - 1)} \sigma_h^2 \right) \right] \quad [56]$$

5.10.2.2. cluster sizes are all equal

$$\text{Estimator for population mean: } \frac{M}{m N} \sum_{h=1}^m N_h \bar{x}_h \quad [56]$$

1. Bias: 0 [56]

$$2. \text{ MSE: } \frac{M}{m N^2} \left[\left(1 - \frac{m}{M} \right) \frac{1}{M-1} \sum_{h=1}^M \left(N_h \mu_h - \frac{N \mu}{M} \right)^2 + \sum_{h=1}^M \left(\frac{N_h^2 (N_h - n_h)}{n_h (N_h - 1)} \sigma_h^2 \right) \right] \quad [56]$$

5.10.3. Multi-Stage Cluster Sampling

It is similar to two-stage cluster sampling, but multiple number of stages based on the application. [56]

5.11. Cross-Sectional Study (population-based)

SEE: (6.4) Conditional Probability ($P(A|B)$)

1. a **simple random sample** of size n is taken from the population [56]
2. For each unit in the sample both the exposure and outcome are being observed and the units are then summarized into the four cells (E,D) , (E,D^c) , (E^c,D) , and (E^c,D^c) . The 2×2 contingency table would then contain the number of units in each cell. [56]
3. This way of sampling implies that the proportions in the last row ($P(D)$ and $P(D^c)$) and the proportions in the last column ($P(E)$ and $P(E^c)$) of contingency table would be unknown before sampling and they are being determined by the probability of outcome and exposure in the population. [56]
4. the observed probabilities in Table (6.1) Conditional probabilities in a 2×2 contingency table [56] obtained from the sample represent unbiased estimates of the population probabilities. [56]
5. we apply the theory of simple random sampling for estimation of a population proportion.
 - (a) if we define the binary variable x_i by 1 if unit i has both events E and D (thus $E \cap D$) and it is zero otherwise, the estimate of the population proportion $P(E \cap D)$ would be the sample average of this binary variable.
 - (b) This sample average is equal to the number of units in cell (E,D) divided by the total sample size n
6. calculation of the risk difference, the relative risk, and the odds ratio are all appropriate for cross-sectional studies. [56]

5.12. Cohort Study (exposure-based)

1. a simple random sample is taken from the population of units who are exposed and another simple random sample is taken from the population of units who are unexposed. Thus this way of sampling relates directly to **stratified sampling** with the strata being the group of exposed (E) and the group of unexposed (E^c). [56]
2. In each sample or stratum the outcome D is noted and the contingency table in Table (6.1) Conditional probabilities in a 2×2 contingency table [56] is filled. [56]
3. In this setting, the probabilities $P(E)$ and $P(E^c)$ are preselected before sampling and are fixed in the sample, whatever they are in the population. Thus the sample and the population may have very different probabilities. [56]
4. the probabilities in the cells of the contingency tables are no longer appropriate estimates for the population probabilities, since we have destroyed the ratio in probabilities for E and E^c . [56]
5. Despite the fact that we cannot use the joint probabilities in the contingency table as estimates for the population probabilities, the risk difference, the relative risk, and the odds ratio in the sample are all appropriate estimates for the population when a cohort study is used. The reason is that these measures use the conditional probabilities only, where conditioning is done on the exposure. The $P(D|E)$ and $P(D|E^c)$ in the sample do represent the conditional population probabilities. [56]

5.13. Case-Control Study (disease-based)

1. a simple random sample is taken from the population of units having the outcome and from the population of units without the outcome. [56]

Chapter 5. Sampling Plans

2. this way of sampling relates also directly to stratified sampling with the strata being the group with outcome (D) and the group without outcome (D^c). [56]
3. In each sample or stratum the exposure of each unit is noted. [56]
4. the probabilities $P(D)$ and $P(D^c)$ are known before sampling and are fixed in the sample. [56]
5. the observed probabilities in the sample are inappropriate as estimates for the same probabilities in the population. [56]
 - (a) we cannot estimate how many units in the population have the outcome. [56]
 - (b) we cannot estimate the joint probabilities $P(D \cap E)$, $P(D \cap E^c)$, $P(D \cap E)$, and $P(D \cap E)$ in the population from the sample. [56]
6. The problem with case-control studies is that the conditional probabilities $P(D|E)$ and $P(D|E^c)$ cannot be determined either. [56]

5.14. Important Notes

5.14.1. Haphazard Sampling VS Random Sampling [7]

Aspect	Haphazard Sampling	Random Sampling
Method	Non-systematic, no clear plan	Systematic, with equal chances for all
Bias	High due to human judgment	Low, minimal bias
Reproducibility	Difficult to reproduce	Easy to reproduce
Accuracy	Low accuracy and reliability	High accuracy and reliability
Use Case	Informal or exploratory research	Formal research, clinical trials

CHAPTER 6

PROBABILITY THEORY

6.1. Definitions

1. **Descriptive statistics** summarizes and visualizes the observed data. It is usually not very difficult, but it forms an essential part of reporting(scientific)results. [56]
2. **Inferential statistics** tries to draw conclusions from the data that would hold true for part or the whole of the population from which the data is collected. [56]
3. The **theory of probability** makes it possible to connect the two disciplines of descriptive and inferential statistics. [56]
- 4.
5. **Definition 6.1** (sample space (Ω)). The sample space is the set of all possible outcomes of the experiment, usually denoted by Ω . [49]
- 6.
7. **Definition 6.2** (event space (\mathcal{A})). The event space is the space of potential results of the experiment. A subset A of the sample space Ω is in the event space \mathcal{A} if at the end of the experiment we can observe whether a particular outcome $\omega \in \Omega$ is in A . The event space \mathcal{A} is obtained by considering the collection of subsets of Ω , and for discrete probability distributions \mathcal{A} is often the power set of Ω . [49]
8. An **event** is defined as something that happens or it is seen as a result of something. [56]
9. **Definition 1:** probability of an event A for a finite population can be given by $P(A) = \frac{N_A}{N}$ with N_A the number of units with characteristic A and N the size of the population. [56]
 - (a) This definition is only correct if each opportunity for the event to occur is as likely to produce the event as any other opportunity. [56]
 - (b) Another limitation of this definition is that it is defined for finite populations only. [56]
10. **Definition 2** (more theoretical): Probability is the proportion of the occurrence of an event obtained from infinitely many repeated and identical trials or experiments under similar conditions. [56]
 - (a) **Example:** if a die is thrown n times and the event A is the single dot facing up, then the probability $P(A)$ of the event A can be **approximated** by the ratio of the number of throws n_A with a single dot facing up and the total number of throws n , i.e., $P(A) \approx \frac{n_A}{n}$. [56]
 - (b) When the number of repeated trials n is increased it is expected that the proportion $\frac{n_A}{n}$ converges to some value p (which would be equal to $1/6$ if the die is *fair*). [56]
 - (c) $P(A) = \lim_{n \rightarrow \infty} \frac{n_A}{n} = p$ [56]
11. **Definition 3:** With each event $A \in \mathcal{A}$, we associate a number $P(A)$ that measures the probability or degree of belief that the event will occur. $P(A)$ is called the probability of A . [49]
12. we simply define the probability $P(A)$ of an event A as an (unknown) value between zero and one, $0 \leq P(A) \leq 1$, where both boundaries are allowed, which could either be approximated by collecting appropriate and real data or by the limit of a proportion of repeated and identical trials. [56]
- 13.

Definition 6.3 (Observed probabilities). Observed probabilities are empirical probabilities — that is, probabilities calculated from the sample data.

- (a) These probabilities are **not theoretical**, but come directly from real-world observations.

12. interpretations of probability:

(a) **Frequentist Interpretation (Classical View)**

- i. Probability is the **long-run frequency** of an event occurring, based on **repeated trials** under identical conditions.

ii. **Key Features:**

- A. Probability is objective and based on data.
- B. An event's probability is defined by the limit of its relative frequency as the number of trials approaches infinity.
- C. Parameters (like the mean or proportion) are fixed but unknown.
- D. You don't assign probabilities to hypotheses or parameters.

(b) **Bayesian Interpretation**

- i. Probability is a measure of belief or **degree of certainty** about an event or statement, **given current knowledge**.

ii. **Key Features:**

- A. Probability is subjective and depends on prior knowledge.
- B. You can assign probabilities to hypotheses, events, and parameters.
- C. Uses Bayes' Theorem to update beliefs based on new evidence.
- D. A parameter can be treated as a random variable with its own probability distribution.

6.2. Event Axioms

1. The complement of event A is denoted by A^c and it indicates that event A does not occur. The probability that event A occurs is one minus the probability that the event A does not occur; thus $P(A) = 1 - P(A^c)$. This rule is based on the assumption that either event A occurs or event A^c occurs. This means that $P(A \cup A^c) = 1$, since we will see either A or A^c . [56]

2.

Definition 6.4 (joint event/ mutual event $(A \cap B)$). The occurrence of two events A and B at the same time is denoted by $A \cap B$. This is often referred to as the joint or mutual event. [56]

3. The event that either A or B (or both) occurs is denoted by $A \cup B$. [56]

4. The probability of no event must be zero. Not having events is indicated by the empty set \emptyset and the probability is $P(\emptyset) = 0$. [56]

5.

Definition 6.5 (mutually exclusive events $(P(A \cap B) = P(\emptyset) = 0)$). if two events A and B can never occur together (mutually exclusive events), then it follows that $A \cap B = \emptyset$ and $P(A \cap B) = P(\emptyset) = 0$. [56]

6.

Definition 6.6 (independent events $(A \perp B)$ $(P(A \cap B) = P(A) \cdot P(B))$). We call two events A and B independent if and only if the probability of the mutual event is equal to the product of the probabilities of each event A and B separately. Thus the independence of events A and B (denoted by $A \perp B$) is equivalent with $P(A \cap B) = P(A) \cdot P(B)$. [56]

- (a) any event A with the non-event \emptyset is independent: $P(A \cap \emptyset) = P(\emptyset) = 0 = 0 \cdot P(A) = P(\emptyset)P(A)$. [56]
- (b) if two events with a positive probability ($P(A) > 0$ and $P(B) > 0$) that are also mutually exclusive can never be independent: $0 = P(\emptyset) = P(A \cap B) < P(A)P(B)$. [56]

7.

Definition 6.7 (associated events/ associated variables). Two events or variables are considered associated when they are not independent.

6.3. Event Rules

1. If the events A and B are independent, then the events A and B^c , the events A^c and B , and the events A^c and B^c are also independent. [56]
 2. The probability of the occurrence of either event A or B or both is equal to the sum of the probabilities of these events separately minus the probability that both events occur at the same time, i.e., $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. [56]
 3. For mutually exclusive events A and B , $P(A \cup B) = P(A) + P(B)$. [56]
 - 4.
- Definition 6.8** (law of total probability ($P(A) = P(A \cap B) + P(A \cap B^c)$)). The probability of an event A is the sum of the probability of both events A and B and the probability of both events A and B^c , thus $P(A) = P(A \cap B) + P(A \cap B^c)$. [56]

6.4. Conditional Probability ($P(A|B)$)

1. In some situations probability statements are of interest for a particular subset of outcomes. [56]
2. If event A represents the occurrence of a specific condition or outcome, and event B represents the occurrence of a related or influencing condition or characteristic, then the probability of interest is the conditional probability of A given B , denoted by $P(A|B)$. We refer to this conditional probability as the probability of event A given event B . [56]
3.
$$P(A|B) = \begin{cases} \frac{P(A \cap B)}{P(B)} & \text{if } P(B) > 0 \\ 0 & \text{if } P(B) = 0 \end{cases}$$
 [56]
4. if event B could never occur (i.e., $P(B) = 0$), there is no reason to define the conditional probability [56]
5. If we deal with two events A and B , the relevant probabilities can be summarized in the a 2×2 contingency table. In column A and row B , the probability of the occurrence of both events A and B at the same time is given by $P(A \cap B)$. Using the conditional relation, it can also be expressed by $P(A|B)P(B)$. [56]

	A	A^c	
B	$P(A \cap B) = P(A B)P(B)$	$P(A^c \cap B) = P(A^c B)P(B)$	$P(B)$
B^c	$P(A \cap B^c) = P(A B^c)P(B^c)$	$P(A^c \cap B^c) = P(A^c B^c)P(B^c)$	$P(B^c)$
	$P(A)$	$P(A^c)$	1

Table 6.1: Conditional probabilities in a 2×2 contingency table [56]

6. **Marginal Probability:** Marginal Probability refers to the probability of a single event occurring, without consideration of any other events. It is derived from a joint probability distribution and represents the likelihood of an event happening in isolation. [58]

$$(a) P(X = x) = \begin{cases} \sum_y P(X = x, Y = y) & \text{discrete} \\ \int_{-\infty}^{\infty} f_{XY}(x, y) dy & \text{continuous} \end{cases} \quad [58]$$

$$(b) P(Y = y) = \begin{cases} \sum_x P(X = x, Y = y) & \text{discrete} \\ \int_{-\infty}^{\infty} f_{XY}(x, y) dx & \text{continuous} \end{cases} \quad [58]$$

7. If we consider a change of variables $x = g(y)$, then a function $f(x)$ becomes $\tilde{f}(y) = f(g(y))$. Now consider a probability density $P_x(x)$ that corresponds to a density $P_y(y)$ with respect to the new variable y , where the suffices denote the fact that $P_x(x)$ and $P_y(y)$ are different densities. Observations falling in the range $(x, x + \delta x)$ will, for small values of δx , be transformed into the range $(y, y + \delta y)$ where $P_x(x)\delta x \simeq P_y(y)\delta y$, and hence: [41]

$$P_y(y) = P_x(x) \left| \frac{dx}{dy} \right| = P_x(g(y)) |g'(y)| \quad [41]$$

6.5. Bayesian Probabilities/ Bayesian Statistics

1.

Theorem 6.1 (Bayes Theorem).

$$\underbrace{P(B|A)}_{\text{posterior}} = \frac{P(A \cap B)}{P(A)} = \frac{\overbrace{P(A|B)P(B)}^{\text{likelihood prior}}}{\underbrace{P(A)}_{\text{evidence}}} \quad [49, 56]$$

(a) **Posterior ($P(B|A)$):**

- i. **Meaning:** The probability of B given that you have observed A . [7]
- ii. **Why it's important:** This is what you're trying to compute or update — your updated belief about B after seeing evidence A . [7]
- iii. It is the quantity of interest in Bayesian statistics because it expresses exactly what we are interested in, i.e., what we know about B after having observed A . [49]

(b) **Likelihood ($P(A|B)$):**

- i. **Meaning:** The probability of seeing evidence A assuming B is true. [7]
- ii. **Why it's important:** It reflects how compatible the data is with different hypotheses. [7]
- iii. **Note:** This is not a probability of B , but rather of the data A under different conditions of B . [7]
- iv. describes how B and A are related, and in the likelihood case of discrete probability distributions, it is the probability of the data A if we were to know the latent variable B . [49]
- v. The likelihood is sometimes also called the “**measurement model**”. [49]
- vi. Note that the likelihood is not a distribution in B , but only in A . We call $P(A|B)$ either the “likelihood of B (given A)” or the “probability of A given B ” but never the “likelihood of A ”. [49]

(c) **Prior ($P(B)$):**

- i. **Meaning:** The initial belief or probability of B before seeing any evidence. [7]
- ii. **Why it's important:** It encodes your assumptions or background knowledge. [7]

- iii. Encapsulates our subjective prior prior knowledge of the unobserved (latent) variable B before observing any data. [49]
- iv. We can choose any prior that makes sense to us, but it is critical to ensure that the prior has a nonzero pdf (or pmf) on all plausible B , even if they are very rare. [49]
- v. The specification of the prior can be tricky for two reasons: [49]
 - A. the prior should encapsulate our knowledge about the problem before we see any data. This is often difficult to describe. [49]
 - B. it is often not possible to compute the posterior distribution analytically. [49]

(d) **Evidence / Marginal Likelihood ($P(A)$):**

- i. $P(A) = \int P(A|B)P(B)dB = \mathbb{E}_B[P(A|B)]$ [49]
- ii. **Meaning:** The total probability of observing the data A , considering all possible values of B . [7]
- iii. **Why it's important:** It acts as a normalizing constant so that the posterior sums (or integrates) to 1. [7]
- iv. **Why it's not a prior:** Even though A appears in $P(B|A)$, in the context of Bayes' theorem, it's the observed data, and not a belief about a variable. That's why it's called the evidence. [7]
- v. the marginal likelihood is independent of B , and it ensures that the posterior $P(B|A)$ is normalized. [49]
- vi. The marginal likelihood can also be interpreted as the expected likelihood where we take the expectation with respect to the prior $P(B)$. [49]
- (e) Bayes' theorem allows us to invert the relationship between B and A given by the likelihood. Bayes' theorem is also called the "**probabilistic inverse.**" [49]

2. In Bayesian statistics the prominence of Bayes' rule is much greater than it is in Frequentist statistics. [56]

- (a) In Bayesian statistics probabilities are not solely regarded as long-run frequencies, but rather, they are regarded as a tool to quantify the "degree of belief" one holds in relation to a specific event. [56]
- (b) In this framework, probabilities can be regarded as an extension to boolean logic (in which statements/events are true or false, 0 or 1), to deal with events that have some probability. [56]
- (c) Bayes rule is subsequently used to update the degree of belief regarding an event in the face of new evidence: i.e., it is used to update $P(A)$ in the formula above to include the evidence provided by $P(B)$, the result of which is expressed as $P(A|B)$. [56]
- (d) This approach takes on a very powerful meaning when we replace $P(A)$ by $P(\boldsymbol{\theta})$ (i.e., the distribution of the parameters) and $P(B)$ by $P(D)$ where D relates to the data we observe in (e.g.,) a scientific study.
- (e) In this interpretation, Bayes' rule provides us with a means of updating our belief regarding a population parameter ($P(\boldsymbol{\theta})$) when we observe sample data ($P(D)$).

3. **maximum likelihood:** \mathbf{w} is set to the value that maximizes the likelihood function $P(D|\mathbf{w})$.

- (a) This corresponds to choosing the value of \mathbf{w} for which the probability of the observed data set is maximized. [41]
- (b) One common criterion for determining the parameters in a probability distribution using an observed data set is to find the parameter values that maximize the likelihood function. This might seem like a strange criterion because, from our foregoing discussion of probability theory, it would seem more natural to maximize the probability of the parameters given the data, not the probability of the data given the parameters. [41]

- (c) The negative log of the likelihood function is called an **error function**. Because the negative logarithm is a monotonically decreasing function, maximizing the likelihood is equivalent to minimizing the error. [41]
4. These are not events that can be repeated numerous times in order to define a notion of probability. [41]
5. Bayes' theorem is used to convert a prior probability into a posterior probability by incorporating the evidence provided by the observed data. [41]
6. We capture our assumptions about \mathbf{w} , before observing the data, in the form of a prior probability distribution $P(\mathbf{w})$. The effect of the observed data $D = \{t_1, \dots, t_N\}$ is expressed through the conditional probability $P(D|\mathbf{w})$. Bayes' theorem, which takes the form $P(\mathbf{w}|D) = \frac{P(\mathbf{w})P(D|\mathbf{w})}{P(D)}$ then allows us to evaluate the uncertainty in \mathbf{w} after we have observed D in the form of the posterior probability $P(\mathbf{w}|D)$. [41]
- (a) The quantity $P(D|\mathbf{w})$ on the right-hand side of Bayes' theorem is evaluated for the observed data set D and can be viewed as a function of the parameter vector \mathbf{w} , in which case it is called the **likelihood function**. It expresses how probable the observed data set is for different settings of the parameter vector \mathbf{w} . [41]
- (b) The denominator ($P(D)$) is the normalization constant, which ensures that the posterior distribution on the left-hand side is a valid probability density and integrates to one. [41]
- $$P(D) = \int P(D|\mathbf{w})P(\mathbf{w}) d\mathbf{w} \quad [41]$$
- (c) Likelihood is not a probability distribution over \mathbf{w} , and its integral with respect to \mathbf{w} does not (necessarily) equal one. [41]
- (d) We can state Bayes' theorem in words: *posterior \propto likelihood \times prior* where all of these quantities are viewed as functions of \mathbf{w} . [41]
7. One common criticism of the Bayesian approach is that the prior distribution is often selected on the basis of mathematical convenience rather than as a reflection of any prior beliefs. [41]
8. Reducing the dependence on the prior is one motivation for so-called noninformative priors. [41]
- 9.

Definition 6.9 (Conjugate Prior). A prior is **conjugate** for the likelihood function if the posterior is of the same form/type as the prior. [49]

- (a) Conjugacy is particularly convenient because we can algebraically calculate our posterior distribution by updating the parameters of the prior distribution. [49]
- (b) When considering the geometry of probability distributions, conjugate priors retain the same distance structure as the likelihood. [49]

Likelihood	Conjugate prior	Posterior
Bernoulli	Beta	Beta
Binomial	Beta	Beta
Gaussian	Gaussian/inverse Gamma	Gaussian/inverse Gamma
Gaussian	Gaussian/inverse Wishart	Gaussian/inverse Wishart
Multinomial	Dirichlet	Dirichlet

Examples of conjugate priors for common likelihood functions. [49]

6.5.1. Bayes' Law for Statistical Models

- we are interested in estimating (or otherwise making decisions about) the value of some population parameter θ by computing $\hat{\theta}$. [56]

2. One fairly general method of computing $\hat{\theta}$ was choosing the value of θ that **maximized the likelihood** $\ell(\theta)$ where, when considering a dataset of n i.i.d. observations, we had [56]

$$\ell(\theta) = \ell(\theta|X_1, X_2, X_3, \dots, X_n) = P(X_1|\theta)P(X_2|\theta) \cdots P(X_n|\theta) = P(D|\theta) \quad [56]$$

3. Bayes theorem in a way provides an alternative to for example Maximum Likelihood Estimation by—opposed to picking the value of θ that maximizes $P(D|\theta)$ —allowing one to quantify the probability of a large number of possible values of θ_i after observing the data: [56]

$$P(\theta_i|D) = \frac{Pr(D|\theta_i)Pr(\theta_i)}{\sum_{i=1}^k P(D|\theta_i)P(\theta_i)} = \frac{\ell(\theta_i|X_1, X_2, X_3, \dots, X_n)P(\theta_i)}{\sum_{i=1}^k \ell(\theta_i|X_1, X_2, X_3, \dots, X_n)P(\theta_i)} \quad [56]$$

4. It isn't a large stretch to specify the prior probability $P(\theta_i)$ off each hypothesis i using a Probability Mass Function (PMF). This highlights that Bayes' Theorem can be used not just for events, but for random variables. The general form of Bayes' Theorem that is most often encountered in Bayesian statistics generalizes the ideas presented above to random variables as these are described by their distribution functions (i.e., their PMFs or PDFs): [56]

$$f(\theta|D) = \frac{\ell(\theta) f(\theta)}{\int \ell(\theta) f(\theta) d\theta} \quad [56]$$

This expression above effectively describes that our posterior beliefs regarding the parameter of interest — i.e., our belief after observing the data — can be quantified using a (potentially continuous) PDF $f(\theta|D)$ which itself is obtained by multiplying the likelihood $\ell(\theta) = P(D|\theta)$ by our prior belief regarding the parameters $f(\theta)$ and dividing by the so-called marginal likelihood $\int \ell(\theta) f(\theta) d\theta$. The latter simply acts as a Normalizing constant to make sure that $f(\theta|D)$ is a valid PDF. [56]

5. when \mathcal{F} is a class of sampling distribution functions, and \mathcal{P} is a class of prior distribution functions, then the class \mathcal{P} is called **conjugate** for \mathcal{F} if $P(\theta|x) \in \mathcal{P}$ for all $P(\cdot|\theta) \in \mathcal{F}$ and $P(\cdot) \in \mathcal{P}$. [56]
- (a) This definition is cumbersome, since if we choose \mathcal{P} as the class of all distribution functions, then \mathcal{P} is always conjugate. [56]
 - (b) However, in practice we are most interested in **natural conjugate prior** families, which arise by taking \mathcal{P} to be the set of all densities having the same functional form as the sampling distribution (i.e., the likelihood). [56]

6.5.2. The Fundamentals of Bayesian Data Analysis

1. $f(\theta|D) = \frac{\ell(\theta) f(\theta)}{\int \ell(\theta) f(\theta) d\theta}$ is the main work-horse of Bayesian data analysis. [56]

2. Contrary to **Frequentist MLE estimation** where a single point $\hat{\theta}$ is often the end-point of the analysis (i.e., her or his best guess regarding the true population value), to a **Bayesian**, $f(\theta|D)$ effectively quantifies all that can be learned based on the observed data: it quantifies, using a PDF, our belief regarding the population value after seeing the data. [56]
3. Generally, the **steps** involved in conducting a Bayesian analysis are thus: [56]

- (a) Create a **sampling model**, i.e., specify the likelihood of the observed data as a function of the parameters θ . [56]
- (b) Specify a **prior distribution** over the parameters $f(\theta)$. Note that we will often encounter sampling models that involve multiple parameters and hence θ itself is a vector and $f(\theta)$ is a multivariate distribution (PMF or PDF). [56]
- (c) Compute $f(\theta|D)$, i.e., compute the **posterior distribution** of θ after seeing the data D . [56]

To a Bayesian, that's pretty much it; we have now updated our belief regarding the population parameters of interest based on the data. [56]

4. In practice, however, $f(\theta|D)$ is not always the final point of the analysis: often some summary of $f(\theta|D)$ is communicated (akin to the selection of the MLE estimate $\hat{\theta}$ that we have seen for frequentists analysis) such as its expected value $\mathbb{E}[\theta|D] = \int \theta f(\theta|D) d\theta$. [56]

6.5.3. Bayesian Decision-Making in Uncertainty

1. A Bayesian analysis will, after choosing a prior, allow you to compute (or at least sample from) the posterior $f(\theta|D)$. [56]
2. we would like to make probabilistic statements regarding populations values, or we would like to test specific hypotheses and make decisions. [56]

6.5.3.1. Providing Point Estimates of Parameters

1. After obtaining either an analytical expression for $f(\theta|D)$, or, as is more often the case in Bayesian analysis in practice, a way to obtain samples from this posterior PDF, we might want to provide a so-called point estimate of the population value in question. [56]
2. A point estimate is simply a single-number summary of the results obtained regarding a single parameter. [56]
3. The Frequentist analogy in this case would be providing (e.g.) a maximum likelihood estimate of a population value. [56]
4. In the Bayesian case providing point-estimates is conceptually very simple: we have already seen various ways of summarizing distributions using (e.g.) the expected value, the median, or the mode. [56]
5. When an analytical expression for $f(\theta|D)$ is available these can often readily be computed. [56]
6. The expected value is

$$\mathbb{E}[\theta] = \int_{\mathbb{R}} \theta f(\theta) d\theta \quad [56]$$

whereas the median is the value of m for which

$$\mathbb{E}[\theta] = \int_{-\infty}^m f(\theta) d\theta = \frac{1}{2} \quad [56]$$

and the mode is the (potentially local) maximum of the distribution function, i.e., where $f'(\theta) = 0$. [56]

7. the mode is, in the literature on Bayesian analysis, often referred to as the **Maximum A Posteriori** (or **MAP**) estimate. [56]

6.5.4. Frequentist AND/VS Bayesian

1. In both the Bayesian and frequentist paradigms, the likelihood function $P(D|\mathbf{w})$ plays a central role. [41]
2. In a frequentist setting, \mathbf{w} is considered to be a fixed parameter, whose value is determined by some form of ‘estimator’, and error bars on this estimate are obtained by considering the distribution of possible data sets D . [41]
3. From the Bayesian viewpoint there is only a single data set D (namely the one that is actually observed), and the uncertainty in the parameters is expressed through a probability distribution over \mathbf{w} . [41]
4. However, bias to prior lead to difficulties when comparing different models, and indeed Bayesian methods based on poor choices of prior can give poor results with high confidence. Frequentist evaluation methods offer some protection from such problems, and techniques such as cross-validation remain useful in areas such as model comparison. [41]

6.6. Expectations and covariances

$$\begin{array}{ll}
 \text{1. } \mathbb{E}[f] = \begin{cases} \sum_x P(x)f(x) & \text{discrete} \\ \int_{-\infty}^{\infty} P(x)f(x)dx & \text{continuous} \end{cases} & [41] \\
 \text{2. } \mathbb{E}[f] \simeq \frac{1}{N} \sum_{n=1}^N f(x_n) & [41] \\
 \text{3. } \mathbb{E}_x[f(x,y)] = \int f(x,y)P(x)dx & [41] \\
 \text{4. } \mathbb{E}_x[f(x,y)|y] = \int f(x,y)P(x|y)dx & [41]
 \end{array}
 \quad
 \begin{array}{ll}
 \text{5. } \text{var}[f] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] & [41] \\
 \text{6. } \text{var}[f] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 & [41] \\
 \text{7. } \text{Cov}[x, y] = \mathbb{E}_{x,y}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] & [41] \\
 & = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \\
 \text{8. } \text{Cov}[\mathbf{x}, \mathbf{y}] = \mathbb{E}_{\mathbf{x},\mathbf{y}}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{y}^\top - \mathbb{E}[\mathbf{y}^\top])] & [41] \\
 & = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\mathbf{xy}^\top] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^\top] \\
 \text{9. } \text{Cov}[x] \equiv \text{Cov}[x, x] & [41]
 \end{array}$$

6.7. Bootstrap

1. Suppose our original data set consists of N data points $\mathbf{X} = \{x_1, \dots, x_N\}$. We can create a new data set \mathbf{X}_B by drawing N points at random from \mathbf{X} , with replacement, so that some points in \mathbf{X} may be replicated in \mathbf{X}_B , whereas other points in \mathbf{X} may be absent from \mathbf{X}_B . This process can be repeated L times to generate L data sets each of size N and each obtained by sampling from the original data set \mathbf{X} . [41]
2. The statistical accuracy of parameter estimates can then be evaluated by looking at the variability of predictions between the different bootstrap data sets. [41]
3. One **advantage** of the Bayesian viewpoint is that the inclusion of prior knowledge arises naturally. [41]
4. For instance, that a fair-looking coin is tossed three times and lands heads each time. A classical maximum likelihood estimate of the probability of landing heads would give 1, implying that all future tosses will land heads! By contrast, a Bayesian approach with any reasonable prior will lead to a much less extreme conclusion. [41]

6.8. Association Measures/ Measures of Risk

1. D : the event of interest, such as a disease, failure, or success (also referred to as the outcome or result). [7, 56]
2. E : the event representing a possible influencing factor, such as exposure, a risk factor, or treatment (also called an explanatory variable). [7, 56]

Exposure	Outcome		TOTAL
	D	D^c	
E	$P(D E)$	$P(D^c E)$	$P(E)$
E^c	$P(D E^c)$	$P(D^c E^c)$	$P(E^c)$
TOTAL	$P(D)$	$P(D^c)$	1

Table 6.2: 2×2 contingency table for exposure-outcome

6.8.1. Risk Difference/ Excess Risk ($ER = P(D|E) - P(D|E^c)$)

- The risk difference or excess risk is an absolute measure of risk, since it is nothing more than the difference in the conditional probabilities, i.e. [56]

$$ER = P(D|E) - P(D|E^c) \quad [56]$$

- The risk difference is based on an additive model, i.e., $P(D|E) = ER + P(D|E^c)$. [56]
- It always lies between -1 and 1 . [56]

$$ER = 0 \iff P(D|E) = P(D|E^c)$$

$$\begin{aligned} &\iff P(E^c)P(D \cap E) = P(E)P(D \cap E^c) \\ 4. \quad &\iff [1 - P(E)]P(D \cap E) = P(E)[P(D) - P(D \cap E)] \\ &\iff P(D \cap E) = P(D)P(E) \end{aligned} \quad [56]$$

- it can be viewed as the excess number of cases (D) as a fraction of the population size. [56]
 - If the complete population (of size N) were to be exposed, the number of cases would be equal to $N \cdot P(D) = N \cdot P(D|E)$. [56]
 - If the complete population were unexposed the number of cases would be equal to $N \cdot P(D) = N \cdot P(D|E^c)$. [56]
 - Thus the difference in these numbers of cases indicates how the number of cases were to change if a completely exposed population would change to a completely unexposed population. [56]

$ER < 0$	exposure (E) is protective for the outcome [56]
$ER = 0$	the outcome (D) is independent of the exposure (E) [56]
$ER > 0$	there is a greater risk of the outcome when exposed (E) than when unexposed (E^c) [56]

6.8.2. Relative Risk ($RR = P(D|E)/P(D|E^c)$)

- The relative risk would compare the two conditional probabilities $P(D|E)$ and $P(D|E^c)$ by taking the ratio, i.e. [56]

$$RR = \frac{P(D|E)}{P(D|E^c)} \quad [56]$$

- It is common to take as denominator the risk of the outcome D for the unexposed group. [56]
- The relative risk is based on a multiplicative model, i.e., $P(D|E) = RR \cdot P(D|E^c)$.

$RR < 1$	unexposed group has a higher probability of the outcome [56]
$RR = 1$	outcome and exposure are independent [56]
$RR > 1$	exposed group has a higher probability of the outcome (D) than the unexposed one [56]

6.8.3. Odds Ratio ($OR = (P(D^c|E^c)/P(D^c|E)) \times RR$)

1.

Definition 6.10 (Odds). The odds is a measure of how likely the outcome occurs with respect to not observing this outcome. The odds comes from gambling, where profits of bets are expressed as 1 to x . For instance, the odds of 1 to n means that it is n times more likely to loose than to win. The odds can be defined mathematically by $O = \frac{p}{(1-p)}$, with p the probability of winning. [56]

- The odds ratio compares the odds for the exposed group with the odds for the unexposed group. [56]

Chapter 6. Probability Theory

3. The odds of the exposed group is $O_E = \frac{P(D|E)}{1 - P(D|E)}$ and the odds for the unexposed group is $O_{E^c} = \frac{P(D|E^c)}{1 - P(D|E^c)}$. [56]

4. The odds ratio is given by:

$$OR = \frac{O_E}{O_{E^c}} = \frac{P(D|E)[1 - P(D|E^c)]}{P(D|E^c)[1 - P(D|E)]} = \frac{P(D^c|E^c)}{P(D^c|E)} \times RR \quad [56]$$

5. it is common to use the unexposed group as reference group, which implies that the odds of the unexposed group O_{E^c} is used in the denominator.

$OR < 1$	unexposed group has a higher probability of the outcome [56]
$OR = 1$	outcome is independent of the exposure [56]
$OR > 1$	exposed group has a higher odds than the unexposed group, which implies that the exposed group has a higher probability of outcome D [56]

6.8.4. Relation in ER, RR & OR

1. the odds ratio and relative risk are always ordered [56]
2. odds ratio is always further away from 1 than the relative risk [56]
.
$$1 < RR < OR \quad OR < RR < 1 \quad [56]$$
3. Proof:
 - (a) If $RR > 1$, we have that $P(D|E) > P(D|E^c)$, using its definition. [56]
 - (b) Since $P(D^c|E) = 1 - P(D|E)$ and $P(D^c|E^c) = 1 - P(D|E^c)$, we obtain that $P(D^c|E) < P(D^c|E^c)$. [56]
 - (c) Combining this inequality with the relation in Odds Ratio equation, we see that $OR > RR$. [56]
4. the odds ratio and relative risk are equal to each other when $RR = 1$ (or $OR = 1$). [56]
5. The odds ratio is often considered more complex than the relative risk, in particular because of the simplicity of interpretation of the relative risk. [56]
6. The odds ratio is, however, more frequently used in practice than the relative risk.
 - (a) An important reason for this is that the odds ratio is symmetric in exposure E and outcome D . If the roles of the exposure and outcome are interchanged the odds ratio does not change, but the relative risk does. [56]

6.9. Simpson's Paradox

1. Simpson's Paradox is when a trend or relationship that appears in separate groups of data reverses or disappears when the groups are combined. [7]
2. In other words:
 - (a) A conclusion that seems true when you look at each group individually [7]
 - (b) Can be completely misleading or even opposite when you combine the data. [7]
3. Why does it happen? Because of confounding variables — something else (like unequal group sizes) is influencing the results. [7]

4. Simpson demonstrated that the association between D and E in the collapsed contingency table is preserved in the two separate contingency tables for C and C^c whenever one or both of the following restrictions hold true:

$$P(D \cap E \cap C)P(D \cap E^c \cap C^c) = P(D \cap E^c \cap C)P(D \cap E \cap C^c) \quad [56]$$

$$P(D \cap E \cap C)P(D^c \cap E \cap C^c) = P(D^c \cap E \cap C)P(D \cap E \cap C^c) \quad [56]$$

- (a) The first equation implies that the odds ratio for having the exposure E for the presence or absence of C in the outcome group D is equal to one, i.e. [56]

$$OR_{EC|D} = \frac{P(E|C,D)[1 - P(E|C^c,D)]}{P(E|C^c,D)[1 - P(E|C,D)]} = 1 \quad [56]$$

Thus E and C must be independent in the outcome group D , which means that $P(E \cap C|D) = P(E|D)P(C|D)$. [56]

- (b) The second equation implies that the odds ratio for the outcome D in the presence or absence of C in the exposed group E is equal to one, i.e. [56]

$$OR_{DC|E} = \frac{P(D|C,E)[1 - P(D|C^c,E)]}{P(D|C^c,E)[1 - P(D|C,E)]} = 1 \quad [56]$$

Thus this means that D and C are independent in the exposed group E , which means that $P(D \cap C|E) = P(D|E)P(C|E)$. [56]

5. If the two independence requirements are violated, the event C is called a **confounder**. [56]

- (a) To say that C is not a confounder, we would want: [7]

- i. C is independent of E (i.e., the confounder is not related to the exposure): $C \perp E$ [7]
- ii. C is independent of D given E (i.e., once we account for exposure, the confounder does not affect disease): $C \perp D|E$ [7]

- (b) If C is related to both E and D — i.e., [7]

- i. C affects the likelihood of being exposed (E), and [7]
- ii. C also affects the chance of getting the disease (D) independently of E , [7]

Then the independence assumptions are violated. Therefore, C is a confounder — because it distorts the observed relationship between E and D . [7]

Example:

Exposure	Kidney Stones Outcome $\leq 2\text{cm}$ (C)		Total
	Removal (D)	No Removal (D^c)	
Nephrolithotomy (E)	234	36	270
Open Surgery(E^c)	81	6	87
Total	315	42	357

Exposure	Kidney Stones Outcome $> 2\text{cm}$ (C^c)		Total
	Removal (D)	No Removal (D^c)	
Nephrolithotomy (E)	55	25	80
Open Surgery(E^c)	192	71	263
Total	247	96	343

Table 6.3: 2×2 contingency table for removal of kidney stones and two surgical treatments by size of kidney stones. [56]

Exposure	Kidney Stones Outcome		Total
	Removal (D)	No Removal (D^c)	
Nephrolithotomy (E)	289	61	350
Open surgery (E^c)	273	77	350
Total	562	138	700

 Table 6.4: 2×2 contingency table for removal of kidney stones and two surgical treatments [56]

1. $RR = (289/350)/(273/350) = 1.0586$. This means that percutaneous nephrolithotomy increases the “risk” of successful removal of the kidney stones with respect to open surgery. [56]
2. $RR_{\leq 2} = 0.9309$ and $RR_{>2} = 0.9417$. it seems that open surgery has a higher success of kidney stone removal than percutaneous nephrolithotomy for both small and large stones. [56]
3. This is a contradiction. [56]

CHAPTER 7

RANDOM VARIABLES & DISTRIBUTIONS

7.1. Intro

1. A random variable is a numerical outcome of a random process. A random variable assigns a number to each possible outcome of a random experiment. [7]
2. An intuitive definition of a random variable or random quantity is a variable for which the value or outcome is unknown and for which the outcome is influenced by some form of random phenomenon. [56]
3. Why is it called "random"? Because the value it takes depends on chance. You don't know the outcome in advance, but you know the possible values and can assign probabilities to them. [7]
4. This allows us to extend our theory on probability to other types of data without restricting it to specific events (i.e., binary data). [56]
5. The probability sampling approach makes the variable of interest a random variable, as the sampling approach is here the random phenomenon. [56]
6. A realization or an outcome of the random variable is then indicated by the same, but lower case, letter. [56]
7. a random variable may be seen as a variable that can in principle be equal to any of the values in the population, and after probability sampling the outcome(s) will become known. [56]
8. Types of Random Variables:
 - (a) Discrete Random Variable:
 - i. Takes countable values (e.g., 0, 1, 2, 3...)
 - ii. Examples: number of heads in 10 coin tosses, number of emails received
 - (b) Continuous Random Variable
 - i. Takes uncountably many values in an interval (e.g., any real number)
 - ii. Examples: height of a person, temperature, time taken to run a race
9. Notation:
 - (a) Random variables are often written as capital letters like X, Y, Z
 - (b) Their values are written in lowercase, e.g., $X = x$
10. The **probability function** $P(X \leq x)$ is a general concept and can be used for any random variable. [56]
11. The PMF for a discrete random variable is the equivalent of the PDF for a continuous random variable. [56]
12. The probability $P(X \leq x)$ is also referred to as the **distribution function** obtained in x and it is denoted by $F(x) = P(X \leq x)$. Thus every random variable X has a distribution function F through $F(x) = P(X \leq x)$, but also every distribution function F has a random variable X , namely the random variable X that makes $P(X \leq x) = F(x)$. Thus the two concepts are directly related to each other and we then typically say that X is distributed according to F , i.e., $X \sim F$. [56]
13. Each distribution function F typically satisfies three conditions:

- (a) When the value x increases to infinity, the distribution function becomes equal to one, i.e.,
 $\lim_{x \rightarrow \infty} F(x) = 1$. [56]
- (b) When the value x decreases to minus infinity the distribution function becomes equal to zero, i.e.,
 $\lim_{x \rightarrow -\infty} F(x) = 0$. [56]
- (c) The distribution function is a non-decreasing function, i.e., $F(x_1) \leq F(x_2)$ when $x_1 \leq x_2$. [56]
14. Data points that are drawn independently from the same distribution (D OR $D(\dots)$) are said to be **independent and identically distributed**, which is often abbreviated to i.i.d. (denoted by $x \sim D$ OR $x \sim D(\dots)$) [41]
15. We consider multivariate random variables X as a finite vector of univariate random variables $[X_1 \dots X_D]^\top$. [49]
16. the desiderata (meaning: something that is needed or wanted) for manipulating probability distributions in the machine learning context: [49]
- (a) There is some “closure property” when applying the rules of probability, e.g., Bayes’ theorem.
By closure, we mean that applying a particular operation returns an object of the same type. [49]
 - (b) As we collect more data, we do not need more parameters to describe the distribution. [49]
17. There are three possible levels of **abstraction** we can have when considering distributions (of discrete or continuous random variables). [49]
- (a) At level one (the most concrete end of the spectrum), we have a particular **named distribution with fixed parameters**, for example a univariate Gaussian $\mathcal{N}(0, 1)$ with zero mean and unit variance. [49]
 - (b) In machine learning, we often use the second level of abstraction, that is, we fix the **parametric form** (the univariate Gaussian) and **infer the parameters from data**. For example, we assume a univariate Gaussian $\mathcal{N}(\mu, \sigma^2)$ with unknown mean μ and unknown variance σ^2 , and use a maximum likelihood fit to determine the best parameters (μ, σ^2) . [49]
 - (c) A third level of abstraction is to consider **families of distributions**. The univariate Gaussian is an example of a member of the exponential family. [49]

7.2. Joint Distribution Function ($F_{XY}, F_{X_1 X_2 \dots X_K}$)

1. The joint distribution function contains all the information on how the random variables are related to each other, i.e., how the random variables are dependent on each other. [56]
2. If one variable increases and the other variable also increases (on average) or if one variable increases while the other variable decreases (on average) they are said to **co-relate**. [56]
3. We say that random variables X and Y have a joint distribution function F_{XY} if the probability that X is observed in the interval $(-\infty, x]$ and the probability that Y is observed in the interval $(-\infty, y]$ is given by the function F_{XY} , i.e., $P(X \leq x, Y \leq y) = F_{XY}(x, y)$. The joint distribution function of two random variables is also called a **bivariate distribution function**. [56]
4. the joint distribution function of K random variables X_1, X_2, \dots, X_K , i.e., $F_{X_1 X_2 \dots X_K} = P(X_1 \leq x_1, X_2 \leq x_2, \dots, X_K \leq x_K)$ is called a **multivariate distribution function**. [56]

7.2.1. Independence of Random Variables

1. Two random variables X and Y are called independent when the bivariate distribution function is equal to the product of the marginal distribution functions. [56]

2. independence of X and Y holds when $F_{XY}(x,y) = F_X(x)F_Y(y)$ for all $(x,y) \in \mathbb{R} \times \mathbb{R}$, with F_X the CDF of X and F_Y the CDF of Y [56]
3. The random variables X_1, X_2, \dots, X_K are called **mutually independent** when the joint distribution function is the product of the marginal distribution functions, i.e., [56]

$$F_{X_1 X_2 \dots X_K}(x_1, x_2, \dots, x_K) = F_{X_1}(x_1)F_{X_2}(x_2) \dots F_{X_K}(x_K) \text{ for all } x_k \in \mathbb{R}.$$
 [56]
4. If in case we assume that all distribution functions are identical, i.e., $F_{X_1}(x) = F_{X_2}(x) = \dots = F_{X_K}(x) = F(x)$ for all x , then we have the concept of X_1, X_2, \dots, X_K being i.i.d. with distribution function F [56]
5. The random variables X and Y are called **dependent** when they are not independent. [56]
6. we may have that X and Y , X and Z , and Y and Z are **(pairwise) independent**, but X , Y , and Z are not mutually independent. Thus we may have $F_{XY}(x,y) = F_X(x)F_Y(y)$, $F_{XZ}(x,z) = F_X(x)F_Z(z)$, and $F_{YZ}(y,z) = F_Y(y)F_Z(z)$ for all x, y , and z , but we may not have $F_{XYZ}(x,y,z) = F_X(x)F_Y(y)F_Z(z)$ for all x, y , and z . [56]
7. Pairwise independence is thus **weaker** than mutual independence. When we talk about independence among multiple random variables we mean mutual independence. [56]
8. 2 types of dependencies/ independencies:
 - (a) Variables of same unit, (X_i, Y_i, \dots) , where i is unit. Example: height & weight of same person
 - (b) Different Units of sample/ population. Example: heights of siblings, hair color of a family, etc

7.3. Discrete Functions

7.3.1. Univariate PMF ($f(x_k) = p_k = P(X = x_k)$)

1. Discrete does not always mean that we observe values in \mathbb{N} . For instance, grades on a data science test may take values in $\{1, 1.5, 2.0, 2.5, \dots, 9.0, 9.5, 10\}$. Thus, it would be more rigorous to say that a discrete random variable X takes its values in the set $\{x_0, x_1, x_2, \dots, x_k, \dots\}$, with x_k an element of the real line ($x_k \in \mathbb{R}$) and with an ordering of the values $x_0 < x_1 < x_2 < \dots$. However, in many practical settings we can map this set to a subset of \mathbb{N} or to the whole set \mathbb{N} . [56]
2. For discrete random variables we can define $p_k = P(X = k)$ as the probability of observing the outcome k . [56]
3. This is referred to as the probability mass function (PMF) if the probabilities p_k satisfy two conditions. [56]
 - (a) all probabilities p_k should be nonnegative ($p_k \geq 0, \forall k$) [56]
 - (b) probabilities need to add up to one, i.e., $\sum_{k=0}^{\infty} p_k = 1$ [56]

7.3.2. CDF of Univariate PMF ($F(x)$)

1. The distribution function or cumulative density function (CDF) for a discrete random variable X is now given by
$$F(x) = P(X \leq x) = \sum_{k=0}^x f(k).$$
 [56]
2. If the set is $\{x_0, x_1, x_2, \dots, x_k, \dots\}$, with $x_0 < x_1 < x_2 < \dots$, then the CDF is defined as
$$F(x) = \sum_{k=0}^{m_x} f(x_k),$$
 with m_x the largest value for k that satisfies $x_k \leq x.$ [56]

7.3.3. Bivariate Joint PMF ($f_{XY}(x,y) = P(X = x, Y = y)$)

1. If X and Y are both discrete random variables, than the joint probability mass function (PMF) is $f_{XY}(x,y) = P(X = x, Y = y), x,y \in \mathbb{N}$ [56]

2. Some or many combinations of pairs (x,y) may not occur, which implies that the probability for these values is zero, i.e., $f_{XY}(x,y) = 0$. [56]

$$3. \sum_{(x,y) \in \mathbb{N} \times \mathbb{N}} f_{XY}(x,y) = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} f_{XY}(x,y) = \sum_{y=0}^{\infty} \sum_{x=0}^{\infty} f_{XY}(x,y) = 1 \quad [56]$$

4. Marginal PMFs:

$$(a) \text{ marginal PMF of } X: f_X(y) = \sum_{y=0}^{\infty} f_{XY}(x,y) \quad [56]$$

$$(b) \text{ marginal PMF of } Y: f_Y(y) = \sum_{x=0}^{\infty} f_{XY}(x,y) \quad [56]$$

5. When the random variables X and Y are **independent**, $f_{XY}(x,y) = f_X(x)f_Y(y)$. [56]

Proof:

$$\begin{aligned} f_{XY}(x,y) &= P(X = x, Y = y) \\ &= P(X \leq x, Y \leq y) - P(X \leq x, Y \leq y-1) - P(X \leq x-1, Y \leq y) + P(X \leq x-1, Y \leq y-1) \\ &= F_{XY}(x,y) - F_{XY}(x,y-1) - F_{XY}(x-1,y) + F_{XY}(x-1,y-1) \\ &= F_X(x)F_Y(y) - F_X(x-1)F_Y(y) - F_X(x)F_Y(y-1) + F_X(x-1)F_Y(y-1) \\ &= (F_X(x) - F_X(x-1))(F_Y(y) - F_Y(y-1)) \\ &= P(X = x)P(Y = y) = f_X(x)f_Y(y) \end{aligned} \quad [56]$$

7.3.4. CDF of Bivariate Joint PMF (F_{XY})

1. The joint CDF F_{XY} of the random variables X and Y is now provided in the same way as we did for single random variables $F_{XY}(x,y) = P(X \leq x, Y \leq y) = \sum_{k=0}^x \sum_{l=0}^y f_{XY}(k,l)$, for every $x,y \in \mathbb{N}$. [56]

7.4. Continuous Functions

7.4.1. Univariate PDF ($f(x) \neq P(X = x)$)

1. The density function may be viewed as a smooth version of the histogram if we standardize the frequencies on the vertical axis to proportions. [56]

2. The density function characterizes the occurrence of values for a specific variable (as depicted on the x-axis) on all units from the population. [56]

3. Since in practice all populations are finite, the density function is an abstract formulation of, or a “model” for, the “frequencies” of all population values. [56]

4. PDF must satisfy two important conditions or properties: [56]

(a) it cannot be negative, i.e., $f(x) \geq 0$ for every value x that is present in the population. For values of x outside this domain, the PDF can then be defined equal to zero: $f(x) = 0$. [56]

(b) the “area under the curve” (AUC) is equal to one, i.e., $\int_{\mathbb{R}} f(x)dx = 1$ [56]

5. Many different PDFs exist and they have been proposed over a period of more than two centuries to be able to describe populations and data in practical settings. [56]

Chapter 7. Random Variables & Distributions

- (a) These functions are often parametric functions, i.e., the PDF is known up to a set of parameters. [56]
- (b) The PDF is then often denoted by f_{θ} , where θ represents the set or **vector** of m density parameters: $\theta = [\theta_1 \ \theta_2 \ \dots \ \theta_m]^T$. [56]
- 6. $f(x)$ is **not** be equal to $P(X = x)$ [56]
 - (a) The probability $P(X = x)$ is equal to zero for continuous random variables, since there is no surface area under $f(x)$. [56]

7.4.2. CDF of Univariate PDF ($F(X)$)

1. There is a direct relation between distribution functions and densities. If we start with a PDF, we can define a distribution function in the following way: $F(x) = P(X \leq x) = \int_{-\infty}^x f(z) dz$ [56]
2. The full distribution function of $\psi(X)$ can always be established, but it does not always have a simple workable form. [56]

$$P(\psi^{-1}(X) \leq x) = P(X \leq \psi(x)) = F(\psi(x)) = \int_{-\infty}^{\psi(x)} f(z) dz = \int_{-\infty}^x \psi'(z) f(\psi(z)) dz \quad [56]$$

The calculations now show that the distribution function of random variable $\psi^{-1}(X)$ is equal to $F(\psi(x))$ and the PDF is $\psi'(x) f(\psi(x))$. [56]

Example: Let X being normally distributed with parameters μ and σ , and consider the function $\psi(x) = \exp\{x\}$ [56]

$$P(X \leq x) = \int_{-\infty}^x \frac{1}{\sigma} \phi\left(\frac{z-\mu}{\sigma}\right) = \int_{-\infty}^{(x-\mu)/\sigma} \phi(z) dz = \Phi\left(\frac{x-\mu}{\sigma}\right) \quad [56]$$

$$\begin{aligned} P(\exp\{X\} \leq x) &= P(X \leq \log(x)) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right) \\ &= \int_{-\infty}^{\log(x)} \frac{1}{\sigma} \phi\left(\frac{z-\mu}{\sigma}\right) dz = \int_0^x \frac{1}{z\sigma} \phi\left(\frac{\log(z)-\mu}{\sigma}\right) dz \end{aligned} \quad [56]$$

3. $P(X < x) = P(X \leq x)$ for continuous random variables

4. $P(X > x) = 1 - P(X \leq x)$ [56]

5. $P(x_1 < X \leq x_2) = P(X \leq x_2) - P(X \leq x_1)$ [56]

7.4.3. Bivariate Joint PDF

1. $\exists x, y$ such that $f_{XY}(x, y) > P(X = x, Y = y) = 0$ [56]
2. Marginal PDFs:

- (a) marginal PDF of X : $f_X(y) = \int_{-\infty}^{\infty} f_{XY}(x, y) dy$ [56]

- (b) marginal PDF of Y : $f_Y(y) = \int_{-\infty}^{\infty} f_{XY}(x, y) dx$ [56]

7.4.4. CDF of Bivariate Joint PDF (F_{XY})

1. The joint CDF F_{XY} of the random variables X and Y is now provided in the same way as we did for single random variables $F_{XY}(x, y) = P(X \leq x, Y \leq y) = \int_{-\infty}^x \int_{-\infty}^y f_{XY}(u, v) du dv$, for every $x, y \in \mathbb{N}$. [56]

2. if we consider any set $A \subset \mathbb{R} \times \mathbb{R}$ and define the probability that $(X, Y) \in A$ by $P((X, Y) \in A) = \iint_A f_{XY}(u, v) du dv$ [56]

3. Marginal CDFs:

$$(a) \text{ marginal CDF of } X: F_X(y) = \int_{-\infty}^y f_X(t) dt \quad [56]$$

$$(b) \text{ marginal CDF of } Y: F_Y(y) = \int_{-\infty}^y f_Y(t) dt \quad [56]$$

$$(c) \text{ OR } F_X(x) = \lim_{y \rightarrow \infty} F_{XY}(x, y) \text{ and } F_Y(y) = \lim_{x \rightarrow \infty} F_{XY}(x, y) \quad [56]$$

7.5. Sufficient Statistics

1. statistic of a random variable is a deterministic function of that random variable [49]
2. the idea that there are statistics that will contain all available information that can be inferred from data corresponding to the distribution under consideration. [49]
3. sufficient statistics carry all the information needed to make inference about the population, that is, they are the statistics that are sufficient to represent the distribution. [49]
As we observe more data, do we need more parameters θ to describe the distribution? It turns out that the answer is yes in general. [49]
4. For a set of distributions parametrized by θ , let X be a random variable with distribution $P(x|\theta)$ given an unknown θ_0 . A vector $\phi(x)$ of statistics is called sufficient statistics for θ_0 if they contain all possible information about θ_0 . To be more formal about “contain all possible information”, this means that the probability of x given θ can be factored into a part that does not depend on θ , and a part that depends on θ only via $\phi(x)$. [49]

5.

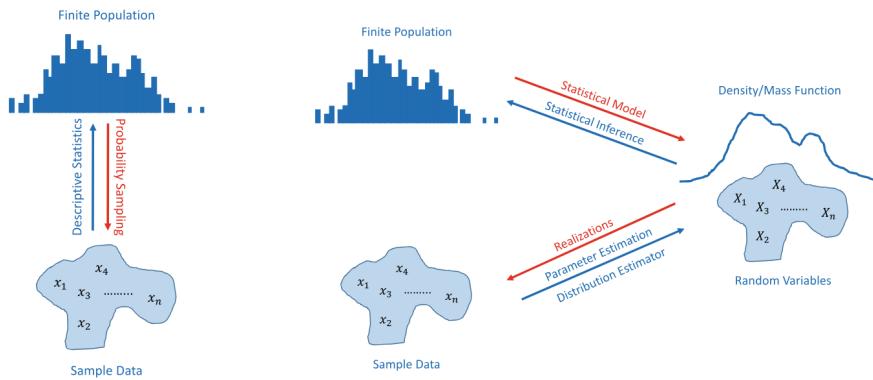
Theorem 7.1 (Fisher-Neyman (sufficient statistics)). Let X have probability density function $P(x|\theta)$. Then the statistics $\phi(x)$ are sufficient for θ if and only if $P(x|\theta)$ can be written in the form: [49]

$$P(x|\theta) = h(x)g_\theta(\phi(x)) \quad [49]$$

where $h(x)$ is a distribution independent of θ and g_θ captures all the dependence on θ via sufficient statistics $\phi(x)$. [49]

- (a) If $P(x|\theta)$ does not depend on θ , then $\phi(x)$ is trivially a sufficient statistic for any function ϕ . [49]
- (b) The more interesting case is that $P(x|\theta)$ is dependent only on $\phi(x)$ and not x itself. In this case, $\phi(x)$ is a sufficient statistic for θ . [49]

7.6. Sample Statistics (T_n), Estimators & Summary Statistics



we use the theory of random variables and distribution functions allows us to make more detailed statements about the population of interest [56]

1. Sample statistics are numerical values calculated from a sample (a subset of a population) to describe some aspect of the sample. [7]
2. An **estimator** is a rule or formula (usually a function of sample data) used to estimate a population parameter. [7]
3. These are statistics that summarize or describe features of a dataset. They can be from either a sample or an entire population. [7]
4. We are often interested in summarizing sets of random variables and comparing pairs of random variables. [49]
5. A **statistic of a random variable** is a deterministic function of that random variable. [49]
6. The summary statistics of a distribution provide one useful view of how a random variable behaves, and provides numbers that summarize and characterize the distribution. [49]
7. Going from the data to the PDF or PMF to the population is also called statistical inference, but now we are using (parametric) statistical models. [56]

8. Population Characteristics:

$$(a) \text{ standardized variable: } Z = \frac{X - \mu(f)}{\sigma(f)} \quad [56]$$

$$(b) \text{ population mean: } \mu(f) = \mathbb{E}[X] = \int_{\mathbb{R}} xf(x)dx \quad [56]$$

$$(c) \text{ population variance: } \sigma^2(f) = \mathbb{E}[(X - \mu(f))^2] = \int_{\mathbb{R}} (x - \mu(f))^2 f(x)dx \quad [56]$$

$$(d) \text{ population standard deviation: } \sigma(f) = \sqrt{\sigma^2(f)} \quad [56]$$

$$(e) \text{ Third moment (skewness): } \gamma_1(f) = \mathbb{E}[(Z)^3] \quad [56]$$

$$(f) \text{ Fourth moment (excess kurtosis): } \gamma_2(f) = \mathbb{E}[(Z)^4] - 3 \quad [56]$$

$$(g) \text{ Quantiles: } F(x_p(f)) = \int_{-\infty}^{x_p(f)} f(x)dx = p \Rightarrow x_p(f) = F^{-1}(p) \quad [56]$$

F^{-1} is the inverse distribution function [56]

Note: we are now using the notation $\mu(f)$ to make explicit that the population mean μ will depend on our choice of f . [56]

7.6.1. Distributions of Sample Statistic T_n

1. **PMF or PDF** [56]

- (a) In many settings, the sample distribution function has a sample PMF or PDF f_{T_n} , often referred to as the sample density function of T_n . [56]
 - (b) it is not always easy to determine the sample distribution or sample density function in general [56]
 - (c) Only in special cases is it possible to determine closed-form functions. [56]
2. **CDF:** $F_{T_n}(x) = P(T_n \leq x)$ [56]
- (a) This CDF is often referred to as the sample distribution function of statistic T_n , as it describes how the sample statistics vary if we draw (new) samples from the population. [56]
3. We study the distributions of T_n , as these distribution functions effectively allow us to examine the quality of our estimators: the distribution function of the random variable T_n provides a measure of how well an estimator approximates a population value.
- (a) The expected value of an estimator $\mathbb{E}[T_n]$ is a measure for the central tendency of a sample statistic [56]
 - (b) The standard deviation of T_n provides a measure for the variability of a sample statistic. The standard deviation of a sample statistic T_n is the standard error of that sample statistic; the standard error of a sample statistic of interest is commonly reported in scientific texts to quantify the uncertainty associated with the sample statistic. [56]

7.6.1.1. Distribution of the Sample Minimum

1. $F_{X_{(1)}}(x) = 1 - [1 - F(x)]^n$ [56]
2. $f_{X_{(1)}}(x) = n[1 - F(x)]^{n-1} f(x)$ [56]

7.6.1.2. Distribution of the Sample Maximum

1. Let T_n be the maximum of X_1, X_2, \dots, X_n , the distribution function of T_n is given by: [56]

$$F_{X_{(n)}}(x) = P(X_{(n)} \leq x) = P(\max\{X_1, X_2, \dots, X_n\} \leq x)$$

$$= P(X_1 \leq x, X_2 \leq x, \dots, X_n \leq x) = \prod_{i=1}^n P(X_i \leq x) = \prod_{i=1}^n F(x)$$

$$= [F(x)]^n$$
2. $f_{X_{(n)}}(x) = n[F(x)]^{n-1} f(x)$ [56]

7.6.1.3. Distribution of the Sample Average

1. The distribution function of the sample average $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ is not so easy to determine in general. [56]
2. The p -th moment of a general sample statistic T_n is given by $\mathbb{E}[T_n^p] = \int_{\mathbb{R}} t^p f_{T_n}(t) dt$, if the sample density f_{T_n} exists, using the definition of moments for any random variable. [56]
3. It can, however, also be calculated in a different way, using the population density f and the fact that X_1, X_2, \dots, X_n are i.i.d. F . The p -th moment of T_n is: [56]

$$\mathbb{E}[T_n^p] = \int_{\mathbb{R}} t^p f_{T_n}(t) dt = \mathbb{E}[T_n^p(X_1, X_2, \dots, X_n)]$$

$$= \int_{\mathbb{R}^n} T_n^p(x_1, x_2, \dots, x_n) f(x_1)f(x_2)\cdots f(x_n) dx_1 dx_2 \cdots dx_n$$
4. A special case is the first moment $\mu(f_{T_n}) = \mathbb{E}[T_n]$ [56]
5. The advantage of $\mathbb{E}[T_n^p]$ equation is that the moments of the sample statistic $T_n = \bar{X}$ can be expressed in moments of the random variables X_1, X_2, \dots, X_n . Indeed, the first moment of \bar{X} is now given by [56]

$$\begin{aligned}
 \mu(f_{\bar{X}}) &= \mathbb{E}[\bar{X}] = \int_{\mathbb{R}^n} \left(\frac{1}{n} \sum_{i=1}^n x_i \right) f(x_1) f(x_2) \cdots f(x_n) dx_1 dx_2 \cdots dx_n \\
 &= \frac{1}{n} \sum_{i=1}^n \int_{\mathbb{R}^n} x_i f(x_1) f(x_2) \cdots f(x_n) dx_1 dx_2 \cdots dx_n = \frac{1}{n} \sum_{i=1}^n \int_{\mathbb{R}} x_i f(x_i) dx_i \\
 &= \frac{1}{n} \sum_{i=1}^n \mathbb{E}[X_i] = \mu(f)
 \end{aligned} \tag{56}$$

6. The p -th central moment of T_n is given by $\mathbb{E}[(T_n - \mu(f_{T_n}))^p]$ [56]

(a) The second central moment is the variance of the sample statistic T_n . [56]

$$\mathbb{E}[(\bar{X} - \mu(f))^2] = \frac{\sigma^2(f)}{n} \tag{56}$$

(b) Taking the square root second central moment, we obtain the standard deviation of T_n . [56]

7. The sample average has an expectation of $\mu(f)$ and a variance of $\frac{\sigma^2(f)}{n}$, irrespective of the population density f . In other words, the sample average \bar{X} is an appropriate estimator for the population mean $\mu(f)$, and it has a standard error (SE) that is a factor \sqrt{n} smaller than the standard deviation of the population, i.e. the standard error is $SE(\bar{X}) = \frac{\sigma(f)}{\sqrt{n}}$. Note that this standard error is typically unknown, since it depends on the unknown population standard deviation $\sigma(f)$. This unknown parameter can also be estimated from the sample data. [56]

8. The skewness is given by $\gamma_1(f_{\bar{X}}) = \frac{\gamma_1(f)}{\sqrt{n}}$ and the excess kurtosis is given by $\gamma_2(f_{\bar{X}}) = \frac{\gamma_2(f)}{n}$. Thus the skewness and excess kurtosis are close to zero when the sample size is getting large. [56]

7.6.1.4. Distribution of the Sample Variance

1. The distribution function of the sample variance $T_n = S^2 \equiv \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$ is in general unknown, but we are able to determine a few moments. [56]

2. we can rewrite the variance first into $S^2 = \frac{1}{n-1} \left(\sum_{i=1}^n (X_i - \mu(f))^2 \right) - \frac{n(\bar{X} - \mu(f))^2}{n-1}$ [56]

3. first moment:

$$\begin{aligned}
 \mu(f_{S^2}) &= \mathbb{E}[S^2] \\
 &= \frac{1}{n-1} \sum_{i=1}^n \mathbb{E}[(X_i - \mu(f))^2] - \frac{n}{n-1} \mathbb{E}[(\bar{X} - \mu(f))^2] \\
 &= \frac{1}{n-1} (n\sigma^2(f)) - \frac{n}{n-1} \frac{\sigma^2(f)}{n} = \frac{n}{n-1} \sigma^2(f) - \frac{1}{n-1} \sigma^2(f) = \sigma^2(f)
 \end{aligned} \tag{56}$$

4. The sample variance S^2 is an unbiased estimator of the population variance $\sigma^2(f)$. [56]

5. estimate the standard error of the sample average: $\hat{SE}(\bar{X}) = \frac{S}{\sqrt{n}}$ [56]

6. The second moment of the sample variance is more difficult to determine, but it is possible. [56]

$$\sigma^2(f_{S^2}) = \mathbb{E}[(S^2 - \mu(f_{S^2}))^2] = \mathbb{E}[(S^2 - \sigma^2(f))^2] = \left(\frac{1}{n} \gamma_1(f) + \frac{2}{n-1} \right) \sigma^4(f) \tag{56}$$

7. The standard deviation $\sigma(f_{S^2}) = \sigma^2(f) \sqrt{\frac{(n-1) \gamma_2(f) + 2n}{n(n-1)}}$ is also referred to as the standard error of the sample variance S^2 . [56]

$$8. \text{ standard error: } \hat{SE}(S^2) = S^2 \sqrt{\frac{(n-1) b_2 + 2n}{n(n-1)}} \quad [56]$$

7.6.2. Central Limit Theorem (CLT)

1. If the sample size increases, the standard deviation vanishes ($\frac{\sigma(f)}{\sqrt{n}} \rightarrow 0$ if $n \rightarrow \infty$). This implies that the sample average converges to the population mean $\mu(f)$. [56]
 - 2.
- Theorem 7.2** (Central Limit Theorem (CLT) (Lindeberg-Levy)). If we study the standardized sample average, i.e. $Z_n = \frac{\bar{X} - \mu(f)}{\sigma(f)/\sqrt{n}} = \frac{\sqrt{n}(\bar{X} - \mu(f))}{\sigma(f)}$, the mean would be equal to zero ($\mathbb{E}[Z_n] = 0$) and the variance would be equal to one ($\mathbb{E}[Z_n^2] = 1$). This is true irrespective of the sample size n . [56]
3. Note that the distribution function of Z_n may still depend on n , since the skewness and kurtosis of Z_n are given by $\frac{\gamma_1(f)}{\sqrt{n}}$ and $\frac{\gamma_2(f)}{n}$, respectively, and are different for different n . [56]
 4. If the sample size becomes large, $Z_n = \frac{\sqrt{n}(\bar{X} - \mu(f))}{\sigma(f)} \sim \mathcal{N}(0, 1)$, becomes almost normal. [56]
 5. Note that it did not imply anything about the shape of the population density f , just the existence of $\mu(f)$ and $\sigma^2(f)$ (and of course the assumption that X_1, X_2, \dots, X_n are i.i.d. with density f). [56]
 6. Any statistic of the form $S_n = \frac{1}{n} \sum_{i=1}^n \psi(X_i)$ would also converge to a normal distribution function when the mean $\mu_\psi(f) = \mathbb{E}[\psi(X_k)]$ and variance $\sigma_\psi^2(f) = \mathbb{E}[(\psi(X_k) - \mu_\psi(f))^2]$ are finite. [56]
 7. Using the central limit theorem, $\psi(X_1), \psi(X_2), \dots, \psi(X_n)$ are i.i.d. and have a finite variance; thus, the statistic $\sqrt{n}(S_n - \mu_\psi(f))$ converges to a normal distribution with mean zero and variance $\sigma_\psi^2(f)$. [56]

7.6.2.1. Central Limit Theorem Applied to Variances

1. The central limit theorem can also be applied to the sample variance S^2 , when the fourth central moment $\mathbb{E}[(X_k - \mu(f))^4]$ exists. The sample variance can be rewritten as [56]
- $$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \mu(f))^2 - \frac{n}{n-1} (\bar{X} - \mu(f))^2 \quad [56]$$
2. We may apply the central limit theorem to $\frac{1}{n} \sum_{i=1}^n (X_i - \mu(f))^2$, where $\psi(x) = (x - \mu(f))^2$. [56]
 - (a) **mean:** $\mu_\psi(f) = \mathbb{E}[(X_i - \mu(f))^2] = \sigma^2(f)$ [56]
 - (b) **variance:** $\sigma_\psi^2(f) = \mathbb{E}[(\psi(X_i) - \mu_\psi(f))^2] = \mathbb{E}[(X_i - \mu(f))^4] - \sigma^4(f) = (\gamma_2(f) + 2) \sigma^4(f)$ [56]
 - (c) $\mathbb{V} \left[\frac{1}{n} \sum_{i=1}^n (X_i - \mu(f))^2 \right] = \frac{(\gamma_2(f) + 2) \sigma^4(f)}{n}$ [56]
 - i. $\frac{1}{\sqrt{n}} \sum_{i=1}^n [(X_i - \mu(f))^2 - \sigma^2(f)] \xrightarrow{n \rightarrow \infty} \mathcal{N}(0, [\gamma_2(f) + 2] \sigma^4(f))$ [56]
 - ii. $\frac{1}{n} \sum_{i=1}^n (X_i - \mu(f))^2$ is approximately normally distributed with $\mathcal{N}\left(\sigma^2(f), \frac{[\gamma_2(f) + 2] \sigma^4(f)}{N}\right)$, which implies that $\frac{1}{n-1} \sum_{i=1}^n (X_i - \mu(f))^2$ is approximately normally distributed with $\mathcal{N}\left(\sigma^2(f), \frac{[\gamma_2(f) + 2] \sigma^4(f)}{n}\right)$. [56]

3. The distribution of $\sqrt{n}(\bar{X} - \mu(f))$ converges to the normal distribution $\mathcal{N}(0, \sigma^2(f))$. [56]
- (a) $(\bar{X} - \mu(f))^2 = \frac{[\sqrt{n}(\bar{X} - \mu(f))]^2}{n} \rightarrow 0$ when n converges to ∞ [56]
4. $\sqrt{n}(S^2 - \sigma^2(f))$ converges to a normal distribution $\mathcal{N}(0, [\gamma_2(f) + 2] \sigma^4(f))$, with $\sigma^2(f)$ the population variance and $\gamma_2(f)$ the population excess kurtosis. [56]

7.6.3. Asymptotic Confidence Intervals

1. The sample distribution function F_{T_n} can help quantify how much the statistic is varying around the population characteristic it is trying to approach. [56]
2. Based on the definition of quantiles and the sampling distribution function F_{T_n} , the sample statistic will fall in the interval $(x_p(f_{T_n}), x_{1-p}(f_{T_n}))$ with probability $1 - 2p$. [56]

$$\begin{aligned} P(T_n \in (x_p(f_{T_n}), x_{1-p}(f_{T_n}))) &= P(T_n \leq x_{1-p}(f_{T_n})) - P(T_n \leq x_p(f_{T_n})) \\ 3. \quad &= F_{T_n}(x_{1-p}(f_{T_n})) - F_{T_n}(x_p(f_{T_n})) \\ &= 1 - p - p = 1 - 2p \end{aligned} \quad [56]$$

4. If τ_n is the standard error of the sample statistic T_n , $z_p = x_p(\phi)$ is the quantile of the standard normal distribution, then the sample statistic T_n falls in the interval $(\theta + z_p \tau_n, \theta + z_{1-p} \tau_n) = (\theta - z_{1-p} \tau_n, \theta + z_{1-p} \tau_n)$ with probability approximately equal to $1 - 2p$. [56]

$$\begin{aligned} P(T_n \in (\theta - z_{1-p} \tau_n, \theta + z_{1-p} \tau_n)) \\ 5. \quad &= P((T_n - \theta)/\tau_n \in (-z_{1-p}, z_{1-p})) \\ &\approx \Phi(z_{1-p}) - \Phi(-z_{1-p}) \\ &= 1 - p - p = 1 - 2p \end{aligned} \quad [56]$$

6. We can rewrite the probability $P(T_n \in (\theta - z_{1-p} \tau_n, \theta + z_{1-p} \tau_n))$ into $P(\theta \in (T_n - z_{1-p} \tau_n, T_n + z_{1-p} \tau_n))$, which means that the population characteristic is contained within limits $T_n - z_{1-p} \tau_n$ and $T_n + z_{1-p} \tau_n$ with probability equal to $1 - 2p$. [56]

7. The interval $(T_n - z_{1-p} \tau_n, T_n + z_{1-p} \tau_n)$ is now called an **asymptotic confidence interval** for θ with confidence level $1 - 2p$. [56]
8. In practice we still need to estimate the standard error τ_n to be able to calculate the confidence interval, as τ_n would be a function of the density parameters and is unknown in the calculation of the interval $(T_n - z_{1-p} \tau_n, T_n + z_{1-p} \tau_n)$. [56]
 - (a) It is then common to replace τ_n by its estimator $\hat{\tau}_n$. [56]
 - (b) In some cases, we would also change the normal quantile z_{1-p} by a quantile of the t -distribution if we could formulate a degrees of freedom for the estimator $\hat{\tau}_n$ [56]

7.6.4. Methods of Estimation: Method of Moments Estimation (MME)

1. Assume that the population density f_{θ} depends on a set of parameters $\theta = (\theta_1, \theta_2, \dots, \theta_m)^T$ [56]
2. Estimates of these parameters can then be obtained by computing m or more central moments. [56]

$$3. \quad \mu_r(f_{\theta}) = \mathbb{E}[(X - \mu(f_{\theta}))^r] = \begin{cases} \int_{\mathbb{R}} (x - \mu(f_{\theta}))^r f_{\theta}(x) dx & \text{continuous} \\ \sum_{k=0}^{\infty} (k - \mu(f_{\theta}))^r f_{\theta}(k) & \text{discrete} \end{cases} \quad \text{where} \quad [56]$$

- (a) $\mu(f_{\theta}) = \mathbb{E}[X]$ the mean value [56]
- (b) $f_{\theta}(k)$ represents the probability that X is equal to k [56]

4. The moments can be estimated with the sample moments $M_r = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^r$ with $M_1 = \bar{X}$ and \bar{X} the sample average. [56]
5. If we equate the sample moments M_r to the centralized population moments μ_r , we create a system of equations that can possibly be solved for $\boldsymbol{\theta}$. [56]
6. When executing the method of moments, we are looking for parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_m)^\top$ that satisfy the following equations $\bar{X} = \mu(f_{\boldsymbol{\theta}})$ and $M_r = \mu_r(f_{\boldsymbol{\theta}})$ for $r = 2, 3, \dots, m$. [56]
7. The solution $\tilde{\boldsymbol{\theta}} = (\tilde{\theta}_1, \tilde{\theta}_2, \dots, \tilde{\theta}_m)^\top$ is called the **method of moments estimator**. [56]
8. Note that the second sample moment (M_2) is equal to $\frac{(n-1)}{n} S^2$ and is thus not an unbiased estimator of $\mu_2(f_{\boldsymbol{\theta}})$, since we already know that S^2 is unbiased. [56]
9. The moment estimators on transformed data are different from the moment estimators on the original data [56]
10. The non-uniqueness issue is considered a real disadvantage of the moment estimators. [56]

7.6.5. Methods of Estimation: Maximum Likelihood Estimation (MLE)

1. If X_1, X_2, \dots, X_n are i.i.d. with density $f_{\boldsymbol{\theta}}$, the likelihood function is given by $L(\boldsymbol{\theta}|X_1, X_2, \dots, X_n) = \prod_{i=1}^n f_{\boldsymbol{\theta}}(X_i)$ and the log likelihood function is given by $\ell_{\boldsymbol{\theta}} \equiv (\boldsymbol{\theta}|X_1, X_2, \dots, X_n) = \sum_{i=1}^n \log f_{\boldsymbol{\theta}}(X_i)$ [56]
2. The maximum likelihood estimator $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_n)^\top$ is the set of parameters that maximizes the log likelihood function. It is considered a (vector of) random variable(s), since it is a (set of) function(s) of the random variables X_1, X_2, \dots, X_n . [56]
3. X_1, X_2, \dots, X_n can often be determined by solving the set of equations given by [56]

$$\frac{\partial \ell_{\boldsymbol{\theta}}}{\partial \theta_k} = \sum_{i=1}^n \left(\frac{1}{f_{\boldsymbol{\theta}}(X_i)} \frac{\partial f_{\boldsymbol{\theta}}(X_i)}{\partial \theta_k} \right) = 0 \quad [56]$$

This set of equations is often referred to as the **likelihood equations**. [56]

4. The solution $\hat{\boldsymbol{\theta}}$ does not always result in a closed form expression, which means that we have to resort to numerical approaches if we want to determine the MLE on data. [56]
5. This is often considered a drawback of the MLE. It may estimate the parameters of the PDF or PMF unbiasedly, but they do not always estimate the population mean and variance unbiasedly. This would be different from the moment estimators applied to the original scale of the observations. [56]

7.6.5.1. Standard Error of MLE

1. The standard errors of the maximum likelihood estimators are calculated based on the variance of the large sample distribution (or asymptotics) of the maximum likelihood estimators. [56]
2. Under certain regularity conditions, it can be shown that $\sqrt{n}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})$ converges to a normal distribution $\mathcal{N}(0, I^{-1}(\boldsymbol{\theta}))$, with $\hat{\boldsymbol{\theta}}$ the MLE and $I(\boldsymbol{\theta})$ the so-called Fisher information. [56]

(Informal) Illustrative proof:

1. we will assume that the density $f_{\boldsymbol{\theta}}$ has only one parameter instead of multiple parameters $\theta_1, \theta_2, \dots, \theta_m$. [56]
2. The first derivative of the log likelihood, which is also referred to as the score function $S(\boldsymbol{\theta})$, is given by $S_n(\boldsymbol{\theta}) \equiv \frac{d \ell_{\boldsymbol{\theta}}}{d \boldsymbol{\theta}} = \sum_{i=1}^n \frac{d}{d \boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(X_i) = \sum_{i=1}^n \frac{f'_{\boldsymbol{\theta}}(X_i)}{f_{\boldsymbol{\theta}}(X_i)}$ [56]

3. The expectation of the score function is zero (under certain regularity conditions), since [56]

$$\mathbb{E} \left[\frac{f'_\theta(X_i)}{f_\theta(X_i)} \right] = \int_{\mathbb{R}} f'_\theta(x) dx = \frac{d}{d\theta} \int_{\mathbb{R}} f_\theta(x) dx = 0 \quad [56]$$

This implies that $\mathbb{E}[S_n(\theta)] = 0$. [56]

4. The variance of the score function is now given by $\mathbb{E}[S_n^2(\theta)] = n \int_{\mathbb{R}} \frac{(f'_\theta(x))^2}{f_\theta(x)} dx$. [56]

$$5. \textbf{Fisher information: } I(\theta) = \mathbb{E} \left[\left(\frac{d \log f_\theta(X)}{d\theta} \right)^2 \right] = \int_{\mathbb{R}} \frac{(f'_\theta(x))^2}{f_\theta(x)} dx \quad [56]$$

6. The score function is a sum of independent random variables, which means that if we standardize the score function appropriately, the central limit theorem tells us that the distribution function of the score function will converge to a normal distribution function. $S_n(\theta)/\sqrt{nI(\theta)}$ converges to a standard normal random variable $Z \sim \mathcal{N}(0, 1)$. [56]

7. The Fisher information is equal to the minus expectation of the derivative of the score function with respect to the parameter θ , i.e. $nI(\theta) = -\mathbb{E}[S'_n(\theta)]$ with $S'_n(\theta) = \frac{dS_n(\theta)}{d\theta}$. [56]

8. $S'_n(\theta)$ is also a sum of independent random variables; thus properly standardized it will converge to a standard normal random variable. This also implies that $\frac{S'_n(\theta)}{n}$ will converge to the Fisher information $I(\theta)$. [56]

9. using a first-order Taylor expansion for the score function $S_n(\hat{\theta})$ that is evaluated in the ML estimator $\hat{\theta}$, we obtain that $S_n(\hat{\theta}) \approx S_n(\theta) + (\hat{\theta} - \theta) S'_n(\theta)$. But the score function in the MLE estimator is zero, i.e. $S_n(\hat{\theta}) = 0$, since the MLE estimator $\hat{\theta}$ maximizes the likelihood. This means that [56]

$$\sqrt{n}(\hat{\theta} - \theta) \approx -\frac{\sqrt{n}S_n(\theta)}{S'_n(\theta)} = -\frac{1}{\sqrt{I(\theta)}} \frac{S_n(\theta)}{\sqrt{nI(\theta)}} \frac{nI(\theta)}{S'_n(\theta)} \quad [56]$$

10. Since $\frac{nI(\theta)}{S'_n(\theta)}$ converges to 1 and $\frac{S_n(\theta)}{\sqrt{nI(\theta)}}$ converges to $Z \sim \mathcal{N}(0, 1)$, we obtain that $\sqrt{n}(\hat{\theta} - \theta)$ converges to $\mathcal{N}\left(0, \frac{1}{I(\theta)}\right)$. [56]

11. we see that the asymptotic variance of the maximum likelihood estimator is $I^{-1}(\theta)$. [56]

12. This implies that the approximate standard error (or asymptotic standard error) of the maximum likelihood estimator is now $SE(\hat{\theta}) = \frac{1}{\sqrt{nI(\theta)}}$. Since we have an estimator $\hat{\theta}$ of the parameter θ , the standard error is estimated with $\hat{SE}(\hat{\theta}) = \frac{1}{\sqrt{nI(\hat{\theta})}}$. This asymptotic standard error ($SE(\hat{\theta})$) can be used in the calculation of confidence intervals on θ . [56]

13. When $T_n = \hat{\theta}$ is the MLE for θ having an estimated standard error $\hat{s}_n = \frac{1}{\sqrt{nI(\hat{\theta})}}$, the $1 - 2p$ asymptotic confidence interval is $\left(\frac{\hat{\theta} - z_{1-p}}{\sqrt{nI(\hat{\theta})}}, \frac{\hat{\theta} + z_{1-p}}{\sqrt{nI(\hat{\theta})}} \right)$ [56]

7.7. Expected Values ($E(X)$)

1. **Covariance:** Covariance measures how two variables change together. If one variable increases when the other increases, the covariance is positive; if one decreases when the other increases, it's negative.

[7]

- (a) Units depend on the original variables. [7]
 - (b) Hard to interpret the strength of the relationship due to unit dependence. [7]
 - (c) Sensitive to scale. [7]
2. **Correlation:** Correlation standardizes covariance and measures both direction and strength of a linear relationship between two variables. [7]
- (a) Always between -1 and $+1$. [7]
 - i. $+1$: perfect positive linear relationship [7]
 - ii. 0 : no linear relationship [7]
 - iii. -1 : perfect negative linear relationship [7]
 - (b) Unitless. [7]
 - (c) Easier to interpret than covariance. [7]
3. **Causation:** Causation means that changes in one variable directly cause changes in another. [7]
- (a) Implies a cause-and-effect relationship. [7]
 - (b) Cannot be determined just by looking at data or statistical relationships. [7]
 - (c) Requires controlled experiments or strong observational evidence (e.g., through randomized trials or causal inference techniques). [7]

7.7.1. For PDF (Continuous)

1. If we consider a (not necessarily continuous or differentiable) function $\psi : \mathbb{R} \rightarrow \mathbb{R}$, then the expected value of the random variable $\psi(X)$ is defined by
$$\mathbb{E}[\psi(X)] = \int_{\mathbb{R}} \psi(x) f(x) dx$$
 [56]

7.7.2. For PMF (Discrete)

1. The expectation of a discrete random variable $\psi(X)$ is given by [56]

$$\mathbb{E}[\psi(X)] = \sum_{k=0}^{\infty} \psi(k) p_k = \sum_{k=0}^{\infty} \psi(k) P(X=k)$$
 [56]
2. If we would collect many realizations of the discrete random variable X , say N realizations, we expect to see value k with frequency $N \cdot p_k$. [56]

7.7.3. Common for PDF & PMF

1. **First moment:** mean μ is obtained by
$$\psi(x) = x$$
 [56]
2. **Second central moment:** The population variance σ^2 is obtained by
$$\psi(x) = (x - \mu)^2$$
 [56]
3. **pth moment** of random variable X is obtained by
$$\psi(x) = x^p$$
 [56]
4. **pth central moment** of random variable X is obtained by choosing
$$\psi(x) = (x - \mu)^p$$
 [56]
5. **skewness:** $\gamma_1 = \frac{\mu_3}{\sigma^3}$, where $\mu_3 = \mathbb{E}[(x - \mu)^3]$ [56]
6. **kurtosis:** $\gamma_2 = \frac{\mu_4}{\sigma^4} - 3$, where $\mu_4 = \mathbb{E}[(x - \mu)^4]$ [56]

Note:

1. the moments of a discrete random variable X may **not** always exist: this depends on the choice of probabilities p_k . [56]

7.8. Calculation Rules for Random Variables

1. Let we have 2 random variables X, Y , either discrete or continuous, and a constant c .
2. F_X and F_Y the CDFs for X and Y , respectively. $P(X \leq x, Y \leq y)$ is the joint CDF of X and Y , denoted by $F_{XY}(x,y)$
3. Let events $A = \{X \leq x\}$ and $B = \{Y \leq y\}$
4. the covariance is affected by transformations

7.8.1. Univariate/ Joint PMF/ PDF/ CDF

1. The expected value of a function $g : \mathbb{R} \rightarrow \mathbb{R}$ of a univariate continuous random variable $X \sim P(x)$ is given by $\mathbb{E}_X[g(x)] = \int_{\mathcal{X}} g(x) P(x) dx$ where \mathcal{X} is the set of possible outcomes (the target space) of the random variable X . [49]

The expected value of a function of a random variable is sometimes referred to as the **law of the unconscious statistician**. [49]

It defines the meaning of the notation \mathbb{E}_X as the operator indicating that we should take the integral with respect to the probability density (for continuous distributions) or the sum over all states (for discrete distributions). [49]

When the random variable associated with the expectation or covariance is clear by its arguments, the subscript is often suppressed (for example, $\mathbb{E}_X[x]$ is often written as $\mathbb{E}[x]$). [49]

2. For multivariate random variables, we define the expected value element wise [49]

$$\mathbb{E}_X[g(\mathbf{x})] = \begin{bmatrix} \mathbb{E}_{X_1}[g(x_1)] \\ \vdots \\ \mathbb{E}_{X_D}[g(x_D)] \end{bmatrix} \in \mathbb{R}^D \quad [49]$$

where the subscript \mathbb{E}_{X_d} indicates that we are taking the expected value with respect to the d -th element of the vector \mathbf{x} . [49]

3. The mean of a random variable X with states $\mathbf{x} \in \mathbb{R}^D$ is an average and is defined as [49]

$$\mathbb{E}_X[\mathbf{x}] = \begin{bmatrix} \mathbb{E}_{X_1}[x_1] \\ \vdots \\ \mathbb{E}_{X_D}[x_D] \end{bmatrix} \in \mathbb{R}^D \quad [49]$$

where [49]

$$\mathbb{E}_{X_d}[x_d] := \begin{cases} \int_{\mathcal{X}} x_d P(x_d) dx_d & \text{if } X \text{ is a continuous random variable} \\ \sum_{x_i \in \mathcal{X}} x_i P(x_d = x_i) & \text{if } X \text{ is a discrete random variable} \end{cases} \quad [49]$$

for $d = 1, \dots, D$, where the subscript d indicates the corresponding dimension of \mathbf{x} . The integral and sum are over the states \mathcal{X} of the target space of the random variable X . [49]

4. The expected value is a linear operator. For example, given a real-valued function $f(x) = ag(x) + bh(x)$ where $a, b \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^D$, we obtain $\mathbb{E}_X[f(x)] = a\mathbb{E}_X[g(x)] + b\mathbb{E}_X[h(x)]$ [49]
5. The covariance between two univariate random variables $X, Y \in \mathbb{R}$ is given by the expected product of their deviations from their respective means, i.e., [49]

$$\text{Cov}_{X,Y}[x,y] := \mathbb{E}_{X,Y}[(x - \mathbb{E}_X[x])(y - \mathbb{E}_Y[y])] = \mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y] \quad [49]$$

The covariance of multivariate random variables $\text{Cov}[x,y]$ is sometimes referred to as **cross-covariance**, with covariance referring to $\text{Cov}[x,x]$. [49]

6. The covariance of a variable with itself $\text{Cov}[x,x]$ is called the variance and is denoted by $\mathbb{V}_X[x]$. The square root of the variance is called the **standard deviation** and is often denoted by $\sigma(x)$. [49]

7. If we consider two multivariate random variables X and Y with states $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^E$ respectively, the covariance between X and Y is defined as [49]

$$\text{Cov}[\mathbf{x}, \mathbf{y}] = \mathbb{E}[\mathbf{x}\mathbf{y}^\top] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}]^\top = \text{Cov}[\mathbf{y}, \mathbf{x}]^\top \in \mathbb{R}^{D \times E} \quad [49]$$

8. The variance of a random variable X with states $\mathbf{x} \in \mathbb{R}^D$ and a mean vector $\boldsymbol{\mu} \in \mathbb{R}^D$ is defined as [49]

$$\begin{aligned} \mathbb{V}_X[\mathbf{x}] &= \text{Cov}_X[\mathbf{x}, \mathbf{x}] = \mathbb{E}_X[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top] \\ &= \mathbb{E}_X[\mathbf{x}\mathbf{x}^\top] - \mathbb{E}_X[\mathbf{x}]\mathbb{E}_X[\mathbf{x}]^\top \\ &= \begin{bmatrix} \text{Cov}[x_1, x_1] & \text{Cov}[x_1, x_2] & \cdots & \text{Cov}[x_1, x_D] \\ \text{Cov}[x_2, x_1] & \text{Cov}[x_2, x_2] & \cdots & \text{Cov}[x_2, x_D] \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}[x_D, x_1] & \text{Cov}[x_D, x_2] & \cdots & \text{Cov}[x_D, x_D] \end{bmatrix} \end{aligned} \quad [49]$$

(a) The $D \times D$ matrix is called the **covariance matrix** of the multivariate random variable X . [49]

(b) The covariance matrix is symmetric and positive semidefinite. [49]

(c) On its diagonal, the covariance matrix contains the variances of the **marginals** [49]

$$P(x_i) = \int P(x_1, \dots, x_D) dx_{\setminus i} \quad [49]$$

where “\i” denotes “all variables but i”. [49]

(d) The off-diagonal entries are the **cross-covariance** terms $\text{Cov}[x_i, x_j]$ for $i, j = 1, \dots, D, i \neq j$. [49]

9. Consider a random variable X with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ and a (deterministic) affine transformation $\mathbf{y} = \mathbf{Ax} + \mathbf{b}$ of \mathbf{x} . [49]

$$(a) \mathbb{E}_Y[\mathbf{y}] = \mathbb{E}_X[\mathbf{Ax} + \mathbf{b}] = \mathbf{A}\mathbb{E}_X[\mathbf{x}] + \mathbf{b} = \mathbf{A}\boldsymbol{\mu} + \mathbf{b} \quad [49]$$

$$(b) \mathbb{V}_Y[\mathbf{y}] = \mathbb{V}_X[\mathbf{Ax} + \mathbf{b}] = \mathbb{V}_X[\mathbf{Ax}] = \mathbf{A}\mathbb{V}_X[\mathbf{x}]\mathbf{A}^\top = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top \quad [49]$$

$$(c) \begin{aligned} \text{Cov}[\mathbf{x}, \mathbf{y}] &= \mathbb{E}[\mathbf{x}(\mathbf{Ax} + \mathbf{b})^\top] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{Ax} + \mathbf{b}]^\top = \mathbb{E}[\mathbf{x}]\mathbf{b}^\top + \mathbb{E}[\mathbf{x}\mathbf{x}^\top]\mathbf{A}^\top - \boldsymbol{\mu}\mathbf{b}^\top - \boldsymbol{\mu}\boldsymbol{\mu}^\top\mathbf{A}^\top \\ &= \boldsymbol{\mu}\mathbf{b}^\top - \boldsymbol{\mu}\mathbf{b}^\top + \mathbb{E}[\mathbf{x}\mathbf{x}^\top] - \boldsymbol{\mu}\boldsymbol{\mu}^\top\mathbf{A}^\top = \boldsymbol{\Sigma}\mathbf{A}^\top \end{aligned} \quad [49]$$

where $\boldsymbol{\Sigma} = \mathbb{E}[\mathbf{x}\mathbf{x}^\top] - \boldsymbol{\mu}\boldsymbol{\mu}^\top$ is the covariance of X . [49]

$$10. \text{Cov}[X, Y] = \sigma_{XY} = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \quad [56]$$

$$11. \mathbb{E}[\mathbb{E}[X]] = \mathbb{E}[X] \quad [7] \quad 19. \mathbb{V}[cX] = c^2 \mathbb{V}[X] \quad [56]$$

$$12. \mathbb{E}[c] = c \quad [56] \quad 20. \mathbb{V}_X[x] := \mathbb{E}_X[(x - \boldsymbol{\mu})^2] \quad [49]$$

$$13. \mathbb{E}[cX] = c\mathbb{E}[X] \quad [56] \quad 21. \mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \quad [49, 56]$$

$$14. \mathbb{E}[X] = \mathbb{E}[Y], \text{ when } X = Y \quad [56] \quad 22. \mathbb{V}[X \pm Y] = \mathbb{V}[X] \pm 2\text{Cov}[X, Y] + \mathbb{V}[Y] \quad [49, 56]$$

$$15. \mathbb{E}[X \pm Y] = \mathbb{E}[X] \pm \mathbb{E}[Y] \quad [49, 56] \quad 23. \text{Cov}[X, X] = \mathbb{V}[X] = \sigma_X^2 \quad [56]$$

$$16. \mathbb{E}[g(X, Y)] = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} g(x, y) f_{XY}(x, y) \quad [56] \quad 24. \text{Cov}[X, Y] = \text{Cov}[Y, X] \quad [56]$$

$$17. \mathbb{V}[X] \geq 0 \quad [56] \quad 25. \text{Cov}[aX, Y] = a\text{Cov}[X, Y] \quad [56]$$

$$18. \mathbb{V}[X + c] = \mathbb{V}[X] \quad [56] \quad 26. \text{Cov}[X + c, Y] = \text{Cov}[X, Y] \quad [56]$$

$$27. \text{Cov}[X + Y, Z] = \text{Cov}[X, Z] + \text{Cov}[Y, Z] \quad [56]$$

7.8.1.1. if X and Y are independent

$$28. P(X \leq x, Y \leq y) = P(A \cap B) = P(A)P(B) = P(X \leq x)P(Y \leq y) = F_X(x)F_Y(y) \quad [56]$$

$$29. \mathbb{V}[XY] = \mathbb{V}[X]\mathbb{V}[Y] + \mathbb{V}[X](\mathbb{E}[Y])^2 + \mathbb{V}[Y](\mathbb{E}[X])^2 \quad [56]$$

30. Two random variables can have a covariance of 0, and still be dependent. A simple example is to take X as a standard normal random variable and $Y = X^2$. It is clear that X and Y are dependent, since Y is a function of X , and that $\text{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[X^3] = 0$. [56]

$$31. \phi(X) \text{ and } \psi(Y) \text{ are also independent} \quad [56] \quad 33. \mathbb{V}[X \pm Y] = \mathbb{V}[X] + \mathbb{V}[Y] \quad [49, 56]$$

$$32. \mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \quad [56] \quad 34. \text{Cov}[X, Y] = 0 \quad [49, 56]$$

7.8.2. Conditional PMF, PDF & CDF

$$35. f_{X|Y}(x|y) = \begin{cases} P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} = \frac{f_{XY}(x, y)}{f_Y(y)} & \text{if } f_Y(y) > 0 \\ 0 & \text{otherwise} \end{cases} \quad [56]$$

$$36. f_{Y|X}(y|x) = \begin{cases} P(Y = y|X = x) = \frac{P(X = x, Y = y)}{P(X = x)} = \frac{f_{XY}(x, y)}{f_X(x)} & \text{if } f_X(x) > 0 \\ 0 & \text{otherwise} \end{cases} \quad [56]$$

$$37. \mathbb{V}[Y|X = x] = \mathbb{E}[(Y - \mu_Y(x))^2|X = x] = \sum_{y=0}^{\infty} (y - \mu_Y(x))^2 f_{Y|X}(y|x) \quad [56]$$

$$\begin{aligned} 38. \mathbb{E}[\mu_Y(X)] &= \sum_{x=0}^{\infty} \mu_Y(x) f_X(x) = \sum_{x=0}^{\infty} \mathbb{E}(Y|X = x) f_X(x) = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} y f_{Y|X}(y|x) f_X(x) \\ &= \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} y f_{XY}(x, y) = \sum_{y=0}^{\infty} y f_Y(y) = \mu_Y \end{aligned} \quad [56]$$

$$\begin{aligned} 39. \mathbb{E}[\sigma_Y^2(X)] &= \sum_{x=0}^{\infty} \sigma_Y^2(x) f_X(x) = \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} (y - \mu_Y(x))^2 f_{Y|X}(y|x) f_X(x) \\ &= \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} [(y - \mu_Y)^2 + 2(y - \mu_Y)(\mu_Y - \mu_Y(x)) + (\mu_Y - \mu_Y(x))^2] f_{XY}(x, y) \\ &= \sum_{y=0}^{\infty} (y - \mu_Y)^2 f_Y(y) - \sum_{x=0}^{\infty} (\mu_Y(x) - \mu_Y)^2 f_X(x) = \sigma_Y^2 - \mathbb{V}[\mu_Y(X)] \end{aligned} \quad [56]$$

$$40. \mathbb{E}(\phi(Y)|X = x) = \sum_{y=0}^{\infty} \phi(y) f_{Y|X}(y|x) \quad [56] \quad 43. \mathbb{E}(\phi(Y)|X = x) = \int_{-\infty}^{\infty} \phi(y) f_{Y|X}(y|x) dy \quad [56]$$

$$41. f_{X|Y}(x|y) = \begin{cases} \frac{f_{XY}(x, y)}{f_Y(y)} & \text{if } f_Y(y) > 0 \\ 0 & \text{otherwise} \end{cases} \quad [56] \quad 44. \mu_Y(x) = \mathbb{E}(Y|X = x) \quad [56]$$

$$42. f_{Y|X}(y|x) = \begin{cases} \frac{f_{XY}(x, y)}{f_X(x)} & \text{if } f_X(x) > 0 \\ 0 & \text{otherwise} \end{cases} \quad [56] \quad 45. \mu_Y = \mathbb{E}[\mu_Y(X)] \quad [56]$$

$$46. \mathbb{V}[Y] = \mathbb{E}[\mathbb{V}[Y|X = X]] + \mathbb{V}[\mathbb{E}[Y|X = X]] \quad [56]$$

$$47. \mathbb{V}_X[x] = \mathbb{E}_Y[\mathbb{V}_X[x|y]] + \mathbb{V}_Y[\mathbb{E}_X[x|y]] \quad [49]$$

7.8.2.1. If X & Y are independent

$$48. F_{XY}(x, y) = F_X(x)F_Y(y) \quad [56] \quad 50. f_{X|Y}(x|y) = f_X(x) \quad [49, 56]$$

$$49. f_{XY}(x, y) = f_X(x)f_Y(y) \quad [56] \quad 51. f_{Y|X}(y|x) = f_Y(y) \quad [49, 56]$$

7.9. Constructing Bivariate Distributions

7.9.1. Sums of Random Variables

- Let U , V , and W be three independent random variables and define $X = W + U$ and $Y = W + V$. Since both X and Y share the same random variable W , the random variables X and Y must be **dependent**. If we only observe the random variables X and Y in practice, the random variable W is sometimes referred to as **latent variable**. [56]

- The random variable W represents a “true” value of something that is measured, and the other random variables U and V represent measurement errors. Thus in the bivariate case, the same unit is just measured twice. [56]

- $\sigma_X^2 = \sigma_W^2 + \sigma_U^2$ and $\sigma_Y^2 = \sigma_W^2 + \sigma_V^2$ [56]

$$\text{Cov}[X, Y] = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] = \mathbb{E}[(W - \mu_W + U - \mu_U)(W - \mu_W + V - \mu_V)]$$

- $= \mathbb{V}[W] + \mathbb{E}[(W - \mu_W)(V - \mu_V)] + \mathbb{E}[(U - \mu_U)(W - \mu_W)] + \mathbb{E}[(U - \mu_U)(V - \mu_V)]$ [56]
 $= \sigma_W^2$

- Pearson’s correlation coefficient: $\rho_P = \frac{\sigma_W^2}{\sqrt{(\sigma_W^2 + \sigma_U^2)(\sigma_W^2 + \sigma_V^2)}}$ [56]

- (a) This correlation coefficient is often referred to as the **intraclass correlation coefficient** (ICC). [56]
- Together with the sample variances S_X^2 and S_Y^2 , which are estimators for $\sigma_X^2 = \sigma_W^2 + \sigma_U^2$ and $\sigma_Y^2 = \sigma_W^2 + \sigma_V^2$, respectively, we may obtain an estimator for the variance σ_W^2 of W , using $r_P S_X S_Y$. [56]

7.9.2. Farlie-Gumbel-Morgenstern (FGM) Family of Distributions

- When X and Y are two random variables with marginal CDFs F_X and F_Y , respectively, we can create a bivariate CDF in the following way: [56]

$$F_{XY}(x, y) = F_X(x) F_Y(y) (1 + \alpha(1 - F_X(x))(1 - F_Y(y)))$$
 [56]

- PDF: $f_{XY}(x, y) = f_X(x) f_Y(y) (1 + \alpha(1 - 2F_X(x))(1 - 2F_Y(y)))$ [56]

- The parameter α should be within $[-1, 1]$ and when it is equal to zero, the random variables X and Y are **independent**. The parameter α may be seen as the dependency parameter and the larger the absolute value $|\alpha|$ the stronger the dependency. [56]

- This class of distribution functions is referred to as the one-parameter Farlie–Gumbel–Morgenstern (FGM) distribution function. [56]

- It can be shown that the marginal distribution functions are given by F_X and F_Y , respectively. [56]

- The advantage of this class is that it may be used with many different choices for F_X and F_Y . [56]

$$\text{Cov}[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

- $= \left(\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x y f_X(x) f_Y(y) (1 + \alpha(1 - 2F_X(x))(1 - 2F_Y(y))) dx dy \right) - \mu_X \mu_Y$ [56]
 $= \alpha \mathbb{E}[X(1 - 2F_X(X))] \mathbb{E}[Y(1 - 2F_Y(Y))]$

- Pearson’s correlation coefficient:** $\rho_P = \frac{\alpha \mathbb{E}[X(1 - 2F_X(X))] \mathbb{E}[Y(1 - 2F_Y(Y))]}{\sigma_X \sigma_Y}$ [56]

- (a) Pearson’s correlation coefficient for the FGM family is bounded by $-\frac{1}{3} \leq \rho_P \leq \frac{1}{3}$ [56]

- (b) This implies that the FGM family of CDFs may be useful for applications where the random variables are **weakly correlated**. [56]

9. **Kendall's tau**: $\tau_K = \frac{2\alpha}{9}$ [56]

10. **Spearman's rho correlation**: $\rho_S = \frac{\alpha}{3}$ [56]

11. **Kendall's tau estimator** (r_K): α can be estimated by $\hat{\alpha} = \frac{9r_K}{2}$ [56]

12. **Spearman's rho estimator** (r_S): α can be estimated by $\hat{\alpha} = 3r_K$ [56]

It should be noted that when these estimates are outside the range of the dependency parameters (i.e., $\hat{\alpha} \notin [-1, 1]$), the data may not support these families of distributions. [56]

7.9.3. Fréchet Family of Distributions

1. Maurice Fréchet demonstrated that any bivariate CDF $F_{XY}(x, y)$ can be bounded from below and from above using the marginal CDFs $F_X(x)$ and $F_Y(y)$. [56]

2. $\max \{F_X(x) + F_Y(y) - 1, 0\} \leq F_{XY}(x, y) \leq \min \{F_X(x), F_Y(y)\}$ [56]

3. The boundary functions $F_{BL}(x, y) = \max \{F_X(x) + F_Y(y) - 1, 0\}$ and $F_{BU}(x, y) = \min \{F_X(x), F_Y(y)\}$ are both CDFs themselves. [56]

4. The two boundary CDFs can then be taken to form a one-parameter class of CDFs of the form $F_{LU}(x, y) = \lambda F_{BL}(x, y) + (1 - \lambda) F_{BU}(x, y)$, $\lambda \in [0, 1]$. The parameter λ indicates the strength of the dependence, even though this class does not contain the independent setting. [56]

5. There is **no** λ value that leads to $F_{LU}(x, y) = F_X(x) F_Y(y)$. [56]

6. **Kendall's tau**: $\tau_K = \frac{2\lambda - 1}{3}$ [56]

7. **Spearman's rho correlation**: $\rho_S = \frac{2\lambda - 1}{3}$ [56]

8. **Kendall's tau estimator** (r_K): λ can be estimated by $\hat{\lambda} = \frac{3r_K + 1}{2}$ [56]

9. **Spearman's rho estimator** (r_S): λ can be estimated by $\hat{\lambda} = \frac{3r_S + 1}{2}$ [56]

It should be noted that when these estimates are outside the range of the dependency parameters (i.e., $\hat{\lambda} \notin [0, 1]$), the data may not support these families of distributions. [56]

7.9.4. Mixtures of Probability Distributions

7.9.4.1. Conditionally independent

1. Let Z be a random variable and conditionally on this random variable we assume that X and Y are independent distributed. Thus the joint PDF of X and Y given Z , which is denoted by $f_{XY|Z}(x, y|z)$, is now given by [56]

$$f_{XY|Z}(x, y|z) = f_{X|Z}(x|z) f_{Y|Z}(y|z) = \frac{f_{XZ}(x, z) f_{YZ}(y, z)}{f_Z^2(z)} \quad [56]$$

2. Then the joint PDF for X and Y is obtained by: $f_{XY}(x, y) = \int_{-\infty}^{\infty} f_{X|Z}(x|z) f_{Y|Z}(y|z) f_Z(z) dz$ [56]

3. $\mu_{XY}(z) = \mathbb{E}[XY|Z = z]$ [56]

4. Because X and Y are independent conditionally on $Z = z$, we also obtain that $\mathbb{E}[XY|Z = z] = \mathbb{E}[X|Z = z]\mathbb{E}[Y|Z = z] = \mu_X(z) \mu_Y(z)$. [56]

$$\begin{aligned} 5. \quad \mathbb{E}[\mu_{XY}(Z)] &= \int_{-\infty}^{\infty} \mu_{XY}(z) f_Z(z) dz = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x y f_{XYZ}(x,y|z) f_Z(z) dx dy dz \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x y f_{XYZ}(x,y,z) dx dy dz = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x y f_{XY}(x,y) dx dy = \mathbb{E}[XY] \end{aligned} \quad [56]$$

$$\begin{aligned} 6. \quad \text{Cov}[X,Y] &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[\mu_{XY}(Z)] - \mu_X\mu_Y = \mathbb{E}[\mu_X(Z)\mu_Y(Z)] - \mu_X\mu_Y \\ &= \mathbb{E}[(\mu_X(Z) - \mu_X)(\mu_Y(Z) - \mu_Y)] = \text{Cov}[\mu_X(Z), \mu_Y(Z)] \end{aligned} \quad [56]$$

7. If we assume that X and Y are identically distributed (conditionally on Z), we would have that $\mu_X(z) = \mu_Y(z) = \mu(z)$, which would lead to $\text{Cov}[X,Y] = \mathbb{V}[\mu(Z)]$ [56]

8. marginal CDFs for X and Y : [56]

$$(a) \quad f_X(x) = \int_{-\infty}^{\infty} f_{X|Z}(x|z) f_Z(z) dz \quad [56] \quad (b) \quad f_Y(y) = \int_{-\infty}^{\infty} f_{Y|Z}(y|z) f_Z(z) dz \quad [56]$$

9. variances of X and Y : [56]

$$(a) \quad \mathbb{V}[X] = \mathbb{E}[\mathbb{V}[X|Z=Z]] + \mathbb{V}(\mu_X(Z)) \quad [56] \quad (b) \quad \mathbb{V}[Y] = \mathbb{E}[\mathbb{V}[Y|Z=Z]] + \mathbb{V}(\mu_Y(Z)) \quad [56]$$

10. Pearson's correlation coefficient: [56]

$$\rho_P = \frac{\text{Cov}[\mu_X(Z), \mu_Y(Z)]}{\sqrt{(\mathbb{E}[\mathbb{V}[X|Z=Z]] + \mathbb{V}(\mu_X(Z)))(\mathbb{E}[\mathbb{V}[Y|Z=Z]] + \mathbb{V}(\mu_Y(Z)))}} \quad [56]$$

7.10. Measures of Association

7.10.1. Pearson's Correlation Coefficient ($CORR(X, Y) = \rho_P$)

1. Pearson's correlation coefficient is a way of quantifying how two variables “co-relate”. If Pearson's correlation is **positive**, the two variables X and Y move in the **same direction**. If X is increasing then Y should be increasing as well. When Pearson's correlation coefficient is **zero** the random variables are called **uncorrelated**. [56]

2. standardized random variables: $Z_X = \frac{X - \mu_X}{\sigma_X}$ and $Z_Y = \frac{Y - \mu_Y}{\sigma_Y}$ [56]

$$3. \quad \rho_P = CORR(X, Y) = \text{Cov}[Z_X, Z_Y] = \mathbb{E}\left[\left(\frac{X - \mu_X}{\sigma_X}\right)\left(\frac{Y - \mu_Y}{\sigma_Y}\right)\right] = \frac{\text{Cov}[X, Y]}{\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}} \quad [56]$$

4. Properties:

$$(a) \quad CORR(X, Y) \in [-1, 1] \quad [56]$$

$$(b) \quad \text{If } CORR(X, Y) = 1, \text{ then } Y = aX + b, \text{ where } a > 0 \quad [56]$$

$$(c) \quad \text{If } CORR(X, Y) = -1, \text{ then } Y = aX + b, \text{ where } a < 0 \quad [56]$$

$$(d) \quad CORR(aX + b, cY + d) = CORR(X, Y) \text{ for } a, c > 0 \text{ or } a, c < 0 \quad [56]$$

5. $1 - \rho_P$ is not a distance measure, but $\sqrt{1 - \rho_P}$ is a distance measure. [56]

6. The correlation matrix is the covariance matrix of standardized random variables, $\frac{x}{\sigma(x)}$. In other words, each random variable is divided by its standard deviation (the square root of the variance) in the correlation matrix. [49]

$0.90 < \rho_P \leq 1.00$	Very strong correlation
$0.70 < \rho_P \leq 0.90$	Strong correlation
$0.50 < \rho_P \leq 0.70$	Moderate correlation
$0.30 < \rho_P \leq 0.50$	Low correlation
$0 \leq \rho_P \leq 0.30$	Negligible correlation

Rule of thumb

7.10.1.1. Pearson's Correlation Coefficient estimator (r_P)

1. assume that we have observed the pairs of observations $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ on n units, being i.i.d. $(X_i, Y_i) \sim F_{XY}$ [56]
2. Since the pairs are identically distributed, the covariance $\text{Cov}[X_i, Y_i]$ is equal to $\text{Cov}[X, Y]$ for all units i . [56]
3. **independence between pairs of variables:** $\mathbb{E}[(X_i - \mu_X)(Y_j - \mu_Y)] = \mathbb{E}[X_i - \mu_X]\mathbb{E}[Y_j - \mu_Y] = 0$, when $i \neq j$ [56]
4. **Sample Covariance:** $S_{XY} = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})$ [56]

$$\begin{aligned}\mathbb{E}[S_{XY}] &= \frac{1}{n-1} \sum_{i=1}^n \mathbb{E}[(X_i - \bar{X})(Y_i - \bar{Y})] = \frac{1}{n-1} \sum_{i=1}^n \mathbb{E}[(X_i - \mu_X + \mu_X - \bar{X})(Y_i - \mu_Y + \mu_Y - \bar{Y})] \\ 5. \quad &= \frac{1}{n-1} \sum_{i=1}^n \mathbb{E}[(X_i - \mu_X)(Y_i - \mu_Y) - (X_i - \mu_X)(\bar{Y} - \mu_Y) - (\bar{X} - \mu_X)(Y_i - \mu_Y) + (\bar{X} - \mu_X)(\bar{Y} - \mu_Y)] \quad [56] \\ &= \frac{n}{n-1} \text{Cov}[X, Y] - 2 \frac{1}{n(n-1)} \sum_{i=1}^n \mathbb{E}[(X_i - \mu_X)(Y_i - \mu_Y)] + \frac{1}{n(n-1)} \sum_{i=1}^n \mathbb{E}[(X_i - \mu_X)(Y_i - \mu_Y)] \\ &= \text{Cov}[X, Y]\end{aligned}$$

6. The sample version of Pearson's correlation, r_P , also called the **product-moment correlation coefficient** or **sample correlation coefficient**, can now be computed by substituting the sample covariance and the sample variances S_X^2 and S_Y^2 in the definition of ρ_P . [56]

$$r_P = \frac{S_{XY}}{S_X S_Y} \quad [56]$$

7. The product-moment correlation r_P is typically not unbiased, which means that $\mathbb{E}[r_P] \neq \rho_P$. The reason is that both the numerator and the denominator are random variables and the expectation of a ratio is not equal to the ratio of the expectations. [56]
8. The expectation of the numerator in r_P is unbiased for the numerator in the definition of Pearson's correlation ρ_P . (as $\mathbb{E}[S_{XY}] = \text{Cov}[X, Y]$) [56]

$$\begin{aligned}r_P &= \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)S_X S_Y} = \frac{\sum_{i=1}^n X_i Y_i - n \bar{X} \bar{Y}}{(n-1)S_X S_Y} \\ 9. \quad &= \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}} = \frac{n \sum_{i=1}^n X_i Y_i - \sum_{i=1}^n X_i \sum_{i=1}^n Y_i}{\sqrt{n \sum_{i=1}^n X_i^2 - \left(\sum_{i=1}^n X_i\right)^2} \sqrt{n \sum_{i=1}^n Y_i^2 - \left(\sum_{i=1}^n Y_i\right)^2}} \quad [56]\end{aligned}$$

10. The asymptotic distribution of r_P is normal when the pairs (X_i, Y_i) are i.i.d. F_{XY} with finite fourth central moments. More specifically, $\sqrt{n}(r_P - \rho_P) \sim \mathcal{N}\left(0, \frac{n(1-\rho_P^2)^2}{n-3}\right)$ [56]

11. If the observed data is **not normal** and the **sample size is relatively large**, the asymptotic confidence interval may be applied directly on the product-moment estimator, i.e., [56]

$$\left[r_P - \frac{z_{1-\alpha/2}(1-r_P^2)}{\sqrt{n-3}}, r_P + \frac{z_{1-\alpha/2}(1-r_P^2)}{\sqrt{n-3}} \right] \quad [56]$$

12. As an alternative to this large sample approach, **Fisher z-transformation** is also frequently applied when the underlying data is **not normally distributed**. [56]

7.10.2. Kendall's Tau Correlation (concordance) (τ_K)

1. It is a measure of **concordance** [56]

2.

Definition 7.1 (Concordant & Discordant). A concordance measure is similar to correlation. To understand this alternative approach to dependency between X and Y , we consider two independent draws from F_{XY} , say (X_1, Y_1) and (X_2, Y_2) . Independence means here that X_1 and X_2 are independent and Y_1 and Y_2 are independent. The two pairs of random variables (X_1, Y_1) and (X_2, Y_2) are called concordant when $(X_2 - X_1)(Y_2 - Y_1) > 0$ and discordant when $(X_2 - X_1)(Y_2 - Y_1) < 0$. [56]

(a) If a pair is concordant or discordant, it means that the change in the two random variables $D_X = X_2 - X_1$ (from X_1 to X_2) and $D_Y = Y_2 - Y_1$ (from Y_1 to Y_2) are dependent. [56]

(b) Concordance refers to a “positive” dependency, which means that the direction of change in X is the same as the direction of change in Y . [56]

(c) Discordance means a “negative” dependence, where the direction of change in X is opposite to the direction of change in Y . [56]

3. Kendall's tau measures the strength between the dependency of D_X and D_Y [56]

$$\tau_K = P((X_2 - X_1)(Y_2 - Y_1) > 0) - P((X_2 - X_1)(Y_2 - Y_1) < 0)$$

$$= 2P((X_2 - X_1)(Y_2 - Y_1) > 0) - 1 \quad [56]$$

$$= 4P(X_1 < X_2, Y_1 < Y_2) - 1$$

5. τ_K varies between -1 and 1 [56]

6. the minimum -1 is attained when F_{XY} is equal to the Fréchet lower bound, i.e., [56]

$$F_{XY}(x, y) = \max\{F_X(x) + F_Y(y) - 1, 0\}. \quad [56]$$

7. The value 1 is attained when F_{XY} is equal to the Fréchet upper bound, i.e., [56]

$$F_{XY}(x, y) = \min\{F_X(x), F_Y(y)\} \quad [56]$$

8. When X and Y are **independent**, Kendall's tau is equal to **zero**. [56]

9. Pearson's correlation coefficient of D_X and D_Y is given by $CORR(D_X, D_Y) = CORR(X, Y)$. [56]

7.10.2.1. Kendall's Tau Correlation estimator (r_K)

$$r_K = \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n \text{sgn}(X_j - X_i) \text{sgn}(Y_j - Y_i)$$

$$= \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{sgn}(X_j - X_i) \text{sgn}(Y_j - Y_i) \quad [56]$$

2. r_K depends only on the signs of $X_j - X_i$ and $Y_j - Y_i$, which implies that the estimator is independent of monotone transformations (increasing or decreasing functions) of the data. If we apply the estimator

on $(\psi_1(X_1), \psi_2(Y_1)), (\psi_1(X_2), \psi_2(Y_2)), \dots, (\psi_1(X_n), \psi_2(Y_n))$, when both ψ_1 and ψ_2 are increasing or both decreasing functions, we obtain the exact same estimator. [56]

3. The distribution function of this estimator is approximately normal when sample size converges to infinity. Studying its moments is difficult, but the exact variance of r_K has been determined at $\frac{2(2n+5)}{3n(n-1)}$, when the pairs are uncorrelated (i.e., $\tau_K = 0$). The variance of r_K is bounded from above with $\mathbb{V}[r_K] \leq \frac{2(1-\tau_K^2)}{n}$ [56]
4. 100%(1 - α) **confidence interval** is provided based on the **Fisher z-transformation**. [56]

(a) The variance of r_K in the transformed Fisher z scale has been determined at $\frac{0.437}{n-4}$ [56]

(b) The 100%(1 - α) confidence interval is then determined by [56]

$$\left[z_{r_K} - z_{1-\alpha/2} \sqrt{\frac{0.437}{n-4}}, z_{r_K} + z_{1-\alpha/2} \sqrt{\frac{0.437}{n-4}} \right] \quad [56]$$

with $z_{r_K} = 0.5[\log(1+r_K) - \log(1-r_K)]$ and z_p the p -th quantile of the standard normal distribution. [56]

(c) In the original scale, the confidence interval becomes: [56]

$$\left[\frac{\exp(2[z_{r_K} - z_{1-\alpha/2} \sqrt{0.437/(n-4)}]) - 1}{\exp(2[z_{r_K} - z_{1-\alpha/2} \sqrt{0.437/(n-4)}]) + 1}, \frac{\exp(2[z_{r_K} + z_{1-\alpha/2} \sqrt{0.437/(n-4)}]) - 1}{\exp(2[z_{r_K} + z_{1-\alpha/2} \sqrt{0.437/(n-4)}]) + 1} \right] \quad [56]$$

7.10.3. Spearman's Rho Correlation (ρ_S)

1. Spearman's rho quantifies concordance of change in one dimension from the first to the second observation and change in the second dimension from the first to the third observation. [56]
2. To define Spearman's rho in terms of the joint CDF, we need to use three i.i.d. pairs (X_1, Y_1) , (X_2, Y_2) , and (X_3, Y_3) of random variables having joint distribution function F_{XY} , but we do not need to use all six random variables. [56]
3. $\rho_S = 2[P((X_1 - X_2)(Y_3 - Y_1) > 0) - P((X_1 - X_2)(Y_3 - Y_1) < 0)]$ [56]
4. It ranges from -1 to 1 . When X and Y are independent, Spearman's rho correlation coefficient becomes equal to zero. [56]
5. $-1 \leq 3\tau_K - 2\rho_S \leq 1$ [56]

7.10.3.1. Spearman's Rho Correlation estimator (r_S)

1. The estimator for Spearman's rho ρ_S is defined by the product-moment correlation coefficient on the ranks of the pairs of observations. It is referred to as **Spearman's rank correlation**. [56]
 2. If we assume that we observe the pairs of random variables (X_1, Y_1) , (X_2, Y_2) , \dots , (X_n, Y_n) , we can define the ranks for the x and y coordinates separately. [56]
 3. The rank R_k^X of X_k is the position of X_k in the ordered values $X_{(1)}, X_{(2)}, \dots, X_{(n)}$ from small to large. If we additionally define the rank R_k^Y for Y_k among the random variables $Y_{(1)}, Y_{(2)}, \dots, Y_{(n)}$, we have translated the pair (X_k, Y_k) to a pair of ranks (R_k^X, R_k^Y) . [56]
- Note:** if the same value occurs multiple times they all receive the same rank. [56]
- Example:** if we have observed five observations $x_1 = 5, x_2 = 1, x_3 = 7, x_4 = 2$, and $x_5 = 4$, then the observed ranks will be equal to $r_1^X = 4, r_2^X = 1, r_3^X = 5, r_4^X = 2$, and $r_5^X = 3$. [56]

$$4. r_S = \frac{\sum_{i=1}^n (R_i^X - \bar{R}^X)(R_i^Y - \bar{R}^Y)}{\sqrt{\sum_{i=1}^n (R_i^X - \bar{R}^X)^2 \sum_{i=1}^n (R_i^Y - \bar{R}^Y)^2}} = 1 - 6 \sum_{i=1}^n \frac{(R_i^Y - R_i^X)^2}{n^3 - n} \quad [56]$$

with \bar{R}^X and \bar{R}^Y the average ranks for the X and Y variables, i.e., $\bar{R}^X = \frac{1}{n} \sum_{k=1}^n R_k^X$ and $\bar{R}^Y = \frac{1}{n} \sum_{k=1}^n R_k^Y$. [56]

5. As the total number of ranks in a sample of n observations is equal to $\frac{n(n+1)}{2}$, the average ranks \bar{R}^X and \bar{R}^Y are equal to $\bar{R}^X = \bar{R}^Y = \frac{n+1}{2}$. [56]

6. Since Spearman's rank correlation depends on the ranks of the variables x and y , the estimator is independent of monotonic transformations (increasing or decreasing functions) of the data. Spearman rho's estimator r_S on $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ is exactly the same as Spearman rho's estimator r_S on the transformed pairs $(\psi_1(X_1), \psi_2(Y_1)), (\psi_1(X_2), \psi_2(Y_2)), \dots, (\psi_1(X_n), \psi_2(Y_n))$, when ψ_1 and ψ_2 are both increasing or both decreasing functions. [56]

$$P(R_n^X = n) = P(X_n = \max \{X_1, X_2, \dots, X_n\}) = P(X_n \geq X_1, X_n \geq X_2, \dots, X_n \geq X_{n-1}) \\ 7. = \int_{-\infty}^{\infty} P(X_n \geq X_1, X_n \geq X_2, \dots, X_n \geq X_{n-1} | X_n = x) f_{X_n}(x) dx \\ = \int_{-\infty}^{\infty} P(X_1 \leq x, X_2 \leq x, \dots, X_{n-1} \leq x | X_n = x) f_{X_n}(x) dx = \int_{-\infty}^{\infty} F_X^{n-1}(x) f_{X_n}(x) dx \quad [56]$$

with $f_{X_n} = f_X$, since X_1, X_2, \dots, X_n are i.i.d. with CDF F_X . [56]

8. The Fisher z-transformation of r_S uses normal confidence intervals with a variance $S_{r_S}^2$ of r_S in this transformed scale. Different suggestions for this variance have been proposed: $S_{r_S}^2 = 1.06/(n-3)$, $S_{r_S}^2 = (1+r_S^2/2)/(n-3)$, and $S_{r_S}^2 = (n-2)^{-1} + |z_{r_S}|/[6n+4\sqrt{n}]$, with z_{r_S} the Fisher z-transformation of r_S , i.e., $z_{r_S} = 0.5[\log(1+r_S) - \log(1-r_S)]$. [56]

(a) **100%(1 - α) confidence interval:** $(z_{r_S} - z_{1-\alpha/2} S_{r_S}, z_{r_S} + z_{1-\alpha/2} S_{r_S})$ [56]

(b) Taking the inverse Fisher z-transformation on these limits, a 100%(1 - α) confidence interval for ρ_S is constructed. This gives confidence limits: [56]

$$\left[\frac{\exp(2[z_{r_S} - z_{1-\alpha/2} S_{r_S}]) - 1}{\exp(2[z_{r_S} - z_{1-\alpha/2} S_{r_S}]) + 1}, \frac{\exp(2[z_{r_S} + z_{1-\alpha/2} S_{r_S}]) - 1}{\exp(2[z_{r_S} + z_{1-\alpha/2} S_{r_S}]) + 1} \right] \quad [56]$$

7.10.4. Cohen's Kappa Statistic (κ_C)

1. It is a measure of **agreement** [56]
2. It measures how much two observers or raters agree on the evaluation of the same set of n items or units into K exhaustive and mutually exclusive classes. Since there is also a chance of accidentally classifying units in the same category by the raters, he created a measure that quantifies how well the raters agree on classification that is corrected for accidental or chance agreement. [56]
3. This chance agreement has strong similarities with grade corrections in multiple choice exams. [56]
4. The bivariate random variables X and Y , which represents two readings on the same unit, both take their values in $\{1, 2, 3, \dots, K\}$. The data can be summarized in a $K \times K$ contingency table. The joint PDF is fully defined by the set of PDF parameters $p_{11}, p_{12}, \dots, p_{1K}, p_{21}, p_{22}, \dots, p_{2K}, \dots, p_{K1}, p_{K2}, \dots, p_{KK}$ given by $P(X = x, Y = y) = p_{xy} \geq 0$, and $\sum_{x=0}^K \sum_{y=0}^K p_{xy} = 1$. [56]
5. The probability p_{xy} represents the probability that a unit is classified by the first rater in class x and by the second rater in class y . [56]

6. the probability p_{kk} indicates the probability that both raters classify a unit in the same class k . This implies that when $p_{xy} = 0$ for every $x \neq y$, there will be no difference in classification between the two raters. [56]
7. when the ratings are **independent**, i.e., $P(X = x, Y = y) = f_X(x) f_Y(y)$ [56]
8. $p_O = \sum_{k=1}^K p_{kk}$: It represents the probability that both raters classify units in the same classes. When it is equal to 1, there is perfect agreement. [56]
9. $p_E = \sum_{k=1}^K f_X(k) f_Y(k)$: probability of correctly classifying units in the same classes [56]
10. $\kappa_C = \frac{p_O - p_E}{1 - p_E}$ [56]
11. kappa statistic can become smaller than zero which means **discordance** [56]
12. For ordinal random variables there exists a weighted version of Cohen's kappa. When X and Y are further apart, i.e., $|X - Y| > 1$, this seems to be a more serious misclassification than when X and Y just differ one class, i.e., $|X - Y| = 1$. [56]

$0.80 < \kappa_C \leq 1.00$	High agreement
$0.60 < \kappa_C \leq 0.80$	Substantial agreement
$0.40 < \kappa_C \leq 0.60$	Moderate agreement
$0.20 < \kappa_C \leq 0.40$	Fair agreement
$0 < \kappa_C \leq 0.20$	Poor agreement

Rule of thumb

7.10.4.1. Cohen's Kappa Statistic estimator (hat κ_C)

1. In cell $X = x$ and $Y = y$ of the contingency table, the number of pairs, say N_{xy} , from $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ with the combination (x, y) is reported. [56]

$$N_{xy} = \sum_{i=1}^n 1_{\{x\}}(X_i) 1_{\{y\}}(Y_i) \quad \text{where } 1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases} \quad [56]$$

2. the sum of all numbers in the contingency table should add up to n : $\sum_{x=1}^K \sum_{y=1}^K N_{xy} = n$. [56]

$$3. N_{x \cdot} = \sum_{y=1}^K N_{xy} \quad N_{\cdot y} = \sum_{x=1}^K N_{xy} \quad \hat{p}_O = \frac{1}{n} \sum_{k=1}^K N_{kk} \quad \hat{p}_E = \frac{1}{n^2} \sum_{k=1}^K N_k \cdot N_{\cdot k} \quad [56]$$

4. The distribution function of N_{xy} is **binomial** with parameters n and $p_{xy} = P(X = x, Y = y)$. [56]

5. The multivariate PMF for $N_{11}, N_{12}, \dots, N_{1K}, N_{21}, N_{22}, \dots, N_{2K}, \dots, N_{K1}, N_{K2}, \dots, N_{KK}$ has a **multinomial distribution** function given by [56]

$$P(N_{11} = n_{11}, N_{12} = n_{12}, \dots, N_{KK} = n_{KK}) = n! \prod_{x=1}^K \prod_{y=1}^K \left(\frac{p_{xy}^{n_{xy}}}{n_{xy}!} \right) \quad [56]$$

6. Cohen's Kappa statistic is now estimated by $\hat{\kappa}_C = \frac{\hat{p}_O - \hat{p}_E}{1 - \hat{p}_E}$ [56]

$$7. \mathbb{V}[\hat{\kappa}_C] \approx \frac{p_O(1 - p_O)}{n(1 - p_E)^2} \quad [56]$$

8. asymptotic $100\%(1 - \alpha)$ confidence interval: [56]
- $$\left[\hat{K}_C - z_{1-\alpha/2} \frac{\sqrt{\hat{p}_O(1 - \hat{p}_O)}}{(1 - \hat{p}_E)\sqrt{n}}, \hat{K}_C + z_{1-\alpha/2} \frac{\sqrt{\hat{p}_O(1 - \hat{p}_O)}}{(1 - \hat{p}_E)\sqrt{n}} \right] \quad [56]$$
- with z_p the p -th quantile of the standard normal distribution function. [56]

7.10.5. Risk Difference, Relative Risk, and Odds Ratio

1. We assume that the bivariate binary data $(X_i, Y_i) \in \{0, 1\} \times \{0, 1\}$ can be captured by the summary statistics [56]

$$\cdot \quad N_{xy} = \sum_{i=1}^n 1_{\{x\}}(X_i) 1_{\{y\}}(Y_i) \quad (x, y) \in \{0, 1\} \times \{0, 1\} \quad [56]$$

2. If we assume that Y represents some kind of outcome and X represents some kind of exposure, with $X = 0$ the reference group, the estimators of the risk difference (\hat{RD}), relative risk (\hat{RR}), and odds ratio (\hat{OR}) are defined by [56]

$$(a) \hat{RD} = \frac{N_{11}}{N_{11} + N_{10}} - \frac{N_{01}}{N_{01} + N_{00}} = \frac{N_{11}N_{00} - N_{10}N_{01}}{(N_{11} + N_{10})(N_{01} + N_{00})} \quad [56]$$

$$(b) \hat{RR} = N_{11}(N_{01} + N_{00})N_{01}(N_{11} + N_{10}) \quad [56]$$

$$(c) \hat{OR} = \frac{N_{11}N_{00}}{N_{10}N_{01}} \quad [56]$$

$$3. \hat{p}_0 = \frac{N_{01}}{N_{0.}} \quad \hat{p}_1 = \frac{N_{11}}{N_{1.}} \quad N_{x.} = N_{x1} + N_{x0} \quad [56]$$

$$4. L_x = \frac{2N_{x.}\hat{p}_x + z_{1-\alpha/2}^2 - z_{1-\alpha/2}\sqrt{4N_{x.}\hat{p}_x(1 - \hat{p}_x) + z_{1-\alpha/2}^2}}{2(N_{x.} + z_{1-\alpha/2}^2)} \quad [56]$$

$$U_x = \frac{2N_{x.}\hat{p}_x + z_{1-\alpha/2}^2 + z_{1-\alpha/2}\sqrt{4N_{x.}\hat{p}_x(1 - \hat{p}_x) + z_{1-\alpha/2}^2}}{2(N_{x.} + z_{1-\alpha/2}^2)} \quad [56]$$

for $x \in \{0, 1\}$ and z_p the p -th quantile of the standard normal distribution function. [56]

7.10.5.1. Risk Difference

5. The $100\%(1 - \alpha)$ confidence interval on the risk difference is then given by [56]

$$\cdot \quad \left[\hat{RD} - \sqrt{(\hat{p}_1 - L_1)^2 + (U_0 - \hat{p}_0)^2}, \hat{RD} + \sqrt{(U_1 - \hat{p}_1)^2 + (\hat{p}_0 - L_0)^2} \right] \quad [56]$$

7.10.5.2. Relative Risk

6. For the relative risk, the confidence interval is calculated in the **logarithmic scale**. [56]

7. The estimated standard error of the logarithmic transformed relative risk is given by [56]

$$\cdot \quad \hat{SE}_{RR} = \sqrt{\frac{1 - \hat{p}_0}{N_{0.}\hat{p}_0} + \frac{1 - \hat{p}_1}{N_{1.}\hat{p}_1}} \quad [56]$$

8. the asymptotic $100\%(1 - \alpha)$ confidence interval on the logarithmic transformation of the relative risk is given by [56]

$$\cdot \quad (L_{RR}, U_{RR}] = (\log(\hat{RR}) - z_{1-\alpha/2}\hat{SE}_{RR}, \log(\hat{RR}) + z_{1-\alpha/2}\hat{SE}_{RR}] \quad [56]$$

9. The $100\%(1 - \alpha)$ confidence interval on the relative risk is then calculated by transforming these confidence limits back to the original scale using $(\exp(L_{RR}), \exp(U_{RR}))$. [56]

7.10.5.3. Odds Ratio

10. The confidence interval on the odds ratio is also calculated in the logarithmic scale using the standard error [56]

$$\hat{SE}_{OR} = \sqrt{N_{00}^{-1} + N_{01}^{-1} + N_{10}^{-1} + N_{11}^{-1}} \quad [56]$$
11. 100%(1 – α) confidence interval on the odds ratio in the logarithmic scale is given by [56]

$$(L_{OR}, U_{OR}] = (\log(\hat{OR}) - z_{1-\alpha/2}\hat{SE}_{OR}, \log(\hat{OR}) + z_{1-\alpha/2}\hat{SE}_{OR}] \quad [56]$$
12. the 100%(1 – α) confidence interval on the odds ratio is then calculated by transforming these confidence limits back to the original scale using $(\exp(L_{OR}), \exp(U_{OR}))$. [56]

7.10.6. Comparison: Pearson's Rho (ρ_P), Spearman's Rho (ρ_S), Kendall's Tau Correlation (τ_K), Cohen's kappa (κ_C)

1. The **product-moment correlation** (r_P) provides a measure that is particularly suitable for **linear** associations between variables. [56]
2. when the data are bivariate normally distributed, or when transformations can be applied that would make the data approximately bivariate normal, it is suggested to apply the product-moment estimator on the (transformed) data. [56]
3. Pearson's product-moment estimator is sensitive to outliers. [56]
4. for non-normal bivariate data (that is difficult to transform into normally distributed data) it is probably best not to use Pearson's product-moment estimator. [56]
5. When to choose between Spearman's rank or Kendall's tau estimator: [56]
 - (a) Kendall's tau estimator has a clear interpretation. It measures the difference between concordant and discordant pairs. The interpretation of Spearman is more difficult, although it is often presented as a robust version of Pearson's product-moment estimator. [56]
 - (b) Kendall's tau estimator is computationally more intensive, since it requires the comparisons of pairs with all other pairs, which is not needed with Spearman's rank. Thus on really large data, Kendall's tau may lead to numerical issues. However, for large datasets Kendall's tau estimator is more efficient than Spearman's rank correlation [56]
 - (c) when the data contain ties (i.e., values among the x variable and/or values among the y variable are equal), Spearman's rank correlation has a smaller standard error than Kendall's tau estimator [56]
6. The simplicity of Cohen's kappa statistic has made it very popular in all kinds of sciences to quantify agreement. However, Cohen's kappa statistic should not be used when particular outcomes of X and Y are rare. [56]
7. Cohen's kappa works best when each outcome has approximately the same probability of occurrence, or is at least not strongly imbalanced. [56]

7.10.7. Nominal Association Statistics

1. useful for quantifying the dependency between two nominal random variables X and Y . [56]
2. They measure the departure from dependency and they are all functions of Pearson's chi-square statistic. [56]
3. Although they can be applied to nominal variables, they have also been used for ordinal and binary random variables X and Y . [56]
4. we assume that X can take its value in $\{1, 2, \dots, K\}$ and Y can take its value in $\{1, 2, \dots, M\}$ and we assume that $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ are i.i.d. with CDF F_{XY} . [56]

$$5. N_{xy} = \sum_{i=1}^n 1_{\{x\}}(X_i) 1_{\{y\}}(Y_i) \quad (x,y) \in \{1, 2, \dots, K\} \times \{1, 2, \dots, M\} \quad [56]$$

The numbers N_{xy} represent the frequencies of the cells in a KM contingency table, with K rows and M columns. [56]

$$6. N_{x \cdot} : \text{number of pairs } (X_i, Y_i) \text{ for which the } X \text{ variable is equal to } x, \text{ irrespective of the value of } Y \quad [56]$$

$$\cdot \quad N_{x \cdot} = \sum_{y=1}^M N_{xy} \quad N_{\cdot y} = \sum_{x=1}^K N_{xy} \quad [56]$$

7. If X and Y are **independent**, the joint PMF is the product of the two marginal PMFs: $P(X = x, Y = y) = P(X = x) P(Y = y)$ for all $(x, y) \in \{1, 2, \dots, K\} \times \{1, 2, \dots, M\}$. [56]

7.10.7.1. Cohen's kappa statistic

8. with Cohen's kappa statistic, the joint PMF $P(X = x, Y = y)$ can be estimated by N_{xy}/n and that the marginal PMFs $P(X = x)$ and $P(Y = y)$ can be estimated by $N_{x \cdot}/n$ and $N_{\cdot y}/n$, respectively. [56]

7.10.7.2. Pearson's chi-square statistic (χ_P^2)

9. Pearson's chi-square statistic quantifies the difference between the observed numbers N_{xy} and their expected numbers $N_{x \cdot}N_{\cdot y}/n$ based on independence. [56]

10. A part of Pearson's chi-square statistic is $[N_{xy} - N_{x \cdot}N_{\cdot y}/n]^2$. The square is used to make all differences positive. [56]

11. Pearson normalized the squared differences with the expected numbers. The reason is that a small squared difference between N_{xy} and $N_{x \cdot}N_{\cdot y}/n$ when $N_{x \cdot}N_{\cdot y}/n$ is small is not the same as a small squared difference between N_{xy} and $N_{x \cdot}N_{\cdot y}/n$ when $N_{x \cdot}N_{\cdot y}/n$ is large. [56]

12. Pearson's chi-square statistic: [56]

$$\cdot \quad \chi_P^2 = \sum_{x=1}^K \sum_{y=1}^M \frac{[N_{xy} - N_{x \cdot}N_{\cdot y}/n]^2}{N_{x \cdot}N_{\cdot y}/n} = \frac{1}{n} \sum_{x=1}^K \sum_{y=1}^M \frac{[nN_{xy} - N_{x \cdot}N_{\cdot y}]^2}{N_{x \cdot}N_{\cdot y}} \geq 0 \quad [56]$$

13. χ_P^2 is equal to zero only when the observed frequencies in the cells of the contingency table are equal to the expected frequencies. [56]

14. Thus the larger the value for Pearson's chi-square statistic, the stronger the association between the two random variables X and Y . Since it calculates the differences between observed frequencies and frequencies under the assumption of independence, Pearson's chi-square represents a measure of departure from independence. [56]

15. The CDF of Pearson's chi-square statistic can be approximated with a chi-square distribution with $(K - 1)(M - 1)$ degrees of freedom when X and Y are independent. [56]

16. a disadvantage of Pearson's chi-square statistic is that it cannot be viewed as a correlation coefficient, like Kendall's tau and Spearman's rho, since it is not limited by the value 1. [56]

17. In the case of two categories $K = M = 2$, the numerator $[nN_{xy} - N_{x \cdot}N_{\cdot y}]^2 = [N_{11}N_{22} - N_{12}N_{21}]^2$ for each combination of $(x, y) \in \{1, 2\} \times \{1, 2\}$. [56]

18. Pearson's product-moment estimator can be written in terms of the contingency table frequencies N_{xy} . [56]

19. for binary X and Y , Pearson's chi-square statistic becomes: $\chi_P^2 = n \frac{[N_{11}N_{22} - N_{12}N_{21}]^2}{N_{1 \cdot}N_{2 \cdot}N_{\cdot 1}N_{\cdot 2}}$ [56]

20. For binary random variables X and Y : [56]

$$\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y}) = [N_{11}N_{22} - N_{12}N_{21}]/n \quad [56]$$

$$\sum_{i=1}^n (X_i - \bar{X})^2 = N_{\cdot 1}N_{\cdot 1}/n \quad \sum_{i=1}^n (Y_i - \bar{Y})^2 = N_{\cdot 2}N_{\cdot 2}/n \quad [56]$$

$$21. \rho_P = \frac{[N_{11}N_{22} - N_{12}N_{21}]}{\sqrt{N_{\cdot 1}N_{\cdot 2}N_{\cdot 1}N_{\cdot 2}}} \quad [56]$$

22. normalization of Pearson's chi-square statistic with n was not ideal. maximum value of Pearson's chi-square statistic could become maximally equal to $n \min\{K-1, M-1\}$. [56]

7.10.7.3. Pearson's squared phi-coefficient (ϕ^2)

$$23. \phi^2 = \frac{\chi_P^2}{n} \text{ with the sample size } n \quad [56]$$

$$24. \text{ for binary } X \text{ and } Y: \phi = \frac{|N_{11}N_{22} - N_{12}N_{21}|}{\sqrt{N_{\cdot 1}N_{\cdot 2}N_{\cdot 1}N_{\cdot 2}}} \quad [56]$$

7.10.7.4. Cramér's V (V)

$$25. V = \sqrt{\frac{\chi_P^2}{n \min\{K-1, M-1\}}} \quad [56]$$

26. Cramér's V is equal to Pearson's ϕ coefficient when either the X or the Y variable has just two levels. Cramér's V is the more general measure of nominal association, since it is properly normalized for all contingency tables and it reduces to Pearson's ϕ for 2×2 contingency tables. [56]

7.10.8. Ordinal Association Statistics

- Let X and Y be ordinal random variables, X and Y can take their values in $\{1, 2, \dots, K\}$ and $\{1, 2, \dots, M\}$, respectively. [56]

7.10.8.1. Goodman and Kruskal's gamma (γ)

$$2. \gamma = \frac{P((X_2 - X_1)(Y_2 - Y_1) > 0) - P((X_2 - X_1)(Y_2 - Y_1) < 0)}{P((X_2 - X_1)(Y_2 - Y_1) > 0) + P((X_2 - X_1)(Y_2 - Y_1) < 0)} \quad [56]$$

$$3. N_C = \sum_{i=1}^n \sum_{j=1}^n 1_{(0, \infty)}((X_j - X_i)(Y_j - Y_i)) \quad N_D = \sum_{i=1}^n \sum_{j=1}^n 1_{(-\infty, 0)}((X_j - X_i)(Y_j - Y_i)) \quad [56]$$

N_C and N_D represent the number of concordant and discordant pairs. The sample size n is typically larger than the sum of concordant and discordant pairs, i.e., $n > N_C + N_D$, due to the many ties in data of the $K \times M$ contingency table. [56]

$$4. \gamma \text{ can be estimated by } G = \frac{N_C - N_D}{N_C + N_D} \quad [56]$$

- if there are no ties at all, Goodman and Kruskal's gamma reduces to Kendall's tau. [56]
- Let distribution function of the estimator G and $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ are i.i.d. with distribution function F_{XY} . [56]
- The distribution function of $\sqrt{n}(G - \gamma)$ is asymptotically normal with zero mean and a variance that is smaller than but close to: [56]

$$\frac{2(1 - \gamma^2)}{P((X_2 - X_1)(Y_2 - Y_1) > 0) + P((X_2 - X_1)(Y_2 - Y_1) < 0)} \quad [56]$$

This variance can be estimated with $\frac{2(1-G^2)}{N_D+N_C}$. [56]

8. a $100\%(1-\alpha)$ confidence interval on γ using the theory of asymptotic confidence intervals: [56]

$$\left[G - z_{1-\alpha/2} \sqrt{\frac{2(1-G^2)}{N_D+N_C}}, G + z_{1-\alpha/2} \sqrt{\frac{2(1-G^2)}{N_D+N_C}} \right] \quad [56]$$

with z_p the p -th quantile of the standard normal distribution function. [56]

7.10.9. Binary Association Statistics

1. Lets assume that we observed n binary attributes on two objects, leading to the n pairs $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$. In this type of application, the pairs may not be independent and/or identically distributed with just one CDF F_{XY} . [56]

$$2. N_{xy} = \sum_{i=1}^n 1_{\{x\}}(X_i) 1_{\{y\}}(Y_i) \quad (x,y) \in \{0,1\} \times \{0,1\} \quad [56]$$

3. N_{11} is the number of attributes that both objects share and N_{00} is the number of attributes that both objects lack. N_{01} and N_{10} represent the numbers of objects that are present in one object but absent in the other object. [56]

7.10.9.1. Gower and Legendre (S_θ, T_θ)

$$4. S_\theta = \frac{N_{00} + N_{11}}{N_{00} + N_{11} + \theta[N_{01} + N_{10}]} \quad T_\theta = \frac{N_{11}}{N_{11} + \theta[N_{01} + N_{10}]} \quad [56]$$

with $\theta > 0$ a constant. [56]

5. The similarity measure S_θ focuses on the similarity of both the absence as well as the presence of attributes, since it uses both N_{00} and N_{11} , while the similarity measure T_θ focuses only on the presence of attributes. [56]

6. when we wish to emphasize the similarity on attributes that are present, the set of T_θ is more appropriate than S_θ . [56]

7. both similarity measures are bounded from below with the value zero and from above with the value 1 ($S_\theta, T_\theta \in [0, 1]$). [56]

8. The closer the similarity measures get to one the more similar the two objects would be on the presence and absence of attributes. [56]

9. the dissimilarity measures $1 - S_\theta$ and $1 - T_\theta$ are distance measures when $\theta \geq 1$ and $\sqrt{1 - S_\theta}$ and $\sqrt{1 - T_\theta}$ are distance measures when $\theta \geq \frac{1}{3}$. [56]

a distance measure d is defined by four characteristics: [56]

$$(a) d(a, b) \geq 0 \quad [56] \quad (c) d(a, b) = d(b, a) \quad [56] \\ (b) d(a, a) = 0 \quad [56] \quad (d) d(a, b) + d(b, c) \geq d(a, c) \quad [56]$$

with a, b , and c representing arbitrary objects on which the distance is applied. [56]

$$10. \textbf{Sokal & Sneath (2): } S_{0.5} = \frac{2[N_{00} + N_{11}]}{2[N_{00} + N_{11}] + N_{01} + N_{10}} \quad [56]$$

$$11. \textbf{Sokal & Michener: } S_1 = \frac{N_{00} + N_{11}}{N_{00} + N_{11} + N_{01} + N_{10}} \quad [56]$$

12. **Roger & Tanimoto:** $S_2 = \frac{N_{00} + N_{11}}{N_{00} + N_{11} + 2[N_{01} + N_{10}]}$ [56]
13. **Czekanowski:** $T_{0.5} = \frac{2N_{11}}{2N_{11} + N_{01} + N_{10}}$ [56]
14. **Jaccard:** $T_1 = \frac{N_{11}}{N_{11} + N_{01} + N_{10}}$ [56]
15. **Sokal & Sneath (1):** $T_2 = \frac{N_{11}}{N_{11} + 2[N_{01} + N_{10}]}$ [56]
16. **Sokal & Sneath (3):** $S_{SS} = \frac{N_{00} + N_{11}}{N_{01} + N_{10}}$ [56]
17. **Kulcynski:** $T_K = \frac{N_{11}}{N_{01} + N_{10}}$ [56]

7.10.9.2. Pearson's squared phi-coefficient (ϕ^2)

18. It is clear that ϕ is an alternative measure with values in $[0, 1]$. [56]
19. As it is not of the form $\sqrt{1 - S_\theta}$ or $\sqrt{1 - T_\theta}$, it is unknown if $1 - \phi$ or $\sqrt{1 - \phi}$ is a distance measure, like the proposed similarity measures for values of θ being large enough. [56]

7.10.9.3. L family of similarity indices

20. \mathcal{L} family of similarity indices: $S = \lambda + \mu(N_{00} + N_{11})$ [56]
where the parameters λ and μ can only be functions of the row and column totals, i.e., functions of $N_{0.}, N_{1.}, N_{.0}$, and $N_{.1}$. [56]
21. **Sokal & Michener:** $\lambda = 0$ $\mu = \frac{1}{n}$ [56]
22. **Czekanowski:** $\lambda = 1 - \frac{1}{N_{1.} + N_{.1}}$ $\mu = \frac{1}{N_{1.} + N_{.1}}$ [56]
23. **Kappa:** $\lambda = -\frac{N_{1.}N_{.1} + N_{0.}N_{.0}}{N_{1.}N_{.0} + N_{.1}N_{.0}}$ $\mu = \frac{1}{N_{1.}N_{.0} + N_{.1}N_{.0}}$ [56]
24. corrected index of \mathcal{L} is of the form $CS = \frac{S - \mathbb{E}[S]}{1 - \mathbb{E}[S]}$ where $S = \lambda + \mu(N_{00} + N_{11})$ [56]

7.10.9.4. Yule's Q statistic (Q)

25. It was termed the coefficient of association, which made sense at the time, as it is one of the oldest measures of association. [56]
26. $Q = \frac{N_{00}N_{11} - N_{01}N_{10}}{N_{00}N_{11} + N_{01}N_{10}}$ [56]
27. The measure ranges from -1 to $+1$ and when Q is zero there is no association, similar to Pearson's, Spearman's, and Kendall's correlation coefficients. [56]
28. Neither is $1 - Q$ or $\sqrt{1 - Q}$ a distance measure, which makes it also different from Pearson's product-moment estimator on binary data. [56]
29. Yule's Q can be rewritten as $Q = \frac{\hat{OR} - 1}{\hat{OR} + 1}$ [56]
- Yule's Q is a monotone function of the odds ratio [56]
30. Yule's Q statistic is robust against the number of features that are present in one object, a characteristic that does not hold for the similarity measures. **Example:** if we randomly eliminate attributes from one

object, say remove half of all the attributes from object 1, then both N_{11} and N_{10} would reduce by a factor 2, but Yule's Q would not reduce. [56]

31. $100\%(1 - \alpha)$ confidence interval: [56]

$$(a) Q = \frac{\hat{OR} - 1}{\hat{OR} + 1} \implies \hat{OR} = -\frac{Q + 1}{Q - 1} \quad [56]$$

$$(b) \text{ Then Use: } (L_{OR}, U_{OR}) = (\log(\hat{OR}) - z_{1-\alpha/2} \hat{SE}_{OR}, \log(\hat{OR}) + z_{1-\alpha/2} \hat{SE}_{OR}] \quad [56]$$

7.11. Change of Variables

1. two approaches for obtaining distributions of transformations of random variables: [49]
 - (a) a direct approach using the definition of a cumulative distribution function [49]
 - (b) change-of-variable approach that uses the chain rule of calculus [49]
2. The change-of-variable approach is widely used because it provides a “recipe” for attempting to compute the resulting distribution due to a transformation. [49]
3. Moment generating functions (MGFs) can also be used to study transformations of random variables [49]

CHAPTER 8

DISCRETE DISTRIBUTIONS

8.1. Bernoulli distribution ($X \sim B(p)$)

8.1.1. Distribution of the Sample Statistic (T_n)

8.1.1.1. Distribution of the Sample Average

1. Let X_1, X_2, \dots, X_n are i.i.d. Bernoulli distributed, $X_i \sim \mathcal{B}(p)$, the sum of the random variables is binomial $\text{Bin}(n, p)$ distributed. [56]
2. This implies that the distribution function of the sample average \bar{X} of Bernoulli distributed random variables in $x \in [0, 1]$ is given by [56]

$$F_{\bar{X}}(x) = P(\bar{X} \leq x) = P(X_1 + X_2 + \dots + X_n \leq nx) = \sum_{k=0}^{\lfloor nx \rfloor} \binom{n}{k} p^k (1-p)^{n-k} \quad [56]$$

8.1.2. Maximum Likelihood Estimation (MLE)

1. Let X_1, X_2, \dots, X_n having an i.i.d. Bernoulli $\mathcal{B}(p)$ distribution. We would like to estimate the parameter p . [56]
2. If we obtain a realization x_1, x_2, \dots, x_n , we could ask how likely it is that we observe this set of results for given values of p . This so-called **likelihood** of the data is given by [56]
3. the term $p^{x_i}(1-p)^{1-x_i}$ is the probability that the random variable X_i will attain the realization x_i , i.e. $P(X_i = x_i) = p^{x_i}(1-p)^{1-x_i}$, with $x_i \in \{0, 1\}$. [56]
4. The product $L(p)$ represents the probability that X_1, X_2, \dots, X_n will attain the realization x_1, x_2, \dots, x_n . [56]
5. exploiting the i.i.d. assumption:

- (a) we assume identical distributions for each random variable X_i (and we thus work with parameters p as opposed to p_i) [56]
- (b) the probability of the joint observations is given by the product over the probability of the individual observations; this is possible by virtue of the independence assumption [56]
6. Since the likelihood is a function of the parameter p , we can search for a p that would maximize the likelihood. [56]
7. The maximum likelihood estimate \hat{p} is the value that maximizes $L(p)$ [56]
8. To obtain this maximum, it is more convenient to take the logarithm of the likelihood $\ell(p)$ given by [56]

$$\ell(p) = \sum_{i=1}^n [x_i \log(p) + (1-x_i) \log(1-p)] \quad [56]$$

The logarithm of a function achieves its maximum value at the same points as the function itself, but is **easier** to work with analytically as we can replace the multiplications by sums that are easier to differentiate. [56]

9. the maximum of a function can be obtained by taking the derivative and then equating it to zero and solving the equation for the variable of interest [56]

$$\begin{aligned} \frac{d\ell(p)}{dp} &= \ell'(p) = \sum_{i=1}^n \left(\frac{x_i}{p} - \frac{1-x_i}{1-p} \right) = 0 \Leftrightarrow (1-p) \sum_{i=1}^n x_i - np + p \sum_{i=1}^n x_i = 0 \\ &\Leftrightarrow np = \sum_{i=1}^n x_i \Leftrightarrow p = \frac{1}{n} \sum_{i=1}^n x_i = \bar{x} \end{aligned} \quad [56]$$

10. The maximum likelihood estimate is now given by $\hat{p} = \bar{x}$. Since any value $\bar{x} - \epsilon, \epsilon > 0$, for p in the derivative gives a positive value the solution is a maximum. The ML estimator is now $\hat{p} = \bar{X}$, which is the same as the MME, since the first moment of a Bernoulli random variable is p . [56]

8.1.3. Estimating the Parameter using Bayesian Methods

1. Let θ denote the probability of heads when throwing a (possibly unfair) coin, and let us treat the observations X_1, \dots, X_n from the coin as i.i.d. $\mathcal{B}(\theta)$. [56]

2. Using x to denote the vector of all x_i realizations, we have: [56]

$$\ell(\theta) = P(x|\theta) = \prod_{i=1}^n \theta^{x_i} (1-\theta)^{1-x_i} = \theta^{[\sum_{i=1}^n x_i]} (1-\theta)^{[n - \sum_{i=1}^n x_i]} = \theta^{n\bar{x}_n} (1-\theta)^{[n(1-\bar{x}_n)]} \quad [56]$$

3. Let prior be the support (i.e., $f(\theta) > 0$) for all plausible values of θ . As in our case θ is a Bernoulli p for which $0 \leq p \leq 1$ we should choose $f(\theta)$ such that it has support on $[0, 1]$. It seems reasonable, in the absence of any other information, to choose a prior distribution that gives equal likelihood to all plausible values of θ : the uniform distribution function on $[0, 1]$ thus seems a reasonable choice. The uniform on $[0, 1]$ is however a special case of a much more flexible distribution function called the **beta distribution** function. [56]

4. The Beta(α, β) distribution function is a continuous distribution function with PDF: ($\alpha > 0$ and $\beta > 0$) [56]

$$f(\theta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} \quad [56]$$

for $0 < \theta < 1$ and zero otherwise. Gamma function $\Gamma(x) = \int_0^\infty s^{x-1} e^{-s} ds$. When we choose $\alpha = 1$ and $\beta = 1$ the resulting beta distribution function is actually a **uniform distribution** function on $[0, 1]$. [56]

5. The Beta($1, 1$) is considered a relatively **uninformative prior** for the Bernoulli p . The density with its peak at 0.5 is given by $\alpha = \beta = 5$; as long as both parameters of the beta are equal and larger than 1 the maximum of the density will be 0.5. [56]

6. **posterior:** [56]

$$\begin{aligned} f(\theta|x) &\propto P(x|\theta)f(\theta) \\ &= \theta^{(\sum_{i=1}^n x_i)} (1-\theta)^{(n - \sum_{i=1}^n x_i)} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} \\ &\propto \theta^{(\alpha + \sum_{i=1}^n x_i) - 1} (1-\theta)^{(\beta + n - \sum_{i=1}^n x_i) - 1} \end{aligned} \quad [56]$$

7. the posterior $f(\theta|D)$ is proportional to the density of a Beta(α', β') distribution, with [56]

$$(a) \alpha' = \alpha + \sum_{i=1}^n x_i \quad [56]$$

$$(b) \beta' = \beta + n - \sum_{i=1}^n x_i \quad [56]$$

(c) the Normalizing constant will be the Normalizing constant of the beta. [56]

8. the beta prior can be regarded as adding additional observations where α specifies the a-priori number of successes, and β specifies the a-priori number of failures. [56]

8.1.4. Bernoulli as Exponential Family

1. Bernoulli distribution: $P(x|\mu) = \mu^x(1-\mu)^{1-x}$ $x \in \{0, 1\}$ [49]

2. Now: [49]

$$\begin{aligned} P(x|\mu) &= \exp[\log(\mu^x(1-\mu)^{1-x})] = \exp[x \log \mu + (1-x) \log(1-\mu)] \\ &= \exp[x \log \mu - x \log(1-\mu) + \log(1-\mu)] = \exp \left[x \log \frac{\mu}{1-\mu} + \log(1-\mu) \right] \end{aligned} \quad [49]$$

This can be identified as being in exponential family form by observing that: [49]

(a) $h(x) = 1$ [49]	(c) $\phi(x) = x$ [49]
(b) $\theta = \frac{\mu}{1-\mu}$ [49]	(d) $A(\theta) = -\log(1-\mu) = \log(1+\exp(\theta))$ [49]

3. $\mu = \frac{1}{1 + \exp(-\theta)}$ [49]

The relationship between the original Bernoulli parameter μ and the natural parameter θ is known as the **sigmoid** or **logistic function**. [49]

4. The canonical conjugate prior has the form: [49]

$$\begin{aligned} P(\mu|\alpha, \beta) &= \frac{\mu}{1-\mu} \exp \left(\alpha \log \frac{\mu}{1-\mu} + (\beta + \alpha) \log(1-\mu) - A_c(\gamma) \right) \\ &= \exp[(\alpha - 1) \log \mu + (\beta - 1) \log(1-\mu) - A_c(\alpha, \beta)] \\ &\propto \mu^{\alpha-1} (1-\mu)^{\beta-1} \end{aligned} \quad [49]$$

where we defined $\gamma := \begin{bmatrix} \alpha \\ \beta + \alpha \end{bmatrix}$ and $h_c(\mu) := \frac{\mu}{1-\mu}$. [49]

5. conjugate prior is Beta distribution [49]

8.1.5. Summary

1. **Notation:** $\mathcal{B}(p)$ [23]

2. **Parameters:** $0 \leq p \leq 1$ $q = 1 - p$ [23]

3. **Support:** $k \in \{0, 1\}$ [23]

4. **PMF:** $\begin{cases} q = 1 - p & \text{if } k = 0 \\ p & \text{if } k = 1 \end{cases}$ [23]

5. **CDF:** $\begin{cases} 0 & \text{if } k < 0 \\ 1-p & \text{if } 0 \leq k < 1 \\ 1 & \text{if } k \geq 1 \end{cases}$ [23]

6. **Mean:** p [23]

7. **Median:** $\begin{cases} 0 & \text{if } p < 1/2 \\ [0, 1] & \text{if } p = 1/2 \\ 1 & \text{if } p > 1/2 \end{cases}$ [23]

8. **Mode:** $\begin{cases} 0 & \text{if } p < 1/2 \\ 0, 1 & \text{if } p = 1/2 \\ 1 & \text{if } p > 1/2 \end{cases}$ [23]

9. **Variance:** $p(1-p) = pq$ [23]

-
- 10. **Median absolute deviation (MAD):** $2p(1-p) = 2pq$ [23]
 - 11. **Skewness:** $\frac{q-p}{\sqrt{pq}}$ [23]
 - 12. **Excess kurtosis:** $\frac{1-6pq}{pq}$ [23]
 - 13. **Entropy:** $-q \ln q - p \ln p$ [23]
 - 14. **Moment-generating function (MGF):** $q + pe^t$ [23]
 - 15. **Characteristic function (CF):** $q + pe^{it}$ [23]
 - 16. **Probability-generating function (PGF):** $q + pz$ [23]
 - 17. **Fisher information:** $\frac{1}{pq}$ [23]

8.2. Binomial distribution ($X \sim B(n, p)$)

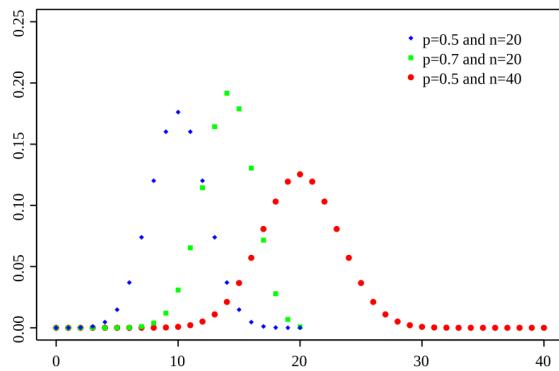


Fig. 8.1: Binomial Distribution: PDF [25]

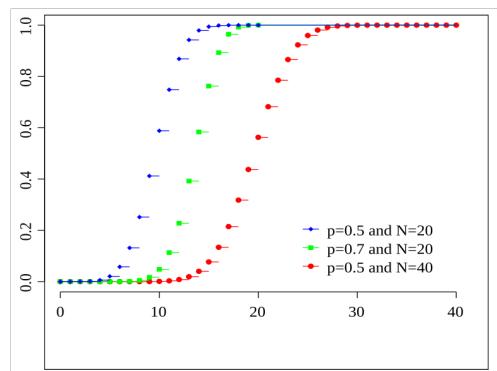


Fig. 8.2: Binomial Distribution: CDF [25]

1. A **disadvantage** of a random variable with a binomial distribution function is that it is bounded by the number of trials n . [56]

8.2.1. PMF

1. binomial random variable S_n is the sum of n independent Bernoulli variables [56]
 2. The random variable S_n is then given by $S_n = X_1 + X_2 + \dots + X_n$, with X_k the binary random variable for turn k [56]
 3. The PMF of the binomial is given by: [56]
- $$P(S_n = s) = f_{n,p}(s) = \binom{n}{s} p^s (1-p)^{n-s}$$
- $$\binom{n}{s} = \frac{n!}{s!(n-s)!}$$

8.2.2. Summary

1. **Notation:** $\mathcal{B}(n, p)$ [25]
2. **Parameters:**
 - (a) $n \in \{0, 1, 2, \dots\}$ – number of trials [25]
 - (b) $p \in [0, 1]$ – success probability for each trial [25]
 - (c) $q = 1 - p$ - failure probability for each trial [25]
3. **Support:** $k \in \{0, 1, \dots, n\}$ – number of successes [25]
4. **PMF:** $\binom{n}{k} p^k q^{n-k}$ [25]
5. **CDF:** $I_q(n - \lfloor k \rfloor, 1 + \lfloor k \rfloor)$ (the regularized incomplete beta function) [25]
6. **Mean:** np [25]
7. **Median:** $\lfloor np \rfloor$ or $\lceil np \rceil$ [25]
8. **Mode:** $\lfloor (n+1)p \rfloor$ or $\lceil (n+1)p \rceil - 1$ [25]
9. **Variance:** $npq = np(1-p)$ [25]
10. **Skewness:** $\frac{q-p}{\sqrt{npq}}$ [25]

-
11. **Excess kurtosis:** $\frac{1 - 6pq}{npq}$ [25]
12. **Entropy:** $\frac{1}{2} \log_2(2\pi enpq) + O\left(\frac{1}{n}\right)$ in shannons. For nats, use the natural log in the log. [25]
13. **Moment-generating function (MGF):** $(q + pe^t)^n$ [25]
14. **Characteristic function (CF):** $(q + pe^{it})^n$ [25]
15. **Probability-generating function (PGF):** $G(z) = [q + pz]^n$ [25]
16. **Fisher information:** $g_n(p) = \frac{n}{pq}$ (for fixed n) [25]

8.3. Multinomial distribution

8.3.1. Summary

1. **Parameters:**
 - (a) $n \in \{0, 1, 2, \dots\}$ number of trials [31]
 - (b) $k > 0$ number of mutually exclusive events (integer) [31]
 - (c) p_1, \dots, p_k event probabilities, where $p_1 + \dots + p_k = 1$ [31]
2. **Support:** $\left\{ (x_1, \dots, x_k) \mid \sum_{i=1}^k x_i = n, x_i \geq 0 \ (i = 1, \dots, k) \right\}$ [31]
3. **PMF:** $\frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$ [31]
4. **Mean:** $E(X_i) = np_i$ [31]
5. **Variance:** $\text{Var}(X_i) = np_i(1 - p_i)$, $\text{Cov}(X_i, X_j) = -np_i p_j \ (i \neq j)$ [31]
6. **Entropy:** $-\log(n!) - n \sum_{i=1}^k p_i \log(p_i) + \sum_{i=1}^k \sum_{x_i=0}^n \binom{n}{x_i} p_i^{x_i} (1 - p_i)^{n-x_i} \log(x_i!)$ [31]
7. **Moment-generating function (MGF):** $\left(\sum_{i=1}^k p_i e^{t_i} \right)^n$ [31]
8. **Characteristic function (CF):** $\left(\sum_{j=1}^k p_j e^{it_j} \right)^n$ where $i^2 = -1$ [31]
9. **Probability-generating function (PGF):** $\left(\sum_{i=1}^k p_i z_i \right)^n \ (z_1, \dots, z_k) \in \mathbb{C}^k$ [31]

8.4. Negative binomial distribution ($NB(r, p)$)

8.4.1. PMF

1. The negative binomial PMF is often considered a Poisson PMF with an extra amount of variation [56]

2. **Parameters:**

(a) λ : mean, the same as for the Poisson random variable [56]

(b) δ : overdispersion parameter, indicating the extra amount of variation on top of the Poisson variation. [56]

3. A negative binomial random variable X has its outcomes in the set $\{0, 1, 2, 3, \dots\}$, like the Poisson random variable. The PMF is defined by: [56]

$$P(X = x) = f_{\lambda, \delta}(k) = \frac{\Gamma(k + \delta^{-1})}{\Gamma(k + 1) \Gamma(\delta^{-1})} \frac{(\delta\lambda)^k}{(1 + \delta\lambda)^{k + \delta^{-1}}} \quad [56]$$

4. $\mu = \mathbb{E}[X] = \lambda$ [56]

5. $\sigma^2 = \mathbb{E}[(X - \lambda)^2] = \lambda + \delta\lambda^2$ [56]

6. In case the parameter δ converges to zero, the variance converges to the variance of a Poisson random variable. This is the reason that the parameter δ is called the overdispersion.

8.4.2. Summary

1. **Notation:** $NB(r, p)$ [33]

2. **Parameters:** $r > 0$ — number of successes until the experiment is stopped (integer, but the definition can also be extended to reals) $p \in [0, 1]$ — success probability in each experiment (real) [33]

3. **Support:** $k \in \{0, 1, 2, 3, \dots\}$ — number of failures [33]

4. **PMF:** $k \mapsto \binom{k+r-1}{k} \cdot (1-p)^k p^r$ involving a binomial coefficient [33]

5. **CDF:** $k \mapsto I_p(r, k+1)$ the regularized incomplete beta function [33]

6. **Mean:** $\frac{r(1-p)}{p}$ [33]

7. **Mode:** $\begin{cases} \left\lfloor \frac{(r-1)(1-p)}{p} \right\rfloor & \text{if } r > 1 \\ 0 & \text{if } r \leq 1 \end{cases}$ [33]

8. **Variance:** $\frac{r(1-p)}{p^2}$ [33]

9. **Skewness:** $\frac{2-p}{\sqrt{(1-p)r}}$ [33]

10. **Excess kurtosis:** $\frac{6}{r} + \frac{p^2}{(1-p)r}$ [33]

11. **Moment-generating function (MGF):** $\left(\frac{p}{1-(1-p)e^t} \right)^r$ for $t < -\log(1-p)$ [33]

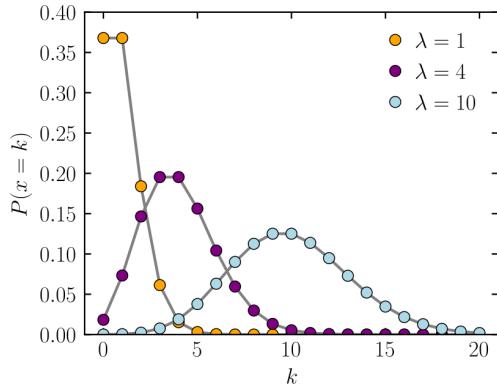
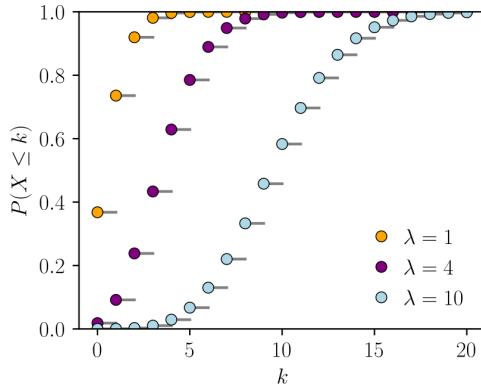
12. **Characteristic function (CF):** $\left(\frac{p}{1-(1-p)e^{it}} \right)^r$ with $t \in \mathbb{R}$ [33]

13. **Probability-generating function (PGF):** $\left(\frac{p}{1-(1-p)z}\right)^r$ for $|z| < \frac{1}{p}$ [33]

14. **Fisher information:** $\frac{r}{p^2(1-p)}$ [33]

15. **Method of moments:** $r = \frac{\mathbb{E}[X]^2}{\mathbb{V}[X] - \mathbb{E}[X]}$ $p = \frac{\mathbb{E}[X]}{\mathbb{V}[X]}$ [33]

8.5. Poisson distribution

**Fig. 8.3:** Poisson distribution: PDF [35]**Fig. 8.4:** Poisson distribution: CDF [35]

8.5.1. Summary

1. **Notation:** $\text{Pois}(\lambda)$ [35]
2. **Parameters:** $\lambda \in (0, \infty)$ (rate) [35]
3. **Support:** $k \in \mathbb{N}_0$ (Natural numbers starting from 0) [35]
4. **PMF:** $\frac{\lambda^k e^{-\lambda}}{k!}$ [35]
5. **CDF:** $\frac{\Gamma(\lfloor k+1 \rfloor, \lambda)}{\lfloor k \rfloor !}$, or $e^{-\lambda} \sum_{j=0}^{\lfloor k \rfloor} \frac{\lambda^j}{j!}$, or $Q(\lfloor k+1 \rfloor, \lambda)$ (for $k \geq 0$, where $\Gamma(x, y)$ is the upper incomplete gamma function, $\lfloor k \rfloor$ is the floor function, and Q is the regularized gamma function) [35]
6. **Mean:** λ [35]
7. **Median:** $\approx \left\lfloor \lambda + \frac{1}{3} - \frac{1}{50\lambda} \right\rfloor$ [35]
8. **Mode:** $\lceil \lambda \rceil - 1, \lfloor \lambda \rfloor$ [35]
9. **Variance:** λ [35]
10. **Skewness:** $\frac{1}{\sqrt{\lambda}}$ [35]
11. **Excess kurtosis:** $\frac{1}{\lambda}$ [35]
12. **Entropy:** $\lambda \left[1 - \log(\lambda) \right] + e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k \log(k!)}{k!}$ or for large λ $\approx \frac{1}{2} \log(2\pi e \lambda) - \frac{1}{12\lambda} - \frac{1}{24\lambda^2} - \frac{19}{360\lambda^3} + \mathcal{O}\left(\frac{1}{\lambda^4}\right)$ [35]
13. **Moment-generating function (MGF):** $\exp[\lambda(e^t - 1)]$ [35]
14. **Characteristic function (CF):** $\exp[\lambda(e^{it} - 1)]$ [35]
15. **Probability-generating function (PGF):** $\exp[\lambda(z - 1)]$ [35]
16. **Fisher information:** $\frac{1}{\lambda}$ [35]

8.5.2. Standard Error of MLE

1. let X_1, X_2, \dots, X_n be i.i.d. Poisson $\mathcal{P}(\lambda)$ distributed. [56]
2. population mean $\mu(f) = \lambda$ and the population variance $\sigma^2(f) = \lambda$. [56]
3. The maximum likelihood estimator for $\theta = \lambda$, which is also the moment estimator, is given by the sample average $\hat{\lambda} = \bar{X}$. [56]
4. the log likelihood is $\ell_\theta = \sum_{i=1}^n [X_i \log(\lambda) - \log(X_i!) - \lambda]$. [56]
5. Equating the derivative with respect to λ to zero results into $\sum \left(\frac{X_i}{\lambda} - 1 \right) = 0$. Solving this for λ gives $\hat{\lambda} = \bar{X}$. [56]
6. As the distribution of $\frac{\sqrt{n}(\bar{X} - \mu(f))}{\sigma(f)}$ converges to a standard normal distribution, $\sqrt{n}(\bar{X} - \lambda)$ converges to $\mathcal{N}(0, \lambda)$ [56]
7. The Fisher information for the Poisson distribution is given by $I(\lambda) = \mathbb{E} \left[\left(\frac{X}{\lambda} - 1 \right)^2 \right] = \mathbb{E} \left[\left(\frac{X - \lambda}{\lambda} \right)^2 \right] = \frac{1}{\lambda}$. [56]
8. Thus, the asymptotic variance of the MLE is given by $I^{-1}(\lambda) = \lambda$ and the standard error of \bar{X} is equal to $\sqrt{\frac{\lambda}{n}}$. [56]

8.5.3. Sums of Random Variables

1. Let U, V , and W be three independent random variables and define $X = W + U$ and $Y = W + V$. When $U \sim \mathcal{P}(\lambda_U)$, $V \sim \mathcal{P}(\lambda_V)$, and $W \sim \mathcal{P}(\lambda_W)$, it can be shown that X is Poisson distributed with parameter $\lambda_W + \lambda_U$ and Y is Poisson distributed with parameter $\lambda_W + \lambda_V$. [56]
2. PDF: $f_{XY}(x, y) = \exp(-[\lambda_U + \lambda_V + \lambda_W]) \frac{\lambda_U^x \lambda_V^y}{x! y!} \sum_{k=0}^{\min(x,y)} \frac{x! y!}{(x-k)! (y-k)! k!} \left(\frac{\lambda_W}{\lambda_U \lambda_V} \right)^k$ [56]

CHAPTER 9

CONTINUOUS DISTRIBUTIONS

9.1. Normal Distribution/ Gaussian Distribution ($N(\mu, \sigma^2)$, $N(x|\mu, \sigma^2)$)

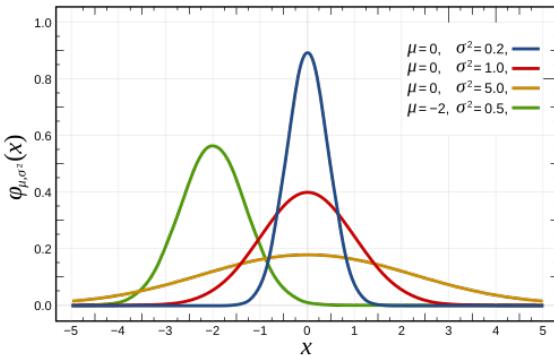


Fig. 9.1: Normal Distribution: PDF [34]

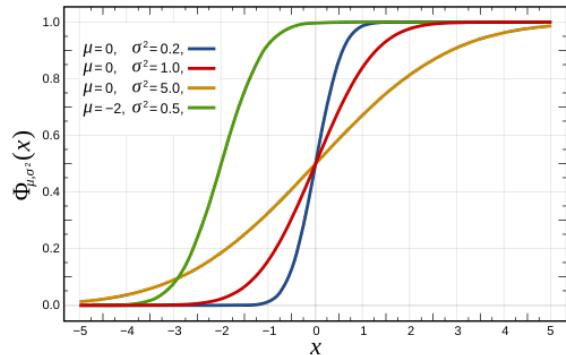


Fig. 9.2: Normal Distribution: CDF [34]

1. Denoted by: $\mathcal{N}(\mu, \sigma^2)$

9.1.1. PDF ($f_{\mu,\sigma}(x)$ or $f(x|\mu, \sigma)$)

1.
$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad [56]$$

$\mu \in \mathbb{R}$: indicates the mean value of the population of the variable of interest [56]

$\sigma \in \mathbb{R}$: indicates the standard deviation ($\sigma^2 > 0$) [56]

2.
$$f_{\mu,\sigma}(x) = \frac{1}{\sigma} \phi\left(\frac{x-\mu}{\sigma}\right) \quad [56]$$

3. it can be used to approximate other PDFs when the sample size or the size of the data is getting large. [56]

4. This has the advantage that important features of the normal density function can be transferred to other densities when the approximation is quite close. [56]

5. The shape of the normal PDF is equal to the famous “bell-shape” curve [56]

6. areas under the curve:

(a) 99.73% of all the population values fall within the interval $[\mu - 3\sigma, \mu + 3\sigma]$ [56]

(b) 95.45% of all the population values fall within the interval $[\mu - 2\sigma, \mu + 2\sigma]$ or [56]

$$\int_{\mu-2\sigma}^{\mu+2\sigma} \phi\left(\frac{x-\mu}{\sigma}\right) dx = \int_{-2}^2 \phi(x) dx = 0.9545 \quad [56]$$

(c) 95% of the values fall within $[\mu - 1.96\sigma, \mu + 1.96\sigma]$ [56]

7. describes both positive and negative values [56]

8. a random measurement error that could be described by a normal PDF is more likely to be closer to zero than to be further away from zero (due to the bell shape of the density). [56]

9.1.2. Maximum Likelihood

1. If our data set \mathbf{x} is i.i.d., then we can therefore write the probability of the data set, given μ and σ^2 , in the form [41]

$$P(\mathbf{x}|\mu, \sigma^2) = \prod_{n=1}^N \mathcal{N}(x_n|\mu, \sigma^2) \quad [41]$$

2. Log likelihood function: [41]

$$\ln(p(\mathbf{x}|\mu, \sigma^2)) = -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 - \frac{N}{2} \ln(\sigma^2) - \frac{N}{2} \ln(2\pi) \quad [41]$$

3. Maximizing $\ln(p(\mathbf{x}|\mu, \sigma^2))$ with respect to μ , we obtain the maximum likelihood solution given by $\mu_{ML} = \frac{1}{N} \sum_{n=1}^N x_n$ which is the sample mean, i.e., the mean of the observed values $\{x_n\}$. [41]

4. Maximizing $\ln(p(\mathbf{x}|\mu, \sigma^2))$ with respect to σ^2 , we obtain the maximum likelihood solution for the variance in the form $\sigma_{ML}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{ML})^2$ which is the sample variance measured with respect to the sample mean μ_{ML} . [41]

$$5. \mathbb{E}[\mu_{ML}] = \mu \quad \mathbb{E}[\sigma_{ML}^2] = \left(\frac{N-1}{N}\right) \sigma^2 \quad [41]$$

9.1.3. Normally Distributed Populations

1. we assume that the random variables X_1, X_2, \dots, X_n are i.i.d. normally distributed, $X_i \sim \mathcal{N}(\mu, \sigma^2)$ [56]

2. Properties:

(a) Property 1: [56]

- i. The sum of the random variables $\sum_{i=1}^n X_i$ is again **normally distributed**, but now with mean $n\mu$ and variance $n\sigma^2$. [56]
- ii. Thus this implies that the sample average \bar{X} has a normal distribution with mean μ and variance σ^2/n and thus has $\frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$, a standard normal distribution function. [56]

(b) Property 2:

- i. The sum of the squared standardized random variables $\frac{1}{\sigma^2} \sum_{i=1}^n (X_i - \mu)^2$ is known to be **chi-square distributed** with n degrees of freedom. [56]
- ii. The sum $\frac{1}{\sigma^2} \sum_{i=1}^n (X_i - \bar{X})^2$ is **chi-square distributed** with $n - 1$ degrees of freedom. [56]

(c) Property 3:

- i. Let Z be standard normally distributed, $Z \sim \mathcal{N}(0, 1)$, let V_n^2 be chi-square distributed with n degrees of freedom, $V_n^2 \sim \chi_n^2$, and assume that Z and V_n^2 are independent. [56]
- ii. The distribution function of the ratio of this standard normal random variable and the square root of a chi-square $\frac{Z}{V_n/\sqrt{n}}$ has a so-called **Student t-distribution** with n degrees of freedom. [56]

9.1.3.1. Confidence Intervals for Normal Populations

1. $Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$ is standard normal distributed (using property 1) [56]

2. $V_{n-1}^2 = \frac{(n-1)S^2}{\sigma^2} \sim \chi_{n-1}^2$ (using property 2) [56]
3. $\frac{\bar{X} - \mu}{S/\sqrt{n}} = \frac{(\bar{X} - \mu)/(\sigma/\sqrt{n})}{\sqrt{(n-1)S^2/\sigma^2/\sqrt{n-1}}} = \frac{Z}{V_{n-1}/\sqrt{n-1}}$ [56]
4. $\frac{\bar{X} - \mu}{S/\sqrt{n}}$ has a Student t-distribution with $n - 1$ degrees of freedom (using property 3) [56]
5. If $x_p(f_t)$ is the p -th quantile of the Student t-distribution with $n - 1$ degrees of freedom, the $1 - 2p$ confidence interval for μ (with $p < 0.5$) is now $\left(\bar{X} - x_{1-p}(f_t) \frac{S}{\sqrt{n}}, \bar{X} + x_{1-p}(f_t) \frac{S}{\sqrt{n}} \right)$ [56]
6. Note that the only restriction on the sample size n , is that it should be larger than 2, otherwise we cannot calculate a sample variance. [56]
7. When the sample size is increasing, the quantile of the Student t-distribution with $n - 1$ degrees of freedom converges to the quantile of the standard normal distribution. The quantiles of the t-distribution are already quite close to the same quantiles of the normal distribution when sample sizes are larger than 100. [56]
8. The asymptotic $1 - 2p$ confidence interval for the variance σ^2 is given by $(S^2 - z_{1-p}\hat{\tau}_n, S^2 - z_{1-p}\hat{\tau}_n)$, where $\hat{\tau}_n = S^2 \sqrt{\frac{b_2 + 2}{n}}$ and b_2 is the sample excess kurtosis. [56]
9. An alternative confidence interval for the variance σ^2 can be created under the assumption that X_1, X_2, \dots, X_n are i.i.d. normally distributed, $X_i \sim \mathcal{N}(\mu, \sigma^2)$. Let $x_p(f_{\chi^2})$ be the p -th quantile of the chi-square distribution with $n - 1$ degrees of freedom.
 - (a) Then $P(V_{n-1}^2 \leq x_p(f_{\chi^2})) = p$. [56]
 - (b) Then $P(\sigma^2 \geq (n-1)S^2/x_p(f_{\chi^2})) = p$ Using $\left(V_{n-1}^2 = \frac{(n-1)S^2}{\sigma^2} \sim \chi_{n-1}^2 \right)$ [56]
 - (c) $P(V_{n-1}^2 > x_{1-p}(f_{\chi^2})) = p \implies P\left(\sigma^2 < \frac{(n-1)S^2}{x_{1-p}(f_{\chi^2})}\right) = p$ [56]
 - (d) Thus a $1 - 2p$ confidence interval for the variance σ^2 is now given by $\left[\frac{(n-1)S^2}{x_{1-p}(f_{\chi^2})}, \frac{(n-1)S^2}{x_p(f_{\chi^2})} \right]$ [56]
10. Confidence intervals for the population variance σ^2 immediately result in confidence intervals on the population standard deviation σ by just taking the square root. [56]

9.1.4. Distribution of the Sample Statistic (T_n)

9.1.4.1. Distribution of the Sample Average

1. let's consider sample average $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ that tries to estimate the population mean $\mu(f)$. [56]
2. From central limit theorem, $X \sim \mathcal{N}\left(\mu(f), \frac{\sigma^2(f)}{n}\right)$ is approximately normally distributed. [56]

3. Asymptotic Confidence Intervals:

- (a) if $CI = 95\%$, and $CI = 1 - 2p$, $p = 2.5\%$, $1 - p = 97.5\%$ quantile of the standard normal distribution function is equal to $z_{1-p} = z_{0.975} = 1.96$ [56]

- (b) Applying the 95% asymptotic confidence interval for $\mu(f)$ using the estimator \bar{X} , results in [56]

$$\left[\bar{X} - \frac{1.96 \sigma(f)}{\sqrt{n}}, \bar{X} + \frac{1.96 \sigma(f)}{\sqrt{n}} \right] \quad [56]$$

(c) Since the standard deviation $\sigma(f)$ is unknown, we may replace $\sigma(f)$ by an estimator. The most commonly used estimator is to use the sample standard deviation $S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2}$. [56]

(d) The 95% confidence interval on $\mu(f)$ that can be calculated from the data is then equal to [56]

$$\left[\bar{X} - \frac{1.96 S}{\sqrt{n}}, \bar{X} + \frac{1.96 S}{\sqrt{n}} \right] \quad [56]$$

9.1.5. Methods of Moments Estimation (MME)

1. The normal density has two parameters $\theta_1 = \mu$ and $\theta_2 = \sigma^2$. [56]

9.1.6. Sums of Random Variables

1. Let U , V , and W be three independent random variables and define $X = W + U$ and $Y = W + V$. When $U \sim \mathcal{N}(\mu_U, \sigma_U^2)$, $V \sim \mathcal{N}(\mu_V, \sigma_V^2)$, and $W \sim \mathcal{N}(\mu_W, \sigma_W^2)$, it can be shown that X and Y are bivariate normally distributed with [56]

$(a) \mu_X = \mu_W + \mu_U$ [56]	$(d) \sigma_Y^2 = \sigma_W^2 + \sigma_U^2$ [56]
$(b) \mu_Y = \mu_W + \mu_V$ [56]	$(e) \rho = \frac{\sigma_W^2}{\sqrt{(\sigma_W^2 + \sigma_U^2)(\sigma_W^2 + \sigma_V^2)}}$ [56]
$(c) \sigma_X^2 = \sigma_W^2 + \sigma_U^2$ [56]	

9.1.7. The t-Test for a Single Sample ($H_0 : \mu(f) \leq \mu_0$)

1. Lets assume random variables Y_1, Y_2, \dots, Y_n are i.i.d. normally $\mathcal{N}(\mu_0, \sigma^2)$ distributed, and test statistic $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ has a t-distribution with $n - 1$ degrees of freedom. [56]
2. instead of using the $1 - \alpha$ quantile $z_{1-\alpha}$ of the normal distribution, we may better use the $1 - \alpha$ quantile $x_{1-\alpha}(f_t)$ of the t-distribution for one-sided testing. [56]
3. we would obtain that $P\left(\frac{\bar{Y} - \mu_0}{S/\sqrt{n}} > x_{1-\alpha}(f_t)\right) = \alpha$ for any sample size n . [56]
4. The test statistic $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ is called the **one-sample t-test** and we would reject the one-sided null hypothesis $H_0 : \mu(f) \leq \mu_0$ in favor of $H_a : \mu(f) > \mu_0$ when the observed value $t_n > x_{1-\alpha}(f_t)$ and do not reject the null hypothesis when $t_n \leq x_{1-\alpha}(f_t)$. The probability for $H_0 : \mu(f) \leq \mu_0$ against $H_a : \mu(f) > \mu_0$ is equal to $p = 1 - F_t(t_n)$, with F_t the t-distribution function with $n - 1$ degrees of freedom and t_n is calculated from the data. If this p-value is below α , we believe that it is unlikely to obtain the result t_n or larger results under the null hypothesis $H_0 : \mu(f) \leq \mu_0$. The p-value would be exactly equal to α if t_n equals the critical value $x_{1-\alpha}(f_t)$. Indeed, $\alpha = 1 - F_t(x_{1-\alpha}(f_t)) = P(T_n > x_{1-\alpha}(f_t))$. [56]
5. we would reject the one-sided null hypothesis $H_0 : \mu(f) \geq \mu_0$ in favor of $H_0 : \mu(f) < \mu_0$ when $t_n < -x_{1-\alpha}(f_t)$ and do not reject the null hypothesis when $t_n \geq -x_{1-\alpha}(f_t)$. For the null hypothesis $H_0 : \mu(f) \geq \mu_0$ against $H_a : \mu(f) < \mu_0$ the p-value is calculated as $p = F_t(t_n)$. [56]
6. we would reject the two-sided null hypothesis $H_0 : \mu(f) = \mu_0$ in favor of $H_a : \mu(f) \neq \mu_0$ when $|t_n| > x_{1-\alpha/2}(f_t)$ and do not reject the null hypothesis when $|t_n| \leq x_{1-\alpha/2}(f_t)$. For the null hypothesis $H_0 : \mu(f) = \mu_0$ against $H_a : \mu(f) \neq \mu_0$, the p-value is calculated as $p = 2[1 - F_t(|t_n|)]$. [56]
7. There is no other test statistic with the same type 1 error α that would reject the null hypothesis quicker than the t-test when the alternative hypothesis is true, i.e., it has the highest power compared to any other test statistic. [56]

9.1.8. The t-Test for Two Independent Samples ($H_0 : \mu_1 = \mu_2$)

1. Goal: to compare the means of two independent samples with each other. [56]

2. Lets assume we have one sample from population $h = 1$ with sample size n_1 and one sample from population $h = 2$ with sample size n_2 . The random variables are denoted by $Y_{h,1}, Y_{h,2}, \dots, Y_{h,n_h}$ for population h .

3. Let $Y_{h,1}, Y_{h,2}, \dots, Y_{h,n_h}$ i.i.d. $\mathcal{N}(\mu_h, \sigma_h^2)$ and we are interested in a testing hypothesis regarding the difference $\mu_1 - \mu_2$ (i.e., the difference in population means), a natural estimator for this difference is

$$\bar{Y}_1 - \bar{Y}_2, \text{ and the standard error of this estimator is } \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}. \quad [56]$$

4. This standard error can be estimated by substituting the sample variance S_h^2 for σ_h^2 , $h = 1, 2$. [56]

5. If the standard deviations of the two populations are equal ($\sigma = \sigma_1 = \sigma_2$), the standard error $\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$ becomes equal to $\sigma \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$. [56]

(a) In the case of equal variances, both sample variances S_1^2 and S_2^2 provide information on the variance σ^2 . [56]

(b) The variance σ^2 can now be estimated by the **pooled sample** variance S_p^2 given by [56]

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2} \quad [56]$$

(c) S_p^2 is a weighted average of the sample variances where the weights are based on the degrees of freedom. [56]

(d) The random variable $\frac{\bar{Y}_1 - \bar{Y}_2}{S_p \sqrt{1/n_1 + 1/n_2}}$ has a t-distribution function with $n_1 + n_2 - 2$ degrees of freedom. It can be used to test a one-sided or two-sided null-hypothesis on the mean difference $\mu_1 - \mu_2$. [56]

6. When the two variances are unequal, the statistic $\frac{\bar{Y}_1 - \bar{Y}_2}{\sqrt{S_1^2/n_1 + S_2^2/n_2}}$ does not follow a t-distribution and the distribution function of the statistic is not free from the ratio $\frac{\sigma_1}{\sigma_2}$. [56]

(a) This does not mean that we should always use asymptotic theory for normally distributed data, since the t-distribution still provides a better approximation when the degrees of freedom are estimated from the data. [56]

(b) This approximation is often referred to as the **Satterthwaite or Welch approximation**. [56]

7. Null hypothesis (no difference in means): $H_0 : \mu_1 = \mu_2$ OR $H_0 : \mu_1 - \mu_2 = 0$ [7]

8. Alternative hypotheses (depending on test type): [7]

(a) Two-sided test (most common): $H_a : \mu_1 \neq \mu_2$ [7]

(b) One-sided tests (if direction is specified): $H_a : \mu_1 > \mu_2$ OR $H_a : \mu_1 < \mu_2$ [7]

9. How to check if we have sufficient evidence: [7]

(a) t-test statistic: $t = \frac{\bar{Y}_1 - \bar{Y}_2}{SE(\bar{Y}_1 - \bar{Y}_2)}$ [7]

where $SE(\bar{Y}_1 - \bar{Y}_2)$ depends on whether we assume equal variances (pooled S_p^2) or unequal variances (Welch's test). [7]

10. Decision rule: [7]

- (a) We test against a significance level (usually $\alpha = 0.05$) [7]
- (b) Compute the p-value from the t-distribution (with appropriate degrees of freedom). [7]
- (c) If p-value $\leq \alpha$: Reject $H_0 \rightarrow$ Evidence suggests a difference in means. [7]
- (d) If p-value $> \alpha$: Fail to reject $H_0 \rightarrow$ Not enough evidence to conclude a difference. [7]

9.1.9. The t-Test for Two Dependent Samples ($H_0 : \mu_D = \mu_1 - \mu_2 \leq 0$)

1. Used to compare the means of two dependent samples: e.g., the means of two related groups. [56]
2. This occurs frequently when we want to compare a property that we measure regarding people (e.g., their happiness) at one point in time with that same property at a later point in time. [56]
3. In this case, contrary to the independent sample case discussed above, the **data are paired** [56]
4. The null hypothesis is still formulated on the difference in means $\mu_1 - \mu_2$ for the two samples, like we did for the two independent samples, but now we must take into account that the data may be dependent. [56]
5. We can consider hypothesis testing regarding the difference in means by calculating difference scores. Thus, we can consider the difference scores $D_1 = Y_{1,1} - Y_{2,1}, D_2 = Y_{1,2} - Y_{2,2}, \dots, D_n = Y_{1,n} - Y_{2,n}$. By considering the difference score we have brought the two samples back to one sample of difference scores and addressed the dependency between the two samples. [56]
6. the test statistic $T_n = \frac{\bar{D}}{\hat{SE}(\bar{D})}$, where \bar{D} is the average of the difference scores and $\hat{SE}(\bar{D}) = \frac{s_D}{\sqrt{n}}$ with s_D the sample standard deviation of the difference scores. The test statistic T_n follows a t-distribution with $n - 1$ degrees of freedom (when the difference scores are normally distributed). [56]
7. This test statistic can be used to test the null hypotheses $H_0 : \mu_D = \mu_1 - \mu_2 \leq 0, H_0 : \mu_D = \mu_1 - \mu_2 \geq 0$ OR $H_0 : \mu_D = \mu_1 - \mu_2 = 0$. [56]

9.1.10. The F-Test for Equal Variances ($H_0 : \sigma_1 = \sigma_2$)

1. Under the two-sided null hypothesis $H_0 : \sigma_1 = \sigma_2$, the ratio $\frac{S_1^2}{S_2^2}$ has an F-distribution with $n_1 - 1$ and $n_2 - 1$ degrees of freedom and the ratio $\frac{S_2^2}{S_1^2}$ has an F-distribution function with $n_2 - 1$ and $n_1 - 1$ degrees of freedom. [56]
2. If $\frac{S_1^2}{S_2^2}$ or $\frac{S_2^2}{S_1^2}$ is large, then the null hypothesis $H_0 : \sigma_1 = \sigma_2$ is rejected. We may choose just one of the two ratios, since there is symmetry. If one ratio is large, the other ratio must be small, or the other way around. Thus if we choose one of the ratios, the null hypothesis is rejected when this ratio is large or when this ratio is small. [56]
3. The critical value can be determined by using the function $qf(1 - \alpha/2, d_1, d_2)$ for large ratios or $qf(\alpha/2, d_1, d_2)$ for small ratios, with d_1 the degrees of freedom for the sample variance in the numerator and d_2 the degrees of freedom of the sample variance in the denominator. Where $qf(p, df_1, df_2)$ gives the quantile of the F-distribution. it tells you the critical value such that $P(F \leq qf(p, df_1, df_2)) = p$ [56]
4. The choice of degrees of freedom depends on which ratio $\frac{S_1^2}{S_2^2}$ or $\frac{S_2^2}{S_1^2}$ is selected, but one of them is equal to $n_1 - 1$ and the other to $n_2 - 1$. Note that we must use $\alpha/2$ since we are using the two-sided test. [56]

5. an observed F value larger than 3 or 4 (or smaller than 1/3 or 1/4) typically indicates that the variances between the two populations are likely to be unequal (even if the data in the two samples are not from a normal distribution). [56]

9.1.11. Tests for Normality

1. Many statistical techniques “require” that the data come from a normal distribution (e.g., the t-test). [56]
2. When the (sample) data deviate greatly from the normal distribution the properties of test statistics that rely on normality assumptions will not hold. This means that their p-values will need to be interpreted with more caution. [56]
3. Under normality ties cannot occur. [56]
4. Assume that we have selected a sample of data Y_1, Y_2, \dots, Y_n . [56]

9.1.11.1. Graphical approach (g-g Plot)

1. A common visualization method for this purpose is the quantile-quantile or q-q plot. [56]
2. In this case, the ordered observations $Y_{(1)}, Y_{(2)}, \dots, Y_{(n)}$ are plotted against the quantiles of the normal distribution. [56]
3. The quantile q_k that belongs to $Y_{(k)}$ is given by $q_k = \Phi^{-1} \left(\frac{k - 0.375}{n + 0.125} \right)$, with Φ the standard normal distribution function. [56]
4. This is close to the quantile $\Phi^{-1} \left(\frac{k}{n} \right)$, but the constants 0.375 and 0.125 are used to make the graphical approach somewhat less biased. [56]
5. If the data Y_1, Y_2, \dots, Y_n come from a normal distribution, the graph $(q_k, Y_{(k)})$ would form approximately a **straight line**. [56]
6. The intercept of this line is an estimate of the population mean and the slope of the line is then an estimate of the population standard deviation. [56]

9.1.11.2. Shapiro–Wilk test

1. The test statistic is of the form $W = \sum_{k=1}^n \frac{a_{n,k} Y_{(k)}}{S \sqrt{n-1}}$, with $a_{n,k}$ constants that depend on the sample size and on the normal quantiles and S the sample standard deviation. [56]
2. Large values of W indicate that it is unlikely that the data was collected from a normal distribution, but intuition on this value is missing. Instead we may use the p-value that is calculated with the statistic. [56]
3. **Disadvantage:** It is sensitive to **ties**. When there are many ties, the Shapiro–Wilk test may show that the data is not normal. If these ties are caused by rounding issues and the sampled data actually come from a normal distribution, the Shapiro–Wilk test may incorrectly reject normality. [56]
4. **Hypotheses:**
 - (a) Null hypothesis (H_0): The sample is drawn from a normally distributed population.
 - (b) Alternative hypothesis (H_a): The sample is not drawn from a normal distribution.
5. **Decision Rule:** [7]
 - (a) The Shapiro–Wilk test produces: [7]
 - i. A test statistic W [7]

- ii. A p-value [7]
- (b) $p \leq \alpha$ (e.g., 0.05): Reject $H_0 \rightarrow$ Data not normal (significant deviation from normality) [7]
- (c) $p > \alpha$: Fail to reject $H_0 \rightarrow$ No evidence against normality (data likely normal) [7]

9.1.12. Grubbs Test for Outliers/ extreme studentized deviate test/ maximum normed residual test

1. The Grubbs test is sometimes called the **extreme studentized deviate test** or the **maximum normed residual test**. [56]
2. The null hypothesis is that no observation is an outlier, while the alternative hypothesis is that there is one observation that is an outlier. [56]
3. the Grubbs test is only searching for **one outlier** observation. [56]
4. The calculation of the Grubbs test is quite easy, since it is equal to $G = \max \{|Z_1|, |Z_2|, \dots, |Z_n|\}$, with $Z_i = \frac{Y_i - \bar{Y}}{S}$ the standardized observations (with \bar{Y} the average and S the standard deviation). [56]
5. The probability that G is larger than some critical value c_α is equal to [56]

$$P(G > c_\alpha) \approx 1 - \prod_{i=1}^n P(|Z_i| \leq c_\alpha) \quad [56]$$

6. The larger the sample size (n), the better the approximation. [56]
7. Critical Value: $c_\alpha = \frac{n-1}{\sqrt{n}} \sqrt{\frac{t_{\alpha/(2n), n-2}^2}{n-2 + t_{\alpha/(2n), n-2}^2}}$ [7]
where $t_{\alpha/(2n), n-2}^2$ is the critical value of the t-distribution with $n-2$ degrees of freedom at the $\alpha/(2n)$ level. [7]
8. For $\alpha = 0.05$ the critical value becomes approximately equal to $c_\alpha = 4$ and for $\alpha = 0.01$ the critical value becomes approximately $c_\alpha = 4.5$. It is not uncommon to qualify an observation as an outlier when the absolute standardized value is larger than 4. Some software packages would even signal absolute standardized values larger than 3 as potential outliers. [56]
9. **Interpretation:** [7]
 - (a) If $G > c_\alpha$: Reject H_0 (no outliers) \rightarrow The data point corresponding to G is an outlier. [7]
 - (b) If $G \leq c_\alpha$: Fail to reject $H_0 \rightarrow$ No significant outlier. [7]

9.1.13. Tukey's Method for Outliers (Using IQR)

1. For the normal distribution function $\Phi\left(\frac{x-\mu}{\sigma}\right)$, with Φ the standard normal distribution function, these criteria become $\mu + \Phi^{-1}(0.25)\sigma - 1.5(\Phi^{-1}(0.75) - \Phi^{-1}(0.25))\sigma = \mu - 2.69796 \cdot \sigma$ and $\mu + 2.69796 \cdot \sigma$ for the lower and upper tail, respectively. [56]
2. Thus the percentage of units from the population that fall outside these criteria is approximately only 0.70%, which does not depend on the parameters μ and σ . [56]
3. Thus under the assumption of normality, an outlier observation has 0.70% probability to occur in the sample. [56]

9.1.14. Estimating the Parameters using Bayesian Methods

1. Let X_1, \dots, X_n are i.i.d. $\mathcal{N}(\mu, \sigma^2)$ [56]

9.1.14.1. Bayesian Analysis for Normal Populations Based on Single Observation

1. Let $X_1 \sim \mathcal{N}(\mu, \sigma^2)$, we only observe a single realization x , and we assume σ^2 known; hence, we only derive the posterior distribution for μ . [56]

2. **sampling model:** $\ell(\theta) = P(x|\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ which is simply the Normal PDF. [56]

3. Since $\mu \in (-\infty, \infty)$ a natural choice for a prior has support on the full real number line. [56]

4. **prior** for μ : $f(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}$ where we use μ_0 and σ_0^2 to denote the parameters of this

prior distribution (not to be confused with μ and σ^2 as they occur in the likelihood). [56]

5. **posterior:** [56]

$$\begin{aligned}
 f(\mu|x) &\propto P(x|\mu)f(\mu) \\
 &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \times \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right) \\
 &= \frac{1}{2\pi\sqrt{\sigma^2\sigma_0^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} - \frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right) \\
 &= c \times \exp\left(-\frac{\mu^2\sigma^2 - 2\mu\mu_0\sigma^2 + \mu_0^2\sigma^2 + \sigma_0^2x^2 - 2\mu\sigma_0^2x + \mu^2\sigma_0^2}{2\sigma_0^2\sigma^2}\right) \\
 &= c \times \exp\left(-\frac{\mu^2(\sigma^2 + \sigma_0^2) - 2\mu(\mu_0\sigma^2 + \sigma_0^2x) + (\mu_0^2\sigma^2 + \sigma_0^2x^2)}{2\sigma_0^2\sigma^2}\right) \\
 &= c \times \exp\left(-\frac{\mu^2(\sigma^2 + \sigma_0^2) - 2\mu(\mu_0\sigma^2 + \sigma_0^2x)}{2\sigma_0^2\sigma^2}\right) \times \exp\left(-\frac{(\mu_0^2\sigma^2 + \sigma_0^2x^2)}{2\sigma_0^2\sigma^2}\right) \\
 &= c \times \exp\left(-\frac{\mu^2 - 2\mu \frac{\mu_0\sigma^2 + \sigma_0^2x}{\sigma^2 + \sigma_0^2} - \left(\frac{\mu_0\sigma^2 + \sigma_0^2x}{\sigma^2 + \sigma_0^2}\right)^2 + \left(\frac{\mu_0\sigma^2 + \sigma_0^2x}{\sigma^2 + \sigma_0^2}\right)^2}{\frac{2\sigma_0^2\sigma^2}{\sigma^2 + \sigma_0^2}}\right) \times \exp\left(-\frac{(\mu_0^2\sigma^2 + \sigma_0^2x^2)}{2\sigma_0^2\sigma^2}\right) \\
 &\propto \exp\left(-\frac{\left(\mu - \frac{\mu_0\sigma^2 + \sigma_0^2x}{\sigma^2 + \sigma_0^2}\right)^2}{2\frac{\sigma^2\sigma_0^2}{\sigma^2 + \sigma_0^2}}\right)
 \end{aligned} \tag{56}$$

where c is a the Normalizing constant. [56]

6. Let:

$$(a) \sigma_1^2 = \frac{\sigma^2\sigma_0^2}{\sigma^2 + \sigma_0^2} = \frac{1}{\sigma^{-2} + \sigma_0^{-2}} \Rightarrow \sigma_1^{-2} = \sigma^{-2} + \sigma_0^{-2} \tag{56}$$

$$(b) \mu_1 = \frac{\mu_0\sigma^2 + x\sigma_0^2}{\sigma^2 + \sigma_0^2} = \frac{\mu_0\sigma^{-2} + x\sigma_0^{-2}}{\sigma^{-2} + \sigma_0^{-2}} = \sigma_1^2(\mu_0\sigma_0^{-2} + x\sigma^{-2}) \Rightarrow \mu_1\sigma_1^{-2} = \mu_0\sigma_0^{-2} + x\sigma^{-2} \tag{56}$$

$$7. f(\mu|x) \propto e^{\frac{-(\mu-\mu_1)^2}{2\sigma_1^2}}, \text{ Normalizing constant : } \frac{1}{\sqrt{2\pi\sigma_1^2}}, \Rightarrow f(\mu|x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{\frac{-(\mu-\mu_1)^2}{2\sigma_1^2}} \tag{56}$$

8. if σ^2 is assumed known, and we use a Normal prior for μ , we find that the posterior for μ is itself a Normal: $\mu|X_1=x \sim \mathcal{N}(\mu_1, \sigma_1^2)$. [56]

9. posterior mean μ_1 is effectively a weighted average of the prior mean μ and the observation x . [56]

9.1.14.2. Bayesian Analysis for Normal Populations Based on Multiple Observations

1. Let realizations x_1, \dots, x_n , iid $X_i \sim \mathcal{N}(\mu, \sigma^2)$; we assume σ^2 is known [56]

$$2. \text{ sampling model: } \ell(\theta) = P(x_1, \dots, x_n | \mu) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) \quad [56]$$

$$3. \text{ prior for } \mu: f(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(\mu - \mu_0)^2}{2\sigma_0^2}} \quad \text{where we use } \mu_0 \text{ and } \sigma_0^2 \text{ to denote the parameters of this prior distribution (not to be confused with } \mu \text{ and } \sigma^2 \text{ as they occur in the likelihood).} \quad [56]$$

$$4. \text{ Posterior: } f(\mu | x_1, \dots, x_n) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(\mu - \mu_1)^2}{2\sigma_1^2}\right) \quad [56]$$

$$(a) \sigma_1^2 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2/n} \right)^{-2} \quad [56]$$

$$(b) \mu_1 = \sigma_1^2 \left(\frac{\mu_0}{\sigma_0^2} + \frac{\bar{x}}{\sigma^2/n} \right) \quad [56]$$

Posterior mean μ_1 is a weighted average of the prior mean μ_0 and the mean of the observations

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i. \quad [56]$$

(c) The variance of the posterior is affected by the number of observations; the term $\frac{\sigma^2}{n}$ clearly shrinks as the number of observations increases leading to a decrease in the variance of the posterior. If $n \rightarrow \infty$, then μ_1 will converge to the sample mean, and σ_1^2 will converge to zero. [56]

9.1.14.3. Bayesian Analysis for Normal Populations with Unknown Mean and Variance

1. **sampling model:** $\ell(\theta) = P(x_1, \dots, x_n | \mu, \sigma^2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) \quad [56]$

2. **prior:** multivariate PDF $f(\mu, \sigma^2)$ with the correct support ($\sigma^2 \in (0, \infty]$). Often used (conjugate) prior is the Normal-inverse- χ^2 distribution [56]

(a) Why a multivariate PDF $f(\mu, \sigma^2)$? [7]

i. Because now both parameters are unknown — the mean μ and the variance σ^2 [7]

ii. So, instead of having a prior for just one parameter (like $P(\mu)$ when σ^2 was known), we now need a joint prior distribution that captures our beliefs about both parameters together: $f(\mu, \sigma^2) = P(\mu, \sigma^2)$ [7]

iii. This is a multivariate probability density function because it describes the probability density over two variables (μ and σ^2). [7]

iv. $P(\mu, \sigma^2) = P(\mu | \sigma^2) P(\sigma^2)$ [7]

(b) What is a (conjugate) prior? [7]

i. A prior is your belief about the parameter(s) before seeing data. [7]

ii. A conjugate prior is a special kind of prior such that the posterior distribution (after seeing data) is of the same functional form as the prior. [7]

iii. **Example:** If the likelihood is Normal, and you choose a Normal prior, then the posterior is also Normal — that's a conjugate prior relationship. [7]

iv. **Here:** The likelihood for data is Normal, and if you choose the Normal-inverse- χ^2 prior, then after applying Bayes' theorem, the posterior also remains a Normal-inverse- χ^2 distribution (same family). [7]

(c) If it's Normal, why "Normal-inverse- χ^2 " distribution? [7]

i. The conditional prior for μ given σ^2 is Normal: $\mu|\sigma^2 \sim \mathcal{N}(\mu_0, \sigma^2/\kappa_0)$ [7]

ii. The marginal prior for σ^2 is inverse- χ^2 (or equivalently inverse-Gamma): $\sigma^2 \sim \text{Inv-}\chi^2(v_0, \sigma_0^2)$ [7]

iii. Combining these two gives the joint prior: $f(\mu, \sigma^2) = \mathcal{N}(\mu|\mu_0, \sigma^2/\kappa_0) \times \text{Inv-}\chi^2(\sigma^2|v_0, \sigma_0^2)$ [7]

$$f(\mu, \sigma) = \mathcal{N}\mathcal{I}\chi^2(\mu_0, \kappa_0, v_0, \sigma_0^2) = P(\mu, \sigma^2) = P(\mu|\sigma^2)P(\sigma^2)$$

$$\begin{aligned} \text{(d) PDF: } &= \mathcal{N}(\mu_0, \sigma^2/\kappa_0) \times \chi^{-2}(v_0, \sigma_0^2) \\ &= \frac{1}{Z(\mu_0, \kappa_0, v_0, \sigma_0^2)} (\sigma^2)^{-(v_0/2+1)/2} \exp\left(-\frac{1}{2\sigma^2}[v_0\sigma_0^2 + \kappa_0(\mu_0 - \mu)^2]\right) \end{aligned} \quad [56]$$

where

$$Z(\mu_0, \kappa_0, v_0, \sigma_0^2) = \frac{\sqrt{2\pi}}{\sqrt{\kappa_0}} \Gamma\left(\frac{v_0}{2}\right) \left(\frac{2}{v_0\sigma_0^2}\right)^{v_0/2} \quad [56]$$

The above shows that our prior for σ^2 is a scaled inverse- χ^2 distribution and the prior for μ conditional on σ is a Normal distribution; both of these are known and well-understood univariate distribution functions. [56]

3. **posterior:** $f(\mu, \sigma^2|x_1, \dots, x_n) = \mathcal{N}\mathcal{I}\chi^2(\mu_n, \kappa_n, v_n, \sigma_n^2)$ [56]

$$(a) \mu_n = \frac{\kappa_0\mu_0 + n\bar{x}}{\kappa_n} \quad [56]$$

$$(b) \kappa_n = \kappa_0 + n \quad [56]$$

$$(c) v_n = v_0 + n \quad [56]$$

$$(d) \sigma_n^2 = \frac{1}{v_n} \left(v_0\sigma_0^2 + \sum_{i=1}^n (x_i - \bar{x})^2 + \frac{n\kappa_0}{\kappa_0 + n}(\mu_0 - \bar{x})^2 \right) \quad [56]$$

4. **marginal posterior PDF**

$$(a) f(\sigma^2|x_1, \dots, x_n) = \chi^{-2}(v_n, \sigma_n^2) \quad [56] \quad (b) f(\mu|x_1, \dots, x_n) = t\left(\mu_n, \frac{\sigma_n^2}{\kappa_n}\right) \quad [56]$$

9.1.15. Gaussian as Exponential Family

1. Consider the univariate Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. Let $\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$. [49]

2. $P(x|\theta) \propto \exp(\theta_1 x + \theta_2 x^2)$ [49]

3. Setting $\theta = \begin{bmatrix} \mu \\ \sigma^2 \end{bmatrix}^\top$, $P(x|\theta) \propto \exp\left(\frac{\mu x}{\sigma^2} - \frac{x^2}{2\sigma^2}\right) \propto \exp\left(\frac{1}{2\sigma^2}(x - \mu)^2\right)$ [49]

4. the univariate Gaussian distribution is a member of the exponential family with sufficient statistic $\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$, and natural parameters given by $\theta = \begin{bmatrix} \mu \\ \sigma^2 \end{bmatrix}^\top$ [49]

9.1.16. Summary

1. **Notation:** $\mathcal{N}(\mu, \sigma^2)$ [34]

2. **Parameters:**

(a) $\mu \in \mathbb{R}$ = mean (location) [34]

(b) $\sigma^2 \in \mathbb{R}_{>0}$ = variance (squared scale) [34]

3. **Support/ Rand Var:** $x \in \mathbb{R}$ [34]

4. **PDF:** $\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ [34]

5. **CDF:** $\Phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2} \left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right) \right]$ [34]

6. **Quantile:** $\mu + \sigma\sqrt{2}\text{erf}^{-1}(2p - 1)$ [34]

7. **Mean:** μ [34]

8. **Median:** μ [34]

9. **Mode:** μ [34]

10. **Variance:** σ^2 [34]

11. **Precision:** $\beta = \frac{1}{\sigma^2}$ [41]

12. **Median absolute deviation (MAD):** $\sigma\sqrt{2}\text{erf}^{-1}(1/2)$ [34]

13. **Average absolute deviation (AAD):** $\sigma\sqrt{2/\pi}$ [34]

14. **Skewness:** 0 [34]

15. **Excess kurtosis:** 0 [34]

16. **Entropy:** $\frac{1}{2}\log(2\pi e\sigma^2)$ [34]

17. **Moment-generating function (MGF):** $\exp(\mu t + \sigma^2 t^2/2)$ [34]

18. **Characteristic function (CF):** $\exp(i\mu t - \sigma^2 t^2/2)$ [34]

19. **Fisher information:**

$$(a) \quad \mathcal{I}(\mu, \sigma) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{pmatrix}$$

$$(b) \quad \mathcal{I}(\mu, \sigma^2) = \begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 1/(2\sigma^4) \end{pmatrix}$$

[34]

20. **Kullback–Leibler divergence:** $D_{KL} = \frac{1}{2} \left\{ \left(\frac{\sigma_0}{\sigma_1} \right)^2 + \frac{(\mu_1 - \mu_0)^2}{\sigma_1^2} - 1 + \ln \frac{\sigma_1^2}{\sigma_0^2} \right\}$ [34]

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{\left(q_p\left(\frac{X-\mu}{\sigma}\right)\right)^2}{2}}$$

21. **Expected shortfall:** $\mu + \sigma \frac{\sqrt{2\pi}}{1-p}$ [34]

22. **second order moment:** $\mathbb{E}[x^2] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x^2 dx = \mu^2 + \sigma^2$ [41]

9.2. Multivariate normal distribution ($N(\mu, \Sigma)$)

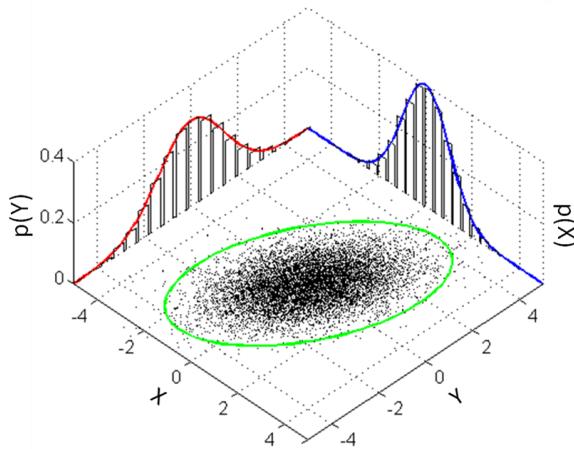


Fig. 9.3: Multivariate normal distribution: PDF [32]

9.2.1. PDF

$$1. \quad \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-D/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \quad [41]$$

$$\mathbf{x}, \boldsymbol{\mu} \in \mathbb{R}^D \quad \boldsymbol{\Sigma} \in \mathbb{R}^{D \times D} \quad |\boldsymbol{\Sigma}| = \det(\boldsymbol{\Sigma}) \in \mathbb{R} \quad [41]$$

2. when X and Y are marginally normally distributed, there are different ways of creating dependencies between X and Y that are different from the bivariate PDF [56]
Even if two random variables X and Y each individually follow a normal distribution (i.e., their marginals are normal), the way in which they are dependent or correlated can vary. That is, their joint distribution doesn't have to be the bivariate normal distribution [7]

9.2.2. Bivariate Normal Distributions

1. PDF: $f(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{z_1^2 - 2\rho z_1 z_2 + z_2^2}{2(1-\rho^2)}\right) \quad [56]$

(a) $z_1 = \frac{x - \mu_X}{\sigma_X}$: standardized normal variable for X [56]

(b) $z_2 = \frac{y - \mu_Y}{\sigma_Y}$: standardized normal variable for Y [56]

(c) ρ : **correlation coefficient** and is contained within the interval $[-1, 1]$. [56]

(d) When $\rho = 0$, the normal random variables X and Y are **independent** [56]

(e) when $\rho \neq 0$, the normal random variables X and Y are **dependent** [56]

$$\text{Cov}[X, Y] = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] = \sigma_X \sigma_Y \mathbb{E}[Z_1 Z_2]$$

$$= \sigma_X \sigma_Y \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} z_1 z_2 \frac{1}{2\pi(1-\rho^2)} \exp\left(-\frac{z_1^2 - 2\rho z_1 z_2 + z_2^2}{2(1-\rho^2)}\right) dz_1 dz_2$$

2. $= \sigma_X \sigma_Y \int_{-\infty}^{\infty} z_2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_2^2}{2}\right) \int_{-\infty}^{\infty} z_1 \frac{1}{\sqrt{2\pi(1-\rho^2)}} \exp\left(-\frac{(z_1 - \rho z_2)^2}{2(1-\rho^2)}\right) dz_1 dz_2 \quad [56]$
 $= \sigma_X \sigma_Y \int_{-\infty}^{\infty} \rho z_2^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_2^2}{2}\right) dz_2 = \rho \sigma_X \sigma_Y$

3. Consider a Gaussian distributed random variable $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For a given matrix \mathbf{A} of appropriate shape, let \mathbf{Y} be a random variable such that $\mathbf{y} = \mathbf{Ax}$ is a transformed version of \mathbf{x} . We can compute the

mean of \mathbf{y} by exploiting that the expectation is a linear operator as follows: [49]

$$\mathbb{E}[\mathbf{y}] = \mathbb{E}[\mathbf{Ax}] = \mathbf{A}\mathbb{E}[\mathbf{x}] = \mathbf{A}\boldsymbol{\mu} \quad [49]$$

the variance of \mathbf{y} : $\mathbb{V}[\mathbf{y}] = \mathbb{V}[\mathbf{Ax}] = \mathbf{A}\mathbb{V}[\mathbf{x}]\mathbf{A}^\top = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top$ [49]

the random variable \mathbf{y} is distributed according to $P(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top)$ [49]

\mathbf{x} is a linear transformation of \mathbf{y} , and we obtain: [49]

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{y}, (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\boldsymbol{\Sigma}\mathbf{A}(\mathbf{A}^\top\mathbf{A})^{-1}) \quad [49]$$

4. The distribution function of any linear combination of X and Y , say $aX + bY$, has a normal distribution function with mean $\mu = a\mu_X + b\mu_Y$ and variance $\sigma^2 = a^2\sigma_X^2 + 2ab\rho\sigma_X\sigma_Y + b^2\sigma_Y^2$. [56]

5. Pearson's Correlation Coefficient: $\rho_P = \text{CORR}(X, Y) = \rho$ [56]

6. Kendall's tau: $\tau_K = \frac{2 \arcsin(\rho)}{\pi}$ [56]

7. Spearman's rho correlation: $\rho_S = \frac{6}{\pi} \arcsin\left(\frac{\rho}{2}\right)$ [56]

8. if $\rho_P = 0$ and the pairs (X_i, Y_i) are i.i.d. bivariate normally distributed, the distribution function of **Pearson's Correlation Coefficient estimator** r_P is related to the t -distribution. [56]

- (a) CDF of $\frac{r_P\sqrt{n-2}}{\sqrt{1-r_P}}$ has the CDF of a t -distribution with $n - 2$ degrees of freedom. [56]

9. For any value of ρ_P , but still assuming that the pairs (X_i, Y_i) are bivariate normally distributed, $z_{r_P} = 0.5[\log(1 + r_P) - \log(1 - r_P)]$ is approximately normally distributed with mean $0.5[\log(1 + \rho_P) - \log(1 - \rho_P)]$ and variance $\frac{1}{n-3}$. [56]

Transformation of $r_P \mapsto z_{r_P}$ is called **Fisher z-transformation**. [7, 56]

10. Under the assumption of normality, it is common to use the Fisher z-transformation and calculate the $100\%(1 - \alpha)$ confidence interval by [56]

$$\left[\frac{1}{2} \log\left(\frac{1+r_P}{1-r_P}\right) - \frac{z_{1-\alpha/2}}{\sqrt{n-3}}, \frac{1}{2} \log\left(\frac{1+r_P}{1-r_P}\right) + \frac{z_{1-\alpha/2}}{\sqrt{n-3}} \right] \quad [56]$$

with $z_{1-\alpha}$ the p -th upper quantile of the standard normal distribution function.

11. These limits can then be transformed back to the original scale using the inverse transformation $\frac{\exp(2x) - 1}{\exp(2x) + 1}$ of the Fisher z -transformation. Confidence interval in the original scale is [56]

$$\left[\frac{\exp 2[z_{r_P} - z_{1-\alpha/2}/\sqrt{n-3}] - 1}{\exp 2[z_{r_P} - z_{1-\alpha/2}/\sqrt{n-3}] + 1}, \frac{\exp 2[z_{r_P} + z_{1-\alpha/2}/\sqrt{n-3}] - 1}{\exp 2[z_{r_P} + z_{1-\alpha/2}/\sqrt{n-3}] + 1} \right] \quad [56]$$

12. Although we may not prefer the use of Spearman's correlation coefficient over Pearson's product-moment estimator under the assumption of normally and independently distributed pairs $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$, the mean and variance of r_S have been established under this setting. [56]

- (a) Thus the parameter ρ of the bivariate normal distribution can now be estimated by $\hat{\rho} = 2 \sin(\pi r_S / 6)$. [56]

- (b) **Mean:** $\rho_S = \mathbb{E}[r_S] = \frac{6}{(n+1)\pi} (\arcsin(\rho) + (n-2) \arcsin(\rho/2)) \approx \frac{6}{\pi} \arcsin(\rho/2)$ [56]

- (c) **Variance:** $\mathbb{V}[r_S] \approx \frac{1}{n} (1 - 1.1563465\rho^2 + 0.304743\rho^4 + 0.155286\rho^6)$ [56]

13. $\mathbb{E}[X] = \mu_X$ [56] | 15. $\mathbb{V}[X] = \sigma_X^2$ [56]

14. $\mathbb{E}[Y] = \mu_Y$ [56] | 16. $\mathbb{V}[Y] = \sigma_Y^2$ [56]

9.2.3. Marginals and Conditionals

1. Let X and Y be two multivariate random variables, that may have different dimensions. [49]

2. Gaussian distribution in terms of the concatenated states $[\mathbf{x}^\top, \mathbf{y}^\top]$: [49]

$$P(x, y) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{bmatrix}\right), \quad [49]$$

where $\boldsymbol{\Sigma}_{xx} = \text{Cov}[\mathbf{x}, \mathbf{x}]$ and $\boldsymbol{\Sigma}_{yy} = \text{Cov}[\mathbf{y}, \mathbf{y}]$ are the marginal covariance matrices of \mathbf{x} and \mathbf{y} , respectively, and $\boldsymbol{\Sigma}_{xy} = \text{Cov}[\mathbf{x}, \mathbf{y}]$ is the cross-covariance matrix between \mathbf{x} and \mathbf{y} . [49]

3. The conditional distribution $P(\mathbf{x}|\mathbf{y})$ is Gaussian: [49]

$$\begin{array}{ll} \text{(a)} & P(\mathbf{x}|\mathbf{y}) = N(\boldsymbol{\mu}_{x|y}, \boldsymbol{\Sigma}_{x|y}) \quad [49] \\ \text{(b)} & \boldsymbol{\mu}_{x|y} = \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy} \boldsymbol{\Sigma}_{yy}^{-1} (\mathbf{y} - \boldsymbol{\mu}_y) \quad [49] \\ \text{(c)} & \boldsymbol{\Sigma}_{x|y} = \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy} \boldsymbol{\Sigma}_{yy}^{-1} \boldsymbol{\Sigma}_{yx} \end{array} \quad [49]$$

where \mathbf{y} -value is an observation and no longer random. [49]

4. The marginal distribution $P(\mathbf{x})$ of a joint Gaussian distribution $P(\mathbf{x}, \mathbf{y})$ is itself Gaussian and computed by applying the sum rule and given by: [49]

$$P(\mathbf{x}) = \int P(\mathbf{x}, \mathbf{y}) d\mathbf{y} = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx}) \quad [49]$$

9.2.4. Product of Gaussian Densities

1. The product of two Gaussians $\mathcal{N}(\mathbf{x} | \mathbf{a}, \mathbf{A}) \mathcal{N}(\mathbf{x} | \mathbf{b}, \mathbf{B})$ is a Gaussian distribution scaled by a $c \in \mathbb{R}$, given by $c \mathcal{N}(\mathbf{x} | \mathbf{c}, \mathbf{C})$ with: [49]

$$\text{(a)} \quad \mathbf{C} = (\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1} \quad [49]$$

$$\text{(b)} \quad \mathbf{c} = \mathbf{C}(\mathbf{A}^{-1}\mathbf{a} + \mathbf{B}^{-1}\mathbf{b}) \quad [49]$$

$$\text{(c)} \quad c = (2\pi)^{-D/2} |\mathbf{A} + \mathbf{B}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{a} - \mathbf{b})^\top (\mathbf{A} + \mathbf{B})^{-1}(\mathbf{a} - \mathbf{b})\right) \quad [49]$$

2. The scaling constant c itself can be written in the form of a Gaussian density either in \mathbf{a} or in \mathbf{b} with an “inflated” covariance matrix $\mathbf{A} + \mathbf{B}$, i.e., $c = \mathcal{N}(\mathbf{a} | \mathbf{b}, \mathbf{A} + \mathbf{B}) = \mathcal{N}(\mathbf{b} | \mathbf{a}, \mathbf{A} + \mathbf{B})$. [49]

9.2.5. Farlie-Gumbel-Morgenstern (FGM) Family of Distributions

1. if we choose F_X and F_Y normal CDFs, we have created a bivariate CDF for X and Y that has marginal normal CDFs, but which is **not equal** to the bivariate normal PDF given by the bivariate PDF: [56]

$$f(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{z_1^2 - 2\rho z_1 z_2 + z_2^2}{2(1-\rho^2)}\right) \quad [56]$$

9.2.6. Mixtures of Probability Distributions

1. Consider a mixture of two univariate Gaussian densities [49]

$$P(x) = \alpha P_1(x) + (1 - \alpha) P_2(x) \quad [49]$$

where the scalar $0 < \alpha < 1$ is the mixture weight, and $P_1(x)$ and $P_2(x)$ are univariate Gaussian densities with different parameters, i.e., $(\mu_1, \sigma_1^2) \neq (\mu_2, \sigma_2^2)$. [49]

2. the mean of the mixture density $P(x)$ is given by the weighted sum of the means of each random variable: $\mathbb{E}[x] = \alpha\mu_1 + (1 - \alpha)\mu_2$ [49]

3. The variance of the mixture density $P(x)$ is given by: [49]

$$\mathbb{V}[x] = [\alpha\sigma_1^2 + (1-\alpha)\sigma_2^2] + [(\alpha\mu_1^2 + (1-\alpha)\mu_2^2) - (\alpha\mu_1 + (1-\alpha)\mu_2)^2] \quad [49]$$

This is also called "law of total variance". [49]

9.2.6.1. Conditionally independent

- Assume that Z is normally distributed with mean μ_Z and variance σ_Z^2 and conditional PDFs $f_{X|Z}(x|z)$ and $f_{Y|Z}(y|z)$ are given by $f_{X|Z}(x|z) = \frac{1}{\sigma_1} \phi\left(\frac{x - \mu_1 - z}{\sigma_1}\right)$ and $f_{Y|Z}(y|z) = \frac{1}{\sigma_2} \phi\left(\frac{y - \mu_2 - z}{\sigma_2}\right)$, with ϕ the standard normal PDF, then: [56]

$$f(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{z_1^2 - 2\rho z_1 z_2 + z_2^2}{2(1-\rho^2)}\right) \quad [56]$$

with

$(a) \mu_X = \mu_Z + \mu_1$ $(b) \mu_Y = \mu_Z + \mu_2$ $(c) \sigma_X^2 = \sigma_Z^2 + \sigma_1^2$	$[56]$ $[56]$ $[56]$	$(d) \sigma_Y^2 = \sigma_Z^2 + \sigma_2^2$ $(e) \rho = \frac{\sigma_Z^2}{\sqrt{(\sigma_Z^2 + \sigma_1^2)(\sigma_Z^2 + \sigma_2^2)}}$	$[56]$ $[56]$
--	----------------------------	---	------------------

9.2.7. Correlation Tests for Numerical Variables ($H_0 : \rho_P = 0$)

- To test the dependency between two normally distributed variables X and Y , Pearson's correlation coefficient can be applied. [56]
- The null hypothesis of independence is then formulated as $H_0 : \rho_P = 0$ against the alternative hypothesis $H_a : \rho_P \neq 0$, with ρ_P the correlation coefficient of the bivariate normal distribution function (which equals Pearson's definition of correlation coefficient). [56]
- Under normality, independence is the same as being uncorrelated, but this may not be true when X and Y do not follow a normal distribution. [56]

- The test statistic for the null hypothesis $H_0 : \rho_P = 0$ is $T_P = \frac{r_P \sqrt{n-2}}{\sqrt{1-r_P^2}}$ with $r_P = \frac{S_{XY}}{S_X S_Y}$ Pearson's product moment estimator. [56]

- Under the assumption of normality and under the null hypothesis $H_0 : \rho_P = 0$ the test statistic T_P has a t-distribution with $n - 2$ degrees of freedom. Thus when the observed value t_P is larger than the upper $\alpha/2$ -quantile of the t-distribution with $n - 2$ degrees of freedom or smaller than the lower $\alpha/2$ -quantile of the t-distribution with $n - 2$ degrees of freedom, the null hypothesis $H_0 : \rho_P = 0$ is rejected. If the observed value t_P does not deviate enough from zero we cannot reject the null hypothesis (but this means that there is no evidence that the null hypothesis is correct). [56]

- we also use Fisher's z-transformation on Pearson's product moment estimator. this alternative confidence interval could also have been used to test $H_0 : \rho_P = 0$ and is often recommended above T_P . The reason is that the Fisher z-transformed Pearson's product moment estimator would more quickly reject the null hypothesis than T_P when the alternative hypothesis is true. Thus the Fisher z-transformed Pearson's product moment estimator has a slightly higher power than the test statistic T_P , while they both have the same type 1 error. [56]

Explanation: [7]

- Original confidence interval (from TP): [7]

- test statistic: $T_P = \frac{r_P \sqrt{n-2}}{\sqrt{1-r_P^2}}$ [7]

- This uses the t-distribution to form the CI for ρ . [7]

- Problem: The sampling distribution of r_P is not symmetric (especially for small n or when ρ is large in magnitude). [7]

- iv. So the resulting CI can be distorted. [7]
- (b) Alternative confidence interval (Fisher's z-transformation)
 - i. Fisher proposed transforming r_P with: $z = \tanh^{-1}(r_P) = \frac{1}{2} \ln \left(\frac{1-r_P}{1+r_P} \right)$ [7]
 - ii. Under H_0 , this is approximately normal: $z \sim \mathcal{N} \left(\tanh^{-1}(\rho), \frac{1}{n-3} \right)$ [7]
 - iii. Confidence interval for ρ is then built by transforming back: $\rho \in \tanh(z \pm z_{\alpha/2} \sqrt{\frac{1}{n-3}})$ [7]
 - iv. This CI is more symmetric and accurate than the one based directly on T_P . [7]
- (c) Why is the Fisher z-based CI better than T_P ? [7]
 - i. Both methods have the same Type I error rate (they're valid tests). [7]
 - ii. But Fisher's z-transformation makes the distribution closer to normal → so:
 - A. More power (rejects false H_0 more often). [7]
 - B. CI has better coverage properties (closer to the nominal 95%). [7]
 - C. Especially useful for small-to-moderate n and correlations far from 0. [7]
- 7. As an alternative approach we could also have used another correlation estimator, like Kendall's tau or Spearman's rho estimators. Then the null hypothesis changes (of course) to $H_0 : \tau_K = 0$ and $H_0 : \rho_S = 0$ for Kendall's tau and Spearman's rho, respectively. [56]
 - (a) If such a null hypothesis were true, the two variables X and Y may still be **dependent**. It merely says that the two variables X and Y are **uncorrelated**. [56]
 - (b) The 95% confidence intervals based on the Fisher z-transformation can then be used to test the corresponding null hypothesis. [56]
 - (c) These alternative correlation coefficients are typically used when X and/or Y are continuous but not normally distributed. [56]
 - (d) If the null hypothesis is rejected, i.e., the value zero is not contained in the 95% confidence interval, the variables X and Y are considered dependent. [56]
 - (e) If one or both variables X and Y are discrete, care should be taken in using any of the described methods. In this case, there will most likely be ties, i.e., there exists pairs of data for which the pairs cannot be ordered, either by the first dimension or by the second dimension (or both). For such variables X and Y , the distribution function of the test statistics just described does not have the same distribution function that was used for the test statistic on continuous variables X and Y . [56]

9.2.8. Sampling from Multivariate Gaussian Distributions

1. the **random sampling** consists of three stages: [49]
 - (a) we need a source of pseudo-random numbers that provide a uniform sample in the interval $[0, 1]$ [49]
 - (b) we use a non-linear transformation such as the Box-Muller transform to obtain a sample from a univariate Gaussian [49]
 - (c) we collate a vector of these samples to obtain a sample from a multivariate standard normal $\mathcal{N}(\mathbf{0}, \mathbf{I})$. [49]
2. For a general multivariate Gaussian, that is, where the mean is non zero and the covariance is not the identity matrix, we use the properties of linear transformations of a Gaussian random variable. Assume we are interested in generating samples $\mathbf{x}_i, i = 1, \dots, n$, from a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. [49]

3. To obtain samples from a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, we can use the properties of a linear transformation of a Gaussian random variable: [49]
- If $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\mu}$, where $\mathbf{A}\mathbf{A}^\top = \boldsymbol{\Sigma}$ is Gaussian distributed with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. [49]
 - One convenient choice of \mathbf{A} is to use the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^\top$. The Cholesky decomposition has the benefit that \mathbf{A} is triangular, leading to efficient computation. [49]

9.2.9. Summary

- Notation:** $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ [32]
- Parameters:**
 - $\boldsymbol{\mu} \in \mathbb{R}^k$ — location [32]
 - $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$: covariance (positive semi-definite matrix) [32]
- Support/ Rand Var:** $\mathbf{x} \in \boldsymbol{\mu} + \text{span}(\boldsymbol{\Sigma}) \subseteq \mathbb{R}^k$ [32]
- PDF:** $(2\pi)^{-k/2} \det(\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$, exists only when $\boldsymbol{\Sigma}$ is positive-definite [32]
- Mean:** $\boldsymbol{\mu}$ [32]
- Mode:** $\boldsymbol{\mu}$ [32]
- Variance:** $\boldsymbol{\Sigma}$ [32]
- Entropy:** $\frac{k}{2} \log(2\pi e) + \frac{1}{2} \log(\det(\boldsymbol{\Sigma}))$ [32]
- Moment-generating function (MGF):** $\exp\left(\boldsymbol{\mu}^\top \mathbf{t} + \frac{1}{2} \mathbf{t}^\top \boldsymbol{\Sigma} \mathbf{t}\right)$ [32]
- Characteristic function (CF):** $\exp\left(i\boldsymbol{\mu}^\top \mathbf{t} - \frac{1}{2} \mathbf{t}^\top \boldsymbol{\Sigma} \mathbf{t}\right)$ [32]

9.3. Standard Normal Distribution ($N(0, 1)$)

- When we choose $\mu = 0$ and $\sigma = 1$ in a Normal Distribution, its called Standard Normal Distribution.
[56]

9.3.1. PDF ($f(x)$ or $f(x)$ or $\phi(x)$)

$$\phi(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}} \quad [56]$$

9.3.2. Bivariate Standard Normal Distribution

- Consider a bivariate standard Gaussian random variable X and performed a linear transformation \mathbf{Ax} on it.
[49]
- The outcome is a Gaussian random variable with mean zero and covariance \mathbf{AA}^\top .
[49]
- Adding a constant vector will change the mean of the distribution, without affecting its variance, that is, the random variable $\mathbf{x} + \boldsymbol{\mu}$ is Gaussian with mean $\boldsymbol{\mu}$ and identity covariance. Hence, any linear/affine transformation of a Gaussian random variable is Gaussian distributed.
[49]

9.4. Lognormal Distribution ($\text{Lognormal}(\mu, \sigma^2)$, $LN(\mu, \sigma^2)$)

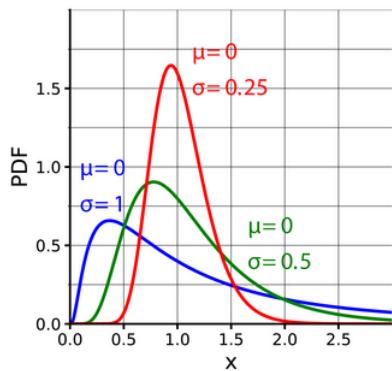


Fig. 9.4: Lognormal Distribution: PDF [30]

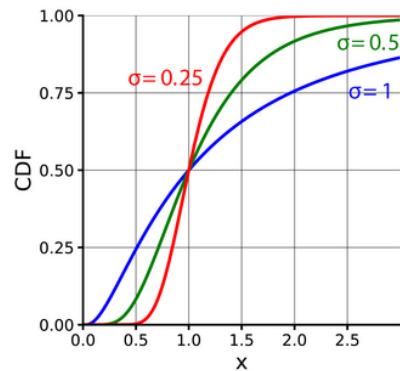


Fig. 9.5: Lognormal Distribution: CDF [30]

9.4.1. PDF

$$f_L(x) = f_{\mu, \sigma}(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left\{-\frac{(\log(x) - \mu)^2}{2\sigma^2}\right\} = \frac{1}{x\sigma} \phi\left(\frac{\log(x) - \mu}{\sigma}\right) \quad [56]$$

1. the lognormal PDF describes populations with positive values [56]
2. relative standard deviation is constant for the lognormal PDF. This means that larger values demonstrate larger variability, but the ratio with variability is constant whether we observe smaller or larger values. [56]
3. the lognormal PDF is not symmetric like the normal PDF, which makes sense when values are limited from below, but not from above. [56]
4. If the population values can be described by a lognormal PDF, the normal PDF would then describe the logarithmic transformation of the population values (using the natural logarithm). [56]
5. The parameters μ and σ have a different meaning in the lognormal PDF than in the normal PDF. They do represent the population mean and standard deviation, but only for the logarithmic transformed values of the population. Their meaning in relation to the mean and standard deviation of the population values in the original scale is now more complicated. [56]

9.4.2. Bivariate Lognormal Distributions

1. If we assume that (X, Y) is bivariate normal, with parameters $\mu_X, \mu_Y, \sigma_X, \sigma_Y$, and ρ , then $(\exp(X), \exp(Y))$ has a bivariate lognormal distribution function. [56]

$$\begin{aligned} \text{Cov}(\exp(X), \exp(Y)) &= \mathbb{E}[\exp(X)\exp(Y)] - \mathbb{E}[\exp(X)]\mathbb{E}[\exp(Y)] \\ &= \mathbb{E}[\exp(X+Y)] - \exp(\mu_X + 0.5\sigma_X^2)\exp(\mu_Y + 0.5\sigma_Y^2) \\ &= \exp(\mu_X + \mu_Y + 0.5(\sigma_X^2 + 2\rho\sigma_X\sigma_Y + \sigma_Y^2)) - \exp(\mu_X + 0.5\sigma_X^2)\exp(\mu_Y + 0.5\sigma_Y^2) \\ &= \exp(\mu_X + 0.5\sigma_X^2 + \mu_Y + 0.5\sigma_Y^2)(\exp(\rho\sigma_X\sigma_Y) - 1) = \mathbb{E}[\exp(X)]\mathbb{E}[\exp(Y)](\exp(\rho\sigma_X\sigma_Y) - 1) \end{aligned} \quad [56]$$

This covariance is quite different from the covariance $\rho\sigma_X\sigma_Y = \text{Cov}[X, Y]$ in the logarithmic scale. [56]

3. Pearson's Correlation Coefficient: $\rho_P = \frac{\exp(\rho\sigma_X\sigma_Y) - 1}{\sqrt{(\exp(\sigma_X^2) - 1)(\exp(\sigma_Y^2) - 1)}}$ [56]
4. When the variances of X and Y are the same, say $\sigma_X^2 = \sigma_Y^2 = \sigma^2$, Pearson's correlation coefficient reduces to $\rho_P = \frac{\exp(\rho\sigma^2) - 1}{\exp(\sigma^2) - 1}$. [56]

5. This correlation is a decreasing function in σ^2 . When σ^2 is close to zero the correlation is ρ and when σ^2 becomes very large the correlations converges to zero. On the other hand, the correlation is relatively constant for the variance $\sigma^2 \leq 1$ [56]

9.4.3. Summary

1. **Notation:** Lognormal (μ, σ^2) OR $\mathcal{LN}(\mu, \sigma^2)$ [30, 56]
2. **Parameters:**
 - (a) $\mu \in (-\infty, +\infty)$: logarithm of location [30, 56]
 - (b) $\sigma > 0$: logarithm of scale [30, 56]
3. **Support:** $x \in (0, +\infty)$ [30, 56]
4. **PDF:** $\frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right)$ [30, 56]
5. **CDF:** $\frac{1}{2} \left[1 + \operatorname{erf}\left(\frac{\ln x - \mu}{\sigma\sqrt{2}}\right) \right] = \Phi\left(\frac{\ln x - \mu}{\sigma}\right)$ [30]
6. **Quantile:** $\exp\left(\mu + \sqrt{2\sigma^2} \operatorname{erf}^{-1}(2p - 1)\right) = \exp(\mu + \sigma\Phi^{-1}(p))$ [30]
7. **Mean:** $\exp\left(\mu + \frac{\sigma^2}{2}\right)$ [30]
8. **Median:** $\exp(\mu)$ [30]
9. **Mode:** $\exp(\mu - \sigma^2)$ [30]
10. **Variance:** $[\exp(\sigma^2) - 1] \exp(2\mu + \sigma^2)$ [30]
11. **Skewness:** $[\exp(\sigma^2) + 2] \sqrt{\exp(\sigma^2) - 1}$ [30]
12. **Excess kurtosis:** $\exp(4\sigma^2) + 2\exp(3\sigma^2) + 3\exp(2\sigma^2) - 6$ [30]
13. **Entropy:** $\log_2\left(\sqrt{2\pi e} \sigma e^\mu\right)$ [30]
14. **Fisher information:** $\frac{1}{\sigma^2} \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ [30]
15. **Method of moments:**
 - (a) $\mu = \ln E[X] - \frac{1}{2} \ln \left(\frac{\operatorname{Var}[X]}{E[X]^2} + 1 \right)$, [30]
 - (b) $\sigma = \sqrt{\ln \left(\frac{\operatorname{Var}[X]}{E[X]^2} + 1 \right)}$ [30]
16. **Expected shortfall:** $\frac{e^{\mu+\frac{\sigma^2}{2}}}{2p} \left[1 + \operatorname{erf}\left(\frac{\sigma}{\sqrt{2}} + \operatorname{erf}^{-1}(2p - 1)\right) \right] = \frac{e^{\mu+\frac{\sigma^2}{2}}}{1-p} [1 - \Phi(\Phi^{-1}(p) - \sigma)]$ [30]
17. **Relative Standard Deviation:** $\sqrt{\exp\{\sigma^2\} - 1}$
(RSD = standard deviation divided by the mean) [56]

9.4.4. Lognormally Distributed Populations

1. If X is lognormally $\mathcal{LN}(\mu, \sigma^2)$ distributed then $\log(X)$ is normal $\mathcal{N}(\mu, \sigma^2)$. [56]
2. If X_1, X_2, \dots, X_n are i.i.d. lognormally distributed, the sample average and sample variance of the transformed random variables $\log(X_1), \log(X_2), \dots, \log(X_n)$, with \log the natural logarithm, are given by $\bar{X}_{\log} = \frac{1}{n} \sum_{i=1}^n \log(X_i)$ and $S_{\log}^2 = \frac{1}{n-1} \sum_{i=1}^n (\log(X_i) - \bar{X}_{\log})^2$ [56]
3. These sample statistics are **unbiased estimates** of μ and σ^2 , respectively. [56]
4. The distribution function of \bar{X}_{\log} is normally distributed $\mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$ and $\frac{(n-1)S_{\log}^2}{\sigma^2}$ has a chi-square distribution function with $n-1$ degrees of freedom. [56]
5. the geometric average $\exp(\bar{X}_{\log}) = \prod_{i=1}^n \sqrt[n]{X_i}$ is lognormally distributed. [56]

9.4.5. Method of Moments Estimation (MME)

1. consider the random variable X that is lognormally distributed, i.e. $X \sim \mathcal{LN}(\mu, \sigma^2)$ [56]
2. The PDF (f_L) contains only two parameters $\theta_1 = \mu$ and $\theta_2 = \sigma^2$ [56]
3. we only need to solve the two equations:
 - (a) $M_1 = \bar{X} = \mu(f_L) = \exp(\mu + 0.5\sigma^2)$ [56]
 - (b) $M_2 = \sigma^2(f_L) = \exp(2\mu + \sigma^2) (\exp(\sigma^2) - 1)$ [56]
4. using $\frac{\sigma^2(f_L)}{\mu^2(f_L)} = \exp(\sigma^2) - 1$, we obtain that σ^2 can be estimated by

$$\tilde{\sigma}^2 = \log\left(1 + \frac{M_2}{\bar{X}^2}\right) = \log(\bar{X}^2 + M_2) - 2\log(\bar{X})$$
 [56]
5. Using the estimator $\tilde{\sigma}^2$, we obtain the moment estimator for μ , which is given by

$$\mu = \log(\bar{X}) - 0.5[\log(\bar{X}^2 + M_2) - 2\log(\bar{X})] = 2\log(\bar{X}) - 0.5\log(\bar{X}^2 + M_2)$$
 [56]
6. **alternative approach:** consider the set of transformed random variables $\log(X_1), \log(X_2), \dots, \log(X_n)$. [56]
 - (a) These random variables are normally distributed with mean μ and variance σ^2 . [56]
 - (b) The moment estimators in this setting are now $\tilde{\mu} = \bar{X}_{\log}$ and $\tilde{\sigma}^2 = \frac{n-1}{n} S_{\log}^2$, where \bar{X}_{\log} and S_{\log}^2 . [56]

9.4.6. Maximum Likelihood Estimation (MLE)

1. the set of parameters that we want to estimate is given by $\theta_1 = \mu$ and $\theta_2 = \sigma^2$. [56]
2. The log likelihood in this setting is equal to

$$\sum_{i=1}^n \log f_L(X_i) = \sum_{i=1}^n \left(-\frac{(\log(X_i) - \mu)^2}{2\sigma^2} - \log(\sqrt{2\pi}) - \log(X_i) - \log(\sigma) \right)$$
 [56]
3. The terms $\log \sqrt{2\pi}$ and $\log(X_i)$ can essentially be ignored, since they will be the same whatever we choose for μ and σ . [56]
4. Taking the derivatives with respect to μ and σ and equating them to zero we obtain the following equations

$$\sum_{i=1}^n \left(\frac{\log(X_i) - \mu}{\sigma^2} \right) = 0 \quad \sum_{i=1}^n \left(\frac{(\log(X_i) - \mu)^2}{\sigma^3} - \frac{1}{\sigma} \right) = 0$$
 [56]

5. Solving these equations leads to the solutions [56]

$$\hat{\mu} = \bar{X}_{\log} = \sum_{i=1}^n \log(X_i) \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (\log(X_i) - \bar{X}_{\log})^2 = \frac{n-1}{n} S_{\log}^2 \quad [56]$$

6. These solutions are the MME of the logarithmically transformed random variables. This implies that the MLE for σ^2 is not unbiased, since the expected value of $\hat{\sigma}^2$ is equal to $\mathbb{E}[\hat{\sigma}^2] = \mathbb{E}\left[\frac{n-1}{n}S_{\log}^2\right] = \frac{n-1}{n}\sigma^2$, although this bias vanishes when the sample size gets large. [56]
7. Using the ML estimates for the density parameters to estimate the population parameters $\mu(f_L)$ or $\sigma^2(f_L)$ in the logarithm scale does not provide unbiased estimator for the mean of the population in the original scale, since the MLE estimate $\exp(\hat{\mu} + 0.5 \hat{\sigma}^2)$ is unequal to the expected value of the sample mean \bar{X} . [56]

9.4.7. Tukey's Method for Outliers (Using IQR)

1. For the lognormal distribution function $\Phi\left(\frac{\ln(x) - \mu}{\sigma}\right)$ though, the lower and upper criteria now become [56]

$$\exp(\mu - 0.67449 \cdot \sigma) - 1.5(\exp(\mu + 0.67449 \cdot \sigma) - \exp(\mu - 0.67449 \cdot \sigma))$$
 and [56]

$$\exp(\mu + 0.67449 \cdot \sigma) + 1.5(\exp(\mu + 0.67449 \cdot \sigma) - \exp(\mu - 0.67449 \cdot \sigma)),$$
 respectively. [56]
2. Calculating the proportion of population units outside Tukey's criteria depends only on σ , not on μ . [56]
3. For the lognormal distribution function the proportion outside Tukey's criteria can be as high as 10%. This would typically occur at the upper tail and not on the lower tail, since the proportion below the lower tail criterion would become equal to zero. [56]

9.5. (Continuous) Uniform Distribution ($U(a, b)$)

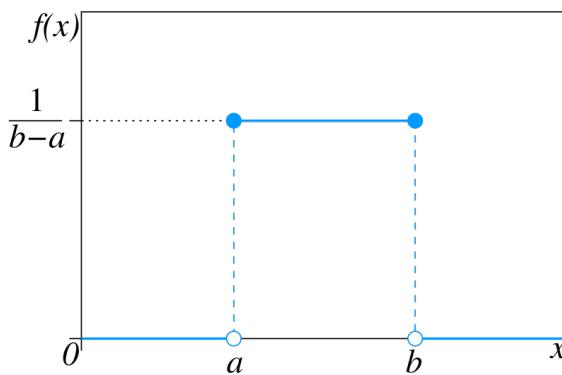


Fig. 9.6: Uniform Distribution: PDF [27]

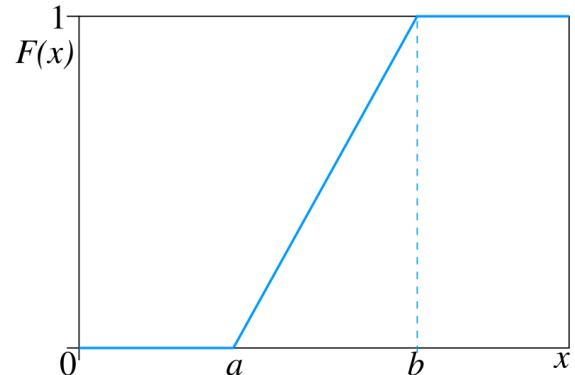


Fig. 9.7: Uniform Distribution: CDF [27]

9.5.1. PDF

$$f_{a,b}(x) = \frac{1}{b-a}, \quad x \in [a, b] \quad [56]$$

- If the random measurement error would be described by a uniform PDF, being close to or far away from zero would be equally likely. However, the uniform PDF has a finite domain, which means that the density is positive on an interval, say $[a, b]$, with $a < b$ and $a, b \in \mathbb{R}$, but zero everywhere else. [56]

9.5.2. Summary

- Notation:** $\mathcal{U}_{[a,b]}$ [27]
- Parameters:** $-\infty < a < b < \infty$ [27]
- Support:** $[a, b]$ [27]
- PDF:** $\begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$ [27, 56]
- CDF:** $\begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } x \in [a, b] \\ 1 & \text{for } x > b \end{cases}$ [27]
- Mean:** $\frac{1}{2}(a+b)$ [27]
- Median:** $\frac{1}{2}(a+b)$ [27]
- Mode:** any value in (a, b) [27]
- Variance:** $\frac{1}{12}(b-a)^2$ [27]
- Median absolute deviation (MAD):** $\frac{1}{4}(b-a)$ [27]
- Skewness:** 0 [27]
- Excess kurtosis:** $-\frac{6}{5}$ [27]
- Entropy:** $\log(b-a)$ [27]

14. **Moment-generating function (MGF):**
$$\begin{cases} \frac{e^{tb} - e^{ta}}{t(b-a)} & \text{for } t \neq 0 \\ 1 & \text{for } t = 0 \end{cases}$$
 [27]

15. **Characteristic function (CF):**
$$\begin{cases} \frac{e^{itb} - e^{ita}}{it(b-a)} & \text{for } t \neq 0 \\ 1 & \text{for } t = 0 \end{cases}$$
 [27]

9.6. Standard (Continuous) Uniform Distribution ($U(0, 1)$)

9.6.1. PDF

$$f_{a,b}(x) = 1, \quad x \in [0, 1] \quad [56]$$

1. If we draw a population using the standard uniform density, we would be able to make a proper transformation of these uniform values such that the transformed values would describe another density.
[56]

9.7. Exponential Distribution (Exponential(λ))

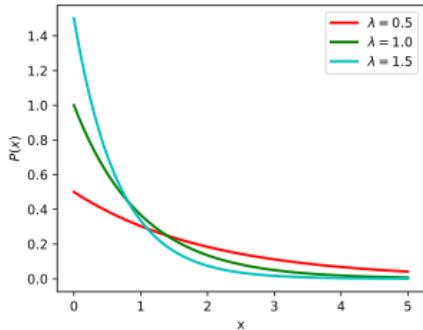


Fig. 9.8: Exponential Distribution: PDF [28]

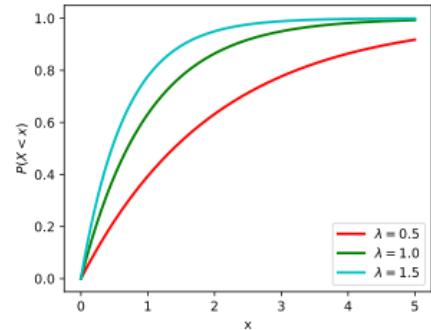


Fig. 9.9: Exponential Distribution: CDF [28]

1. the class of distributions called the exponential family provides the right balance of generality while retaining favorable computation and inference properties. [49]
2. Which class of distributions have finite-dimensional sufficient statistics, that is the number of parameters needed to describe them does not increase arbitrarily? The answer is exponential family distributions [49]

9.7.1. Distribution of the Sample Statistic (T_n)

1. Let X_1, X_2, \dots, X_n are i.i.d. exponentially distributed, i.e. $X_i \sim \exp(\lambda)$. [56]
2. The exponential distribution function is given by $F(x) = 1 - \exp(-\lambda x)$ for $x > 0$ and zero otherwise. [56]

9.7.1.1. Distribution of the Sample Minimum ($T_n = F_{X_{(1)}}(x)$)

1. $F_{X_{(1)}}(x) = 1 - \exp(-n\lambda x)$ for $x > 0$ and zero otherwise [56]
2. Thus this implies that the sample distribution of the minimum $X_{(1)}$ is exponentially distributed, $X_{(1)} \sim \exp(n\lambda)$, but now with parameter $n\lambda$, when X_1, X_2, \dots, X_n are i.i.d. $\exp(\lambda)$ distributed. [56]

9.7.2. Farlie–Gumbel–Morgenstern (FGM) Family

1. assume that F_X and F_Y are exponential CDFs with parameters λ_X and λ_Y , respectively [56]
2. $\mathbb{E}[X(1 - 2F_X(X))] = -[2\lambda_X]^{-1}$ and $\mathbb{E}[Y(1 - 2F_Y(Y))] = -[2\lambda_Y]^{-1}$ [56]
3. $\text{Cov}[X, Y] = -\frac{\alpha}{4\lambda_X\lambda_Y}$ [56]
4. **Pearson's correlation coefficient:** $\rho_P = -\frac{\alpha}{4}$ [56]
5. **Pearson's correlation coefficient estimator (r_P):** The dependency parameter α can be estimated by $r_P \bar{X} \bar{Y}$. Indeed, r_P estimates $\alpha \lambda_X^{-1} \lambda_Y^{-1}$ and the sample averages \bar{X} and \bar{Y} estimate the parameters λ_X^{-1} and λ_Y^{-1} , respectively. [56]

9.7.3. Exponential Family

1. An exponential family is a family of probability distributions, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$, of the form [49]

$$P(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp(\langle \boldsymbol{\theta}, \phi(\mathbf{x}) \rangle - A(\boldsymbol{\theta})) \quad [49]$$

where $\phi(\mathbf{x})$ is the vector of sufficient statistics. [49]

By Default: $\langle \boldsymbol{\theta}, \phi(\mathbf{x}) \rangle = \boldsymbol{\theta}^\top \phi(\mathbf{x})$ (standard dot product) [49]

- (a) The factor $h(\mathbf{x})$ can be absorbed into the dot product term by adding another entry ($\log h(\mathbf{x})$) to the vector of sufficient statistics $\phi(\mathbf{x})$, and constraining the corresponding parameter $\theta_0 = 1$. [49]
 - (b) The term $A(\boldsymbol{\theta})$ is the **normalization constant** that ensures that the distribution sums up or integrates to one and is called the **log-partition function**. [49]
 - (c) The parameters θ are called the **natural parameters**. [49]
2. Note that the form of the exponential family is essentially a particular expression of $g_\theta(\phi(\mathbf{x}))$ in the Fisher-Neyman theorem. [49]
 3. Exponential families provide a convenient way to find **conjugate pairs of distributions**. [49]
 - (a) Consider the random variable X is a member of the exponential family: [49]

$$P(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp(\langle \boldsymbol{\theta}, \phi(\mathbf{x}) \rangle - A(\boldsymbol{\theta})) \quad [49]$$

- (b) Every member of the exponential family has a conjugate prior: [49]

$$P(\boldsymbol{\theta}|\gamma) = h_c(\boldsymbol{\theta}) \exp\left(\left\langle \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\theta} \\ -A(\boldsymbol{\theta}) \end{bmatrix} \right\rangle - A_c(\gamma)\right) \quad [49]$$

where $\gamma = \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}$ has dimension $\dim(\boldsymbol{\theta}) + 1$. [49]

- (c) The sufficient statistics of the conjugate prior are $\begin{bmatrix} \boldsymbol{\theta} \\ -A(\boldsymbol{\theta}) \end{bmatrix}$. [49]
- (d) By using the knowledge of the general form of conjugate priors for exponential families, we can derive functional forms of conjugate priors corresponding to particular distributions. [49]

4.

9.7.4. Summary

1. **Parameters:** $\lambda > 0$ (rate or inverse scale) [28, 56]
2. **Support:** $x \in [0, \infty)$ [28, 56]
3. **PDF:** $\lambda e^{-\lambda x}$ [28, 56]
4. **CDF:** $1 - e^{-\lambda x}$ [28]
5. **Quantile:** $-\frac{\ln(1-p)}{\lambda}$ [28]
6. **Mean:** $\frac{1}{\lambda}$ [28]
7. **Median:** $\frac{\ln(2)}{\lambda}$ [28]
8. **Mode:** 0 [28]
9. **Variance:** $\frac{1}{\lambda^2}$ [28]
10. **Skewness:** 2 [28]

-
- 11. **Excess kurtosis:** 6 [28]
 - 12. **Entropy:** $1 - \ln(\lambda)$ [28]
 - 13. **Fisher information:** $\frac{1}{\lambda^2}$ [28]
 - 14. **Expected shortfall:** $\frac{-\ln(1-p) + 1}{\lambda}$ [28]
 - 15. **Moment-generating function (MGF):** $\frac{\lambda}{\lambda - t}$, for $t < \lambda$ [28]
 - 16. **Characteristic function (CF):** $\frac{\lambda}{\lambda - it}$ [28]
 - 17. **Kullback–Leibler divergence:** $\ln \frac{\lambda_0}{\lambda} + \frac{\lambda}{\lambda_0} - 1$ [28]

9.8. Laplace distribution/ Double Exponential Distribution (Laplace(μ, b))

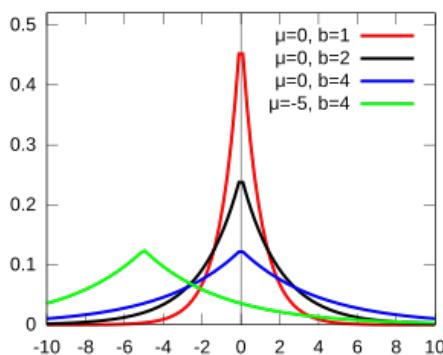


Fig. 9.10: Double Exponential/ Laplace Distribution: PDF [29]

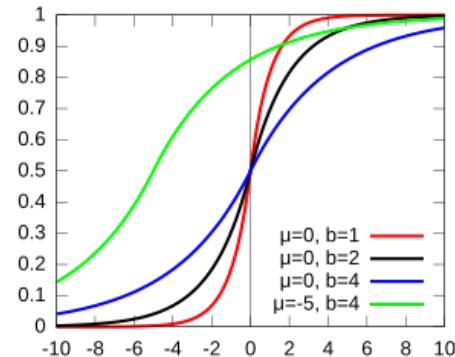


Fig. 9.11: Double Exponential/ Laplace Distribution: CDF [29]

9.8.1. PDF

1. The double exponential is symmetric, like the normal PDF [56]
2. The double exponential PDF can easily be determined using the exponential PDF. The double exponential PDF, for any value $x \in \mathbb{R}$, is defined by $0.5f_\lambda(|x|)$, with $|\cdot|$ the absolute function. [56]

Note: $\mu = 0, b = \frac{1}{\lambda}$ [7]

9.8.2. Summary

1. **Parameters:** [29]
 - (a) μ location (real) [29]
 - (b) $b > 0$ scale (real) [29]
2. **Support:** \mathbb{R} [29, 56]
3. **PDF:** $\frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$ [29, 56]
4. **CDF:**
$$\begin{cases} \frac{1}{2} \exp\left(\frac{x-\mu}{b}\right) & \text{if } x \leq \mu \\ 1 - \frac{1}{2} \exp\left(-\frac{x-\mu}{b}\right) & \text{if } x \geq \mu \end{cases}$$
 [29]
5. **Quantile:**
$$\begin{cases} \mu + b \ln(2F) & \text{if } F \leq \frac{1}{2} \\ \mu - b \ln(2-2F) & \text{if } F \geq \frac{1}{2} \end{cases}$$
 [29]
6. **Mean:** μ [29]
7. **Median:** μ [29]
8. **Mode:** μ [29]
9. **Variance:** $2b^2$ [29]
10. **Skewness:** 0 [29]

-
- 11. **Excess kurtosis:** 3 [29]
 - 12. **Entropy:** $\log(2be)$ [29]
 - 13. **Expected shortfall:**
$$\begin{cases} \mu + b \left(\frac{p}{1-p} \right) (1 - \ln(2p)) & , p < 0.5 \\ \mu + b (1 - \ln(2(1-p))) & , p \geq 0.5 \end{cases}$$
 [29]
 - 14. **Moment-generating function (MGF):** $\frac{\exp(\mu t)}{1 - b^2 t^2}$ for $|t| < 1/b$ [29]
 - 15. **Characteristic function (CF):** $\frac{\exp(\mu it)}{1 + b^2 t^2}$ [29]
 - 16. **Kullback–Leibler divergence:** [29]
 - 17. **Median absolute deviation (MAD):** $b \ln 2$ [29]

9.9. Chi-squared distribution ($\chi^2(k)$ OR χ_k^2)

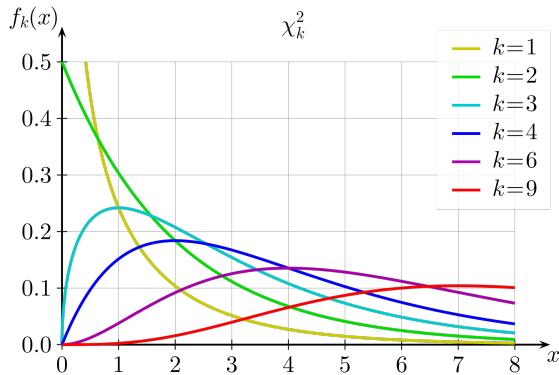


Fig. 9.12: Chi-squared Distribution: PDF [26]

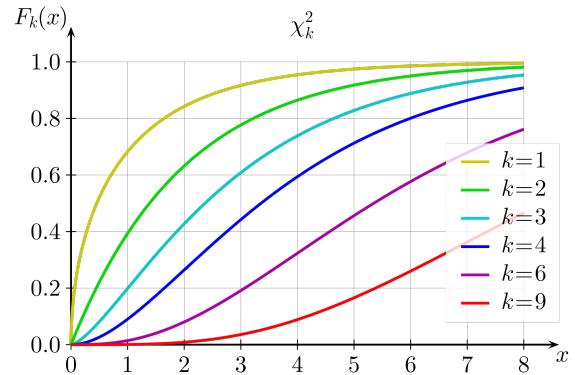


Fig. 9.13: Chi-squared Distribution: CDF [26]

9.9.1. Summary

1. **Parameters:** $k \in \mathbb{N}^*$ (known as "degrees of freedom") [26]
2. **Support:** $x \in (0, +\infty)$ [26]
3. **PDF:** $\frac{1}{2^{k/2}\Gamma(k/2)} x^{(k/2)-1} e^{-x/2}$ [26]
4. **CDF:** $\frac{1}{\Gamma(k/2)} \gamma\left(\frac{k}{2}, \frac{x}{2}\right)$ [26]
5. **Mean:** k [26]
6. **Median:** $\approx k\left(1 - \frac{2}{9k}\right)^3$ [26]
7. **Mode:** $\max(k-2, 0)$ [26]
8. **Variance:** $2k$ [26]
9. **Skewness:** $\sqrt{\frac{8}{k}}$ [26]
10. **Excess kurtosis:** $\frac{12}{k}$ [26]
11. **Entropy:** $\frac{k}{2} + \log\left(2\Gamma\left(\frac{k}{2}\right)\right) + \left(1 - \frac{k}{2}\right) \psi\left(\frac{k}{2}\right)$ [26]
12. **Moment-generating function (MGF):** $(1 - 2t)^{-k/2}$ for $t < \frac{1}{2}$ [26]
13. **Characteristic function (CF):** $(1 - 2it)^{-k/2}$ [26]
14. **Probability-generating function (PGF):** $(1 - 2\ln t)^{-k/2}$ for $0 < t < \sqrt{e}$ [26]

9.10. Student's t-distribution (t_ν)

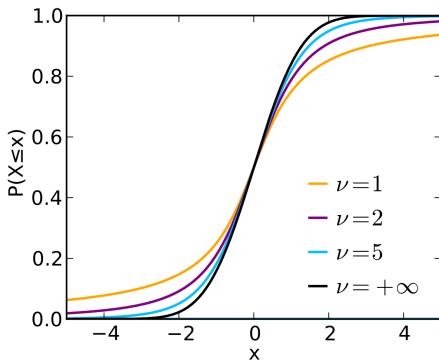


Fig. 9.14: Student's t-distribution: PDF [36]

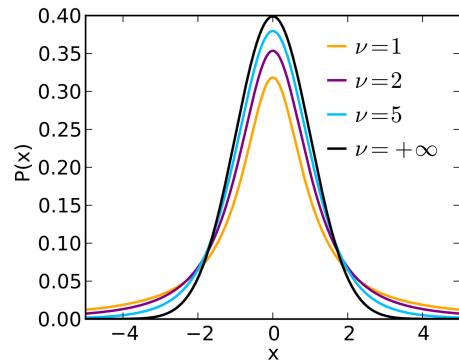


Fig. 9.15: Student's t-distribution: CDF [36]

1. Student's t-density is symmetric around zero. The symmetry implies the following relation for the p -th quantile: $x_p(f_t) = -x_{1-p}(f_t)$. [56]
2. The skewness is zero only when the degrees of freedom are larger than 3. To have a finite kurtosis the degrees of freedom should be larger than 4. [56]

9.10.1. Summary

1. **Parameters:** $\nu > 0$ degrees of freedom (real, almost always a positive integer) [36]
2. **Support:** $x \in (-\infty, \infty)$ [36]
3. **PDF:**
$$\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\pi\nu}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$
 [36]
4. **CDF:**
$$\frac{1}{2} + x\Gamma\left(\frac{\nu+1}{2}\right) \times \frac{{}_2F_1\left(\frac{1}{2}, \frac{\nu+1}{2}; \frac{3}{2}; -\frac{x^2}{\nu}\right)}{\sqrt{\pi\nu}\Gamma\left(\frac{\nu}{2}\right)}$$
 where ${}_2F_1$ is the hypergeometric function [36]
5. **Mean:** 0 for $\nu > 1$, otherwise undefined [36]
6. **Median:** 0 [36]
7. **Mode:** 0 [36]
8. **Variance:** $\frac{\nu}{\nu-2}$ for $\nu > 2$, ∞ for $1 < \nu \leq 2$, otherwise undefined [36]
9. **Skewness:** 0 for $\nu > 3$, otherwise undefined [36]
10. **Excess kurtosis:** $\frac{6}{\nu-4}$ for $\nu > 4$, ∞ for $2 < \nu \leq 4$, otherwise undefined [36]
11. **Entropy:**
$$\frac{\nu+1}{2} \left[\psi\left(\frac{\nu+1}{2}\right) - \psi\left(\frac{\nu}{2}\right) \right] + \ln \left[\sqrt{\nu} B\left(\frac{\nu}{2}, \frac{1}{2}\right) \right]$$
 (nats) [36]
where
(a) ψ is the digamma function [36]
(b) B is the beta function [36]

Chapter 9. Continuous Distributions

12. **Expected shortfall:** $\mu + s \left(\frac{(\nu + [T^{-1}(1-p)]^2) \times \tau(T^{-1}(1-p))}{(\nu - 1)(1-p)} \right)$ where T^{-1} is the inverse standardized Student t CDF, and τ is the standardized Student t PDF [36]
13. **Moment-generating function (MGF):** undefined
14. **Characteristic function (CF):** $\frac{(\sqrt{\nu}|t|)^{\nu/2} K_{\nu/2}(\sqrt{\nu}|t|)}{\Gamma(\nu/2) 2^{\nu/2-1}}$ for $\nu > 0$, where K_ν is the modified Bessel function of the second kind [36]

9.11. Beta distribution ($Beta(\alpha, \beta)$)

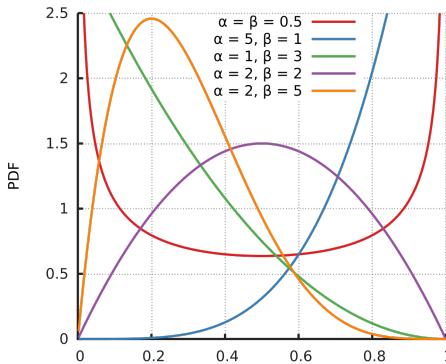


Fig. 9.16: Beta Distribution: PDF [24]

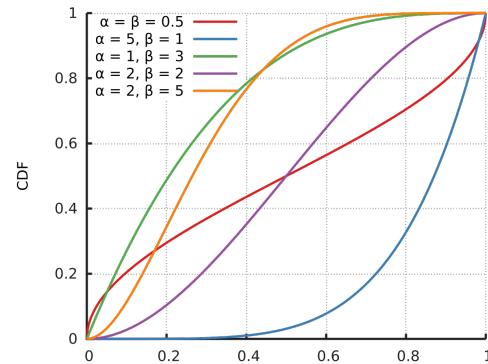


Fig. 9.17: Beta Distribution: CDF [24]

1. The beta distribution is considered a **conjugate prior distribution**. [56]
2. Intuitively, α moves probability mass toward 1, whereas β moves probability mass toward 0. [49]
 - (a) For $\alpha = 1 = \beta$, we obtain the **uniform distribution** $\mathcal{U}[0, 1]$. [49]
 - (b) For $\alpha, \beta < 1$, we get a **bimodal distribution** with spikes at 0 and 1. [49]
 - (c) For $\alpha, \beta > 1$, the distribution is **unimodal**. [49]
 - (d) For $\alpha, \beta > 1$ and $\alpha = \beta$, the distribution is **unimodal**, symmetric, and centered in the interval $[0, 1]$, i.e., the mode/mean is at $\frac{1}{2}$. [49]

9.11.1. Summary

1. **Parameters:**
 - (a) $\alpha > 0$ shape (real) [24]
 - (b) $\beta > 0$ shape (real) [24]
2. **Support:** $x \in [0, 1]$ or $x \in (0, 1)$ [24]
3. **PDF:**
$$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$
 [24]
where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and Γ is the **Gamma function**. [24]
4. **CDF:** $I_x(\alpha, \beta)$ (the regularized incomplete beta function) [24]
5. **Mean:**
 - (a) $\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}$ [24]
 - (b) $\mathbb{E}[\ln X] = \psi(\alpha) - \psi(\alpha + \beta)$ [24]
 - (c) $\mathbb{E}[X \ln X] = \frac{\alpha}{\alpha + \beta} [\psi(\alpha + 1) - \psi(\alpha + \beta + 1)]$ [24]

where ψ is the **digamma function** [24]
6. **Median:** $I_{1/2}^{[-1]}(\alpha, \beta) \approx \frac{\alpha - 1/3}{\alpha + \beta - 2/3}$ for $\alpha, \beta > 1$ (in general) [24]

Chapter 9. Continuous Distributions

7. **Mode:** $\frac{\alpha - 1}{\alpha + \beta - 2}$ for $\alpha, \beta > 1$ [24]
Any value in the domain for $\alpha = \beta = 1$ [24]
No mode if $\alpha < 1$ or $\beta < 1$. Density diverges at 0 for $\alpha \leq 1$, and at 1 if $\beta \leq 1$ [24]
8. **Variance:**
- (a) $\mathbb{V}[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ [24]
(b) $\mathbb{V}[\ln X] = \psi_1(\alpha) - \psi_1(\alpha+\beta)$ [24]
9. **Skewness:** $\frac{2(\beta-\alpha)\sqrt{\alpha+\beta+1}}{(\alpha+\beta+2)\sqrt{\alpha\beta}}$ [24]
10. **Excess kurtosis:** $\frac{6[(\alpha-\beta)^2(\alpha+\beta+1)-\alpha\beta(\alpha+\beta+2)]}{\alpha\beta(\alpha+\beta+2)(\alpha+\beta+3)}$ [24]
11. **Entropy:** $\ln B(\alpha, \beta) - (\alpha - 1) \psi(\alpha) - (\beta - 1) \psi(\beta) + (\alpha + \beta - 2) \psi(\alpha + \beta)$ [24]
12. **Fisher information:** $\begin{bmatrix} \mathbb{V}[\ln X] & \text{Cov}[\ln X, \ln(1-X)] \\ \text{Cov}[\ln X, \ln(1-X)] & \mathbb{V}[\ln(1-X)] \end{bmatrix}$ [24]
13. **Moment-generating function (MGF):** $1 + \sum_{k=1}^{\infty} \left(\prod_{r=0}^{k-1} \frac{\alpha+r}{\alpha+\beta+r} \right) \frac{t^k}{k!}$ [24]
14. **Characteristic function (CF):** ${}_1F_1(\alpha; \alpha + \beta; it)$ (see Confluent hypergeometric function) [24]
15. **Method of moments:**
- (a) $\alpha = \left(\frac{E[X](1-E[X])}{V[X]} - 1 \right) E[X]$ [24]
(b) $\beta = \left(\frac{E[X](1-E[X])}{V[X]} - 1 \right) (1-E[X])$ [24]

CHAPTER 10

RELATIONSHIPS BETWEEN DISTRIBUTIONS

10.1. Binomial-Poisson

1. The Poisson and binomial distribution functions are quite close whenever λ is equal to np and n is relatively large. [56]
2. It can be shown that the Poisson probabilities are the limit of the binomial probabilities when n converges to infinity under the condition that np converges to λ . [56]

10.2. Binomial-Normal

1. Probability calculation with the binomial distribution function can be adequately approximated with a normal distribution function when the mean np and the value $n(1 - p)$ are both larger than 5 and the sample size is at least 20 ($n \geq 20$). [56]

CHAPTER 11

MIX DISTRIBUTIONS

11.1. X, Y: Binary, Z: Std Norm Dist

1. X and Y as binary variables and Z standard normally distributed [56]
2. Binary models of this kind are commonly used to analyze situations where outcomes are limited to two possibilities, such as success/failure or yes/no responses. These models can be extended to handle multiple binary variables, allowing for more complex scenarios. A latent variable (like Z) typically represents an underlying trait or factor—such as ability, risk level, or preference—that influences the observed binary outcomes. [7, 56]

11.1.1. Mixtures of Probability Distributions

11.1.1.1. Conditionally independent

1. **Assumptions:** $f_{X|Z}(1|z) = 1 - f_{X|Z}(0|z) = \phi(\alpha_X + \beta_X z)$ and $f_{Y|Z}(1|z) = 1 - f_{Y|Z}(0|z) = \phi(\alpha_Y + \beta_Y z)$, with $\alpha_X, \beta_X, \alpha_Y, \beta_Y$ unknown parameters and ϕ the standard normal CDF. [56]
2. Then the marginal PMFs $f_X(x)$ and $f_Y(y)$ are given by

$$(a) f_X(1) = 1 - f_X(0) = \phi\left(\frac{\alpha_X}{\sqrt{1 + \beta_X^2}}\right) \quad [56] \quad (b) f_Y(1) = 1 - f_Y(0) = \phi\left(\frac{\alpha_Y}{\sqrt{1 + \beta_Y^2}}\right) \quad [56]$$

3. joint PDF for X and Y :

$$(a) p_{00} = \int_{-\infty}^{\infty} (1 - \phi(\alpha_X + \beta_X z)) (1 - \phi(\alpha_Y + \beta_Y z)) \phi(z) dz \quad [56]$$

$$(b) p_{01} = \int_{-\infty}^{\infty} (1 - \phi(\alpha_X + \beta_X z)) \phi(\alpha_Y + \beta_Y z) \phi(z) dz \quad [56]$$

$$(c) p_{10} = \int_{-\infty}^{\infty} \phi(\alpha_X + \beta_X z) (1 - \phi(\alpha_Y + \beta_Y z)) \phi(z) dz \quad [56]$$

$$(d) p_{11} = \int_{-\infty}^{\infty} \phi(\alpha_X + \beta_X z) \phi(\alpha_Y + \beta_Y z) \phi(z) dz \quad [56]$$

4. Parameters like α_X and α_Y often quantify the difficulty or threshold for each outcome, while parameters such as β_X and β_Y reflect how sensitive or discriminative the outcome is to changes in the latent trait. Larger difficulty parameters indicate harder tasks or stricter thresholds, whereas higher discrimination parameters allow for better differentiation between entities with varying levels of the latent trait. [7, 56]

11.2. X, Y: Exp/ Lognormal, Z: Poisson

1. X and Y : Exponential/ Lognormal distribution and Z : Poisson Distribution [56]

11.2.1. Mixtures of Probability Distributions

11.2.1.1. Conditionally independent

1. assume that the conditional distribution function of X given $Z = z$ is Poisson with parameter $z\lambda_X$, $X|Z=z \sim \mathcal{P}(z\lambda_X)$, and the conditional distribution function of Y given $Z = z$ is Poisson with parameter $z\lambda_Y$, $Y|Z=z \sim \mathcal{P}(z\lambda_Y)$ [56]
2. Conditional expectations: $\mu_X(z) = z\lambda_X$ and $\mu_Y(z) = z\lambda_Y$ [56]
3. $\mu_X = \mathbb{E}[\mu_X(Z)] = \lambda_X \mathbb{E}[Z] = \lambda_X \mu_Z$ and $\mu_Y = \lambda_Y \mu_Z$ [56]
4. $\text{Cov}[\mu_X(Z), \mu_Y(Z)] = \mathbb{E}[\mu_X(Z)\mu_Y(Z)] - \mu_X \mu_Y = \lambda_X \lambda_Y (\mathbb{E}[Z^2] - \mu_Z^2) = \lambda_X \lambda_Y \sigma_Z^2$ [56]
5. The conditional variance of X given Z is given by $\mathbb{V}[X|Z=z] = z\lambda_X$, which is the conditional mean of the Poisson distribution. Thus $\mathbb{E}[\mathbb{V}[X|Z=z]] = \mu_Z \lambda_X$. [56]
6. $\mathbb{V}[\mu_X(Z)] = \mathbb{V}[Z\lambda_X] = \lambda_X^2 \mathbb{V}[Z] = \lambda_X^2 \sigma_Z^2$ [56]
7. $\mathbb{E}[\mathbb{V}[X|Z=z]] + \mathbb{V}[\mu_X(Z)] = \mu_Z \lambda_X + \sigma_Z^2 \lambda_X^2$ [56]
8. $\mathbb{E}[\mathbb{V}[Y|Z=z]] + \mathbb{V}[\mu_Y(Z)] = \mu_Z \lambda_Y + \sigma_Z^2 \lambda_Y^2$ [56]
9. Pearson's correlation coefficient: $\rho_P = \frac{\sigma_Z^2}{\sqrt{(\sigma_Z^2 + \mu_Z \lambda_X^{-1})(\sigma_Z^2 + \mu_Z \lambda_Y^{-1})}}$ [56]

11.3. Beta-Binomial Conjugacy

1. Consider a Binomial random variable $x \sim \text{Bin}(N, \mu)$ where [49]

$$p(x|N, \mu) = \binom{N}{x} \mu^x (1-\mu)^{N-x} \quad x = 0, 1, \dots, N \quad [49]$$

is the probability of finding x times the outcome “heads” in N coin flips, where μ is the probability of a “head”. [49]

2. We place a Beta prior on the parameter μ , that is, $\mu \sim \text{Beta}(\alpha, \beta)$, where [49]

$$P(\mu|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \mu^{\alpha-1} (1-\mu)^{\beta-1} \quad [49]$$

3. If we now observe some outcome $x = h$, that is, we see h heads in N coin flips, we compute the posterior distribution on μ as [49]

$$\begin{aligned} P(\mu|x=h, N, \alpha, \beta) &\propto P(x|N, \mu)P(\mu|\alpha, \beta) \\ &\propto \mu^h (1-\mu)^{N-h} \mu^{\alpha-1} (1-\mu)^{\beta-1} \\ &= \mu^{h+\alpha-1} (1-\mu)^{(N-h)+\beta-1} \\ &\propto \text{Beta}(h+\alpha, N-h+\beta) \end{aligned} \quad [49]$$

i.e., the posterior distribution is a Beta distribution as the prior, i.e., the Beta prior is conjugate for the parameter μ in the Binomial likelihood function. [49]

11.4. Beta-Bernoulli Conjugacy

1. Let $x \in \{0, 1\}$ be distributed according to the Bernoulli distribution with parameter $\theta \in [0, 1]$, that is, $P(x=1|\theta) = \theta$. [49]
2. This can also be expressed as $P(x|\theta) = \theta^x (1-\theta)^{1-x}$. [49]
3. Let θ be distributed according to a Beta distribution with parameters α, β , that is, $P(\theta|\alpha, \beta) \propto \theta^{\alpha-1} (1-\theta)^{\beta-1}$. [49]

4. Multiplying the Beta and the Bernoulli distributions, we get

[49]

$$\begin{aligned} p(\theta|x, \alpha, \beta) &= P(x|\theta)P(\theta|\alpha, \beta) \\ &\propto \theta^x(1-\theta)^{1-x}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \theta^{\alpha+x-1}(1-\theta)^{\beta+(1-x)-1} \\ &\propto P(\theta|\alpha+x, \beta+(1-x)) \end{aligned}$$

[49]

CHAPTER 12

DECISIONS IN UNCERTAINTY

12.1. Bootstrapping

1. The bootstrap provides a very general way to obtain a quantification of the uncertainty of an estimator. [56]
2. a larger sample decreases the variance of an estimator (i.e., the estimator becomes more precise). [56]
3. when estimating a population mean or difference in population means, we find that a smaller population variance leads to a smaller variance of the estimator. [56]
4. The bootstrap has these exact same properties and is easy to carry out for many sample statistics; it provides a first entry into making decisions regarding populations based on sample data. [56]
5. informally, it is quite clear that a large random sample and a relatively small variance of the estimator $\hat{\theta}$ should both increase our confidence regarding statements we can make about the population. [56]

12.1.1. Basic Idea of Bootstrap

1. Given a (random) sample of size n from some population with distribution function F_X , we frequently set out to obtain an estimate of a population parameter $\theta = T(x)$. [56]
2. Our point estimate of the parameter of interest is often what is called the “plug-in estimator” for θ : $\hat{\theta} = T(x_1, \dots, x_n)$; i.e., it is the statistic of interest calculated on the sample data x_1, \dots, x_n . [56]
3. we are interested in the distribution function of $\hat{\theta}$ over repeated samples. We are interested in $F_{\hat{\theta}}$ as it gives us information about the variability of our estimate over repeated samples. Depending on the statistic involved, the sampling plan, and the assumptions one is willing to make about F_X or f_X , we might be able to analytically derive $F_{\hat{\theta}}$. However, obtaining $F_{\hat{\theta}}$ in general (or properties thereof) can be challenging. [56]
4. The bootstrap addresses the problem of deriving $F_{\hat{\theta}}$ using the power of computer simulation. [56]
5. if we have a good estimate of the population distribution function F_X , i.e., \hat{F}_X , we can simply program a computer to obtain M samples (by **simulation**) of size n from \hat{F}_X using the same sampling plan that we have used to collect our initial sample. If \hat{F}_X is close to F_X it does not really matter if we draw from \hat{F}_X or F_X . As long as our estimate of \hat{F}_X is close to the true F_X , our samples of $\hat{F}_{\hat{\theta}}$ can be used to approximate properties of $F_{\hat{\theta}}$ [56]
6. On each sample $m = 1, \dots, M$ we can subsequently compute the statistic of interest, $\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(M)}$ which themselves serve as approximate samples from $F_{\hat{\theta}}$ (approximate as we are using \hat{F}_X , and thus we obtain samples from $\hat{F}_{\hat{\theta}}$). [56]
7. standard error of a statistic can simply be computed by computing the standard deviation of the M samples of the statistic of interest $\hat{SE}(\hat{\theta}) = \sqrt{\frac{\sum_{m=1}^M (\hat{\theta}^{(m)} - \bar{\theta})^2}{M-1}}$ where $\bar{\theta} = \frac{1}{M} \sum_{m=1}^M \hat{\theta}^{(m)}$ [56]
8. we are interested in the variability of our estimator over differently obtained samples, we should adopt our bootstrapping procedure accordingly. [56]
9. Although the bootstrap is appealing as it allows one to quantify the uncertainty for virtually any statistic—by simply replacing tedious analytical work with simple computer operations—one should

always be careful: for complex sampling schemes and complex population distributions, \hat{F}_X , or the resulting M bootstrap samples of the statistic of interest, might not provide a good quantification of the uncertainty associated with $\hat{\theta}$. [56]

10. We can use \hat{F}_X , in combination with computer simulation, to generate bootstrap samples $m = 1, \dots, M$ and compute $\hat{\theta}(m)$ for arbitrary statistics T . [56]

11. Steps:

- (a) we estimate F_X based using our sample x_1, \dots, x_n . This gives us \hat{F}_X . [56]
- (b) we obtain M random samples from \hat{F}_X (each of size n), on which we computed our bootstrap estimates of the statistic of interest $\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(M)}$ which we regard as (approximate) samples from $F_{\hat{\theta}}$. [56]

12. Disadvantages:

- (a) it might be the case that \hat{F}_X is a very poor estimate of F_X . This is often the case when n is small, but it might also be caused by the fact that the original sample x_1, \dots, x_n is not obtained through simple random sampling (resampling with replacement). If the latter is the case, the sampling scheme that was used should be taken into consideration when computing \hat{F}_X . [56]
- (b) the sampling scheme implemented in the second step (resampling step) should mimic the sampling scheme that was originally used: if the M bootstrap samples are generated using a different sampling scheme than the sampling scheme of interest, $F_{\hat{\theta}}$ might not be properly approximated by the M bootstrap samples. [56]
- (c) While the bootstrap provides an easy way of quantifying uncertainty that we can use to make decisions, it is hard in general to make statements about the **quality** of these decisions. [56]

12.1.1.1. Non-Parametric Bootstrap

13. no parametric assumptions regarding F_X are made in this procedure [56]
14. The simplest bootstrap approach for obtaining \hat{F}_X is to simply use the empirical distribution function: the original samples x_1, \dots, x_n in our sample can be used to construct a discrete approximation of F_X by simply giving each unique value v_i in x_1, \dots, x_n probability $\frac{1}{n}$. [56]

12.1.1.2. Parametric Bootstrap

15. in the parametric bootstrap \hat{F}_X is assumed to be of a certain form (e.g., it is assumed to be normal), and plug-in estimates for its parameters (e.g., $\hat{\mu}$ and $\hat{\sigma}^2$ in the normal case) are used to estimate F_X . [56]
16. If the assumptions are correct, the parametric bootstrap is preferable over the non-parametric bootstrap. [56]

12.2. Hypothesis testing

1. **Test Statistics (t):** A single number that summarizes the sample data used to conduct the test hypothesis. [57]
2. ***p*-value:** Probability of observing a test statistics. [57]
3. without making any assumptions regarding the sampling process and/or the population distributions involved, it is practically impossible to say anything about the population based on sample data with full certainty. [56]
4. Hypothesis testing provides a method for making binary decisions that, in many instances, does give us clear quantitative statements about the quality of the decisions we make. [56]

5. Within hypothesis testing the general setup is as follows: we state our decision problem as a choice between two competing hypotheses regarding the population, often called the **null hypothesis** H_0 , and the **alternative hypothesis** H_a . Given that H_0 and H_a are complementary, one of the two **must** be true in the population. [56]

 - (a) **Null Hypothesis** (H_0): A statement of no change and is 0 assumed true until evidence indicates otherwise. [57]
 - (b) **Alternate Hypothesis** (H_a): A statement that the researcher is trying to find evidence to support [57]

6. The subsequent rationale of hypothesis testing is that we assume that the null hypothesis is true and that we gather **sufficient evidence** to demonstrate that it is not true. It defines sufficient evidence such that the probability of making a type 1 error is **at most** α . The level α is called the **significance level** and it is the maximal allowable probability of rejecting the null hypothesis when the null hypothesis is actually true. It is often set equal to a value of $\alpha = 0.05$ or $\alpha = 0.01$. [56]
7. Thus the **goal** of hypothesis testing is to **reject the null hypothesis** on the basis of sufficient and well-collected data. We will decide that either H_0 is rejected (thus H_a must be true) or is not rejected (thus there is no or not enough evidence to demonstrate that H_0 is false). [56]

		Population	
		H_0 is true	H_0 is not true
Sample (decision)	Do not reject H_0	No Error	Type 2 Error β
	Reject H_0	Type 1 Error α	No Error

Fig. 12.1: Types of errors in hypothesis testing [56]

8. 4 situations:
 - (a) **[true positive]** we do not reject H_0 when H_0 is true in the population [56]
 - (b) **[true negative]** we reject H_0 , and subsequently accept H_a , when H_a is true in the population [56]
 - (c) **[false positive]** we reject H_0 while in reality it is true, aka false positive or type 1 error. The probability of a type 1 error is associated with the level α . The α is used as a maximal allowable type 1 error for a decision rule. [56]
 - (d) **[false negative]** we do not reject H_0 , while in actuality H_a is true, aka type 2 error. The probability of a type 2 error is associated with the level β . The value β is used as a maximal allowable type 2 error. One minus the type 2 error ($1 - \beta$) is called the **power** of the binary decision rule. It indicates how likely the null hypothesis is rejected when the alternative hypothesis is true. [56]
9. it is easy to create a decision procedure that has a type 1 error probability equal to zero: if we simply never reject H_0 —in this case basically we state that there is never sufficient evidence to reject H_0 —we will never make a type 1 error. While this decision procedure does control the type 1 error, it is clearly not very useful, as the power of this decision rule is zero: we never accept the alternative hypothesis when it is true. [56]
10. the procedure of hypothesis testing aims to be less conservative (e.g., it will reject the null hypothesis sometimes but not too often when it would be true). [56]

11. **One tailed test:** Test statistics falls into one specified tail of its sampling distribution [57]
12. **Two tailed test:** Test statistics can fall into either tail of its sampling distribution [57]
13. Steps to Significance Testing:
 - (a) Define H_0 and H_a [57]
 - (b) Identify test, α , find critical value, test statistics [57]
 - (c) Construct acceptance/rejection regions [57]
 - (d) Calculate test statistics [57]
 - i. Critical value approach: Determine critical region [57]
 - ii. p-value approach: Calculate p-value [57]
 - (e) Retain or reject the hypothesis [57]
14. **Choosing a Statistical Test:**

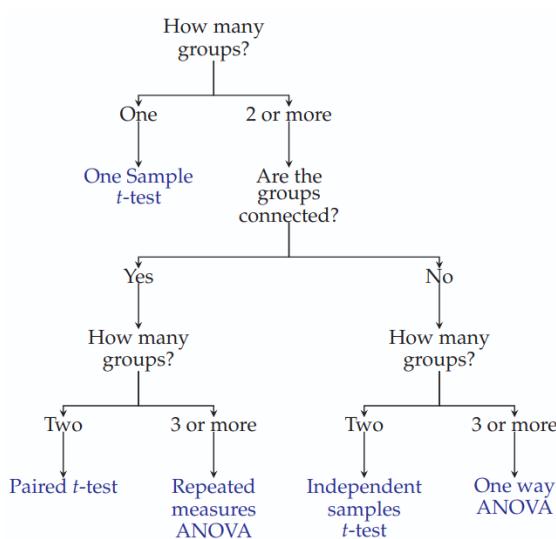
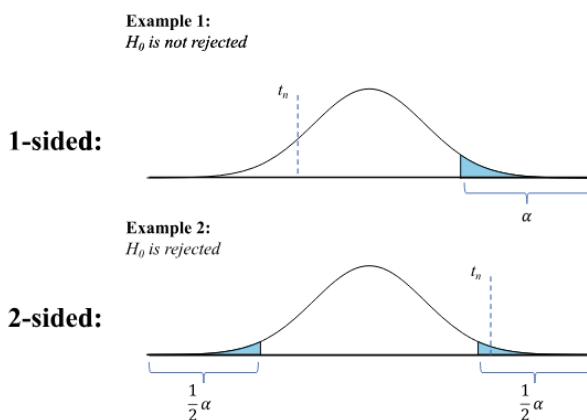


Fig. 12.2: Choosing a Statistical Test: Decision Tree [57]

- (a) Categorical Data: Use Chi Square [57]
- (b) Sample size (n):
 - i. $n < 30$ and Population Variance is unknown - t-test [57]
 - ii. $n < 30$ and Population Variance is known - z-test [57]
 - iii. $n > 30$ - z-test or t-test [57]
15. The z-test heavily depends on asymptotic theory and therefore requires **large sample sizes**. [56]
16. the t-test can be used for small sample sizes but under the strict assumption of having collected data from a **normal distribution**. [56]
17. Alternative approaches for the z-test and t-test have been developed that require fewer or no assumptions. These tests are referred to as **non-parametric tests**. [56]
18. The term **heteroskedasticity** refers to differences in variation or variability. In hypothesis testing this is often translated to a hypothesis on the variances or standard deviations from two different populations, as we described for the two samples t-test. Under the assumption of normality the most efficient test statistic (F-test) is based on a ratio of the two sample variances, but under non-normal data an alternative approach (Levene's test) has been suggested. [56]
19. The common practice of testing the (two-sided) null hypothesis has a few **drawbacks**. [56]

- (a) For (extremely) large samples we will almost always reject the null hypothesis. This might not be desirable, as rejecting the null hypothesis in such cases does not actually imply that the (e.g.,) difference in means of interest is indeed large. [56]
- (b) Not rejecting the null hypothesis $H_0 : \mu(f) = \mu_0$ is no proof that the null hypothesis is true. It is very easy not to reject the null hypothesis: you just need to collect as little information as possible. If we would collect only a few observations the confidence interval would be very wide and the value μ_0 is likely to fall inside this wide confidence interval, but this does not guarantee that $\mu(f) = \mu_0$ or even close to it. [56]

One way of dealing with these two issues on hypothesis testing is to reduce the significance level α to a much lower value than the commonly used $\alpha = 0.05$. We may use $\alpha = 0.001$ or even $\alpha = 0.0001$ to make the hypothesis statements more confident than just 95%. [56]



The difference between one-sided and two-sided tests. In both cases H_0 is not rejected if the observed test statistic t_n falls outside of the rejection region α . However, for a one-sided test (top) the rejection region is located on one tail of the distribution (the blue area denoted α). Hence, only if $t_n > z_{1-\alpha}$ (in this case), and sufficiently large to fall within the rejection region, is H_0 rejected. For a two-sided test the rejection region is split over the two tails of the distribution: hence H_0 can be rejected if t_n is either sufficiently small or sufficiently large. Finally, note that if $t_n > z_{1-\alpha}$ —if this is the direction of the one-sided test—a one-sided test might reject H_0 , whereas the same t_n might not lead to a rejection in the two-sided case, because the critical value for the two-sided test is larger than for the one-sided test [56]

12.2.1. The One-Sided z -Test for a Single Mean ($H_0 : \mu(f) \leq \mu_0$)

1. hypothesis on the population mean: $H_0 : \mu(f) \leq \mu_0$ versus $H_a : \mu(f) > \mu_0$. [56]
2. we assume that the sample is large enough to be able to use the normal distribution function as an approximation to the distribution function of the statistic that we are using to make a decision about the null hypothesis [56]
3. Let's assume that we have collected a random sample Y_1, Y_2, \dots, Y_n from the population. We may estimate the population mean $\mu(f)$ with the sample average \bar{Y} . [56]
4. when \bar{Y} is smaller or equal to μ_0 the random variable \bar{Y} seems to be in line with the null hypothesis $H_0 : \mu(f) \leq \mu_0$. In other words, there is no evidence that the **null hypothesis is false**. [56]
5. Although the random variable does not suggest any conflict with the null hypothesis $H_0 : \mu(f) \leq \mu_0$, it **does not guarantee** that $\mu(f) \leq \mu_0$ either. [56]
6. If the population mean $\mu(f)$ were somewhat larger than μ_0 , it might not be completely unlikely to observe a sample average still below μ_0 due to the sampling (a type 2 error). [56]
7. When \bar{Y} is larger than μ_0 , we might start to believe that the null hypothesis is incorrect. However, when \bar{Y} is just a little higher than μ_0 this might not be very unlikely either, even when $\mu(f) \leq \mu_0$.

[56]

8. For instance, for any symmetric f at $\mu(f) = \mu_0$, the probability that \bar{Y} is larger than μ_0 is equal to 0.5. Only when the sample average \bar{Y} is substantially larger—thus when there is sufficient evidence—than \bar{Y} would we start to indicate that the null hypothesis $H_0 : \mu(f) \leq \mu_0$ is unlikely to be true (although a type 1 error could be made here). [56]

9. we want to find a criterion for the average \bar{Y} such that the average can only be larger than this criterion with a probability that is at most equal to α when the null hypothesis is true. If we assume that this criterion is equal to $\mu_0 + \delta$, with $\delta > 0$, then the probability that \bar{Y} is larger than $\mu_0 + \delta$ is given by [56]

$$P(\bar{Y} > \mu_0 + \delta) = P\left(\frac{(\bar{Y} - \mu(f))}{\sigma(f)/\sqrt{n}} > \frac{\mu_0 - \mu(f) + \delta}{\sigma/\sqrt{n}}\right) \approx 1 - \Phi\left(\frac{\mu_0 - \mu(f) + \delta}{\sigma(f)/\sqrt{n}}\right) \quad [56]$$

where Φ is the standard normal PDF. [56]

10. Under the null hypothesis $H_0 : \mu(f) \leq \mu_0$ this probability is the type 1 error and it is maximized when $\mu(f) = \mu_0$. [56]

11. if we deliberately set $\mu(f) = \mu_0$ to maximize the type 1 error, the probability $P(\bar{Y} > \mu_0 + \delta)$ is given by $1 - \Phi(\delta\sqrt{n}/\sigma)$. [56]

12. When we choose δ equal to $\delta = \frac{z_{1-\alpha}\sigma(f)}{\sqrt{n}}$, the probability becomes $P(\bar{Y} > \mu_0 + \delta) \approx \alpha$. When $\bar{Y} > \mu_0 + \frac{z_{1-\alpha}\sigma(f)}{\sqrt{n}}$ the probability of rejecting the null hypothesis is at most α —and hence we have defined sufficient evidence by the criterion $\mu_0 + \frac{z_{1-\alpha}\sigma(f)}{\sqrt{n}}$ for the null hypothesis $H_0 : \mu(f) \leq \mu_0$ using the statistic \bar{Y} . [56]

13. The null hypothesis could still be true, but that it is just bad luck to have observed such an unlikely large average under the null hypothesis. However, we know that this probability of having bad luck is less than or equal to α and therefore we accept making this potential type 1 error. [56]

14. we cannot use the criterion $\bar{Y} > \mu_0 + \frac{z_{1-\alpha}\sigma(f)}{\sqrt{n}}$ directly, as it depends on the population standard deviation $\sigma(f)$, which is generally not known. We could estimate $\sigma(f)$ using sample standard deviation $S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{Y})^2}$ [56]

15. If the sample size is large enough, the probability $P\left(\bar{Y} > \mu_0 + \frac{z_{1-\alpha}S}{\sqrt{n}}\right)$ would still be close to α . When the sample size is large enough, we may reject the null hypothesis in practice when $\bar{Y} > \mu_0 + \frac{z_{1-\alpha}S}{\sqrt{n}}$ or in other words when the **asymptotic test statistic** $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ is larger than $z_{1-\alpha}$. [56]

16. For large sample sizes n , the null hypothesis $H_0 : \mu(f) \leq \mu_0$ is tested with test statistic $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ and the null hypothesis is rejected with significance level α when the test statistic is larger than the critical value $z_{1-\alpha}$. [56]

17. The null hypothesis is not rejected when $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}} \leq z_{1-\alpha}$, since there would not be enough evidence to reject the null hypothesis with significance level α (i.e., the observed result is not unlikely enough under H_0). This does not mean that the null hypothesis is true. [56]

18. the asymptotic approach for testing $H_0 : \mu(f) \leq \mu_0$ will work for most population densities f_X whenever the sample size is large enough. [56]

19. The p-value is the probability that the observed test statistic t_n and more extreme observations would occur under the null hypothesis. [56]
 - (a) It measures how surprising the observed data is if the null hypothesis H_0 were true. [7]
 - (b) A small p-value means the observed test statistic is very unlikely under H_0 , giving evidence against H_0 [7]
20. Calculating p-value:
 - (a) test statistic: $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ [7]
 - (b) observed test statistic: $t_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ [7]
 - (c) Convert test statistic to a probability under H_0 [7]
 - i. If H_0 is true, then T_n approximately follows a standard normal distribution (for large n). [7]
 - ii. The p-value is the tail probability of seeing t_{obs} or something more extreme. [7]
 - (d) Depending on the test type:
 - i. Right-tailed test ($H_0 : \mu \leq \mu_0$): $p = P(Z \geq t_{obs}) = 1 - \Phi(t_{obs})$ [7]
 - ii. Left-tailed test ($H_0 : \mu \geq \mu_0$): $p = P(Z \leq t_{obs}) = \Phi(t_{obs})$ [7]
 - iii. Two-tailed test ($H_0 : \mu = \mu_0$): $p = 2 \cdot \min(\Phi(t_{obs}), 1 - \Phi(t_{obs}))$ [7]

Here Φ is the cumulative distribution function (CDF) of the standard normal. [7]

12.2.2. The Two-Sided z-Test for a Single Mean ($H_0 : \mu(f) = \mu_0$)

1. The null hypothesis is formulated as $H_0 : \mu(f) = \mu_0$ and the alternative hypothesis is $H_a : \mu(f) \neq \mu_0$. [56]
2. We would reject the null hypothesis when either $T_n = \frac{\bar{Y} - \mu_0}{S/\sqrt{n}}$ is large positive or large negative. [56]
3. If we still keep our significance level at α , we will reject the null hypothesis $H_0 : \mu(f) = \mu_0$ when $\frac{\bar{Y} - \mu_0}{S/\sqrt{n}} > z_{1-\alpha/2}$, with $z_{1-\alpha/2}$ the $1 - \alpha/2$ quantile of the standard normal distribution (when using a normal approximation), or when $\frac{\bar{Y} - \mu_0}{S/\sqrt{n}} < -z_{1-\alpha/2}$. [56]
4. we would reject when $\frac{|\bar{Y} - \mu_0|}{S/\sqrt{n}} > z_{1-\alpha/2}$, with $|x|$ the absolute value of x . [56]
5. we reject this hypothesis when the test statistic is sufficiently small or sufficiently large. [56]
6. To control type 1 error, we therefore split up our rejection region in two: this is why we work with $z_{1-\alpha/2}$. [56]
7. two-sided tests, while used very often in practice, are less useful when sample sizes are very large: with a very large n , the standard error of a test statistic will become small, and we eventually will reject the null hypothesis in most cases. [56]
8. confidence interval contains the statistic we just discussed for hypothesis testing. In the case that μ_0 is not contained in the confidence interval, either $\frac{\bar{Y} - \mu_0}{S/\sqrt{n}} > z_{1-\alpha/2}$ or $\frac{\bar{Y} - \mu_0}{S/\sqrt{n}} < -z_{1-\alpha/2}$ would have occurred. Thus the null hypothesis $H_0 : \mu(f) = \mu_0$ would be rejected if μ_0 is not contained in $\left[\bar{Y} - \frac{z_{1-\alpha/2} S}{\sqrt{n}}, \bar{Y} + \frac{z_{1-\alpha/2} S}{\sqrt{n}} \right]$. Thus confidence intervals relate directly to two-sided hypothesis tests. [56]

9. If the null hypothesis is included in the confidence interval, we do not have sufficient evidence to reject it. If the confidence interval lies fully outside of the null hypothesis, there is sufficient evidence to reject the null hypothesis. Theoretically (and asymptotically) using confidence intervals for testing will lead to the same type 1 error probabilities. [56]
10. The bootstrap approach is an analytical approach to obtain confidence intervals for certain statistics may be an alternative to generate confidence intervals (instead of using asymptotic theory). [56]

12.2.3. Non-Parametric Test: Mann–Whitney U Test for Two Independent Samples ($H_0 : P(Y_1 > Y_2) = P(Y_1 < Y_2)$)

1. It tests the null hypothesis $H_0 : P(Y_1 > Y_2) = P(Y_1 < Y_2)$, with Y_1 a random draw from the first population and Y_2 a random draw from the second population. The null hypothesis is equivalent to $H_0 : P(Y_1 > Y_2) = 0.5$. [56]
2. If the null hypothesis is false, it is more likely to observe larger values in one of the populations with respect to the other population. Thus one population is **stochastically greater** than the other population. [56]
3. If we now assume that the distribution function F_2 for the second population is given by $F_2(y) = F_1(y + \delta)$, with F_1 the distribution function of the first population and δ a (shift) parameter, the null hypothesis $H_0 : P(Y_1 > Y_2) = P(Y_1 < Y_2)$ implies that the medians of the two populations must be equal (i.e., $\delta = 0$). [56]
4. Only in this somewhat restrictive formulation of population distributions, the Mann–Whitney U test investigates whether the two populations have equal medians. [56]

5. Mann–Whitney U test statistic:
$$U = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} [1_{(Y_{2,j} < Y_{1,i})} + 0.5 \cdot 1_{(Y_{1,i} = Y_{2,j})}]$$
 [56]

with 1_A the indicator function equal to 1 if A is true and zero otherwise. [56]

6. It represents the number of pairs $(Y_{1,i}, Y_{2,j})$ of observations from the two populations for which the first population provides a larger value than the second population. If there are ties (i.e., $Y_{1,i} = Y_{2,j}$), we cannot determine which population provides the larger value; thus this pair only contributes half to the total number of observations for which the first population is larger than the second population. [56]
7. If the variable Y_h is continuous, we should not observe any ties if we use enough decimal places in the recording of the values. [56]
8. The Mann–Whitney U test only makes use of the **ordering** of the observations. The total number of **pairs** for which the comparison between the two populations is made is equal to $n_1 n_2$. [56]
9. Each observation of the first sample is compared to each observation in the second sample. Thus the statistic $\frac{U}{n_1 n_2}$ is an estimator of the probability $P(Y_1 > Y_2)$. When $\frac{U}{n_1 n_2}$ is away from 0.5 or in other words, when U is away from $0.5 n_1 n_2$, the null hypothesis $H_0 : P(Y_1 > Y_2) = 0.5$ is rejected and one population is considered stochastically greater than the other population. [56]
10. To determine whether the U statistic is away from $0.5 n_1 n_2$, we will make use of **asymptotic theory**, as we did with the z -test for means. If the sample sizes n_1 and n_2 are large enough, the Mann–Whitney U statistic is approximately normally distributed with mean μ_U and variance σ_U^2 . [56]
 - (a) Under the null hypothesis $H_0 : P(Y_1 > Y_2) = 0.5$, mean $\mu_U = 0.5 n_1 n_2$. [56]
 - (b) The variance is then equal to $\sigma_U^2 = \frac{n_1 n_2 (n_1 + n_2 + 1)}{12}$, but only when there are **no ties**. If there are ties, a correction to this variance is required to make the variance smaller. The ties reduce the variability in U . [56]

- (c) To test the null hypothesis $H_0 : P(Y_1 > Y_2) = 0.5$ we calculate the standardized value $Z = \frac{U - 0.5n_1 n_2}{\sqrt{n_1 n_2 (n_1 + n_2 + 1)/12}}$ and compare this with the quantiles $z_{\alpha/2}$ and $z_{1-\alpha/2}$ of the standard normal distribution. If $|Z| > z_{1-\alpha/2}$ we reject the null hypothesis. [56]

12.2.4. The Sign Test for Two Related Samples ($H_0 : P(Y_1 > Y_2) = 0.5$)

- For two related datasets we observe the pairs of data $(Y_{1,i}, Y_{2,i})$ for the units $i = 1, 2, \dots, n$. A relevant question for paired data is whether one component is stochastically greater than the other component: $H_0 : P(Y_1 > Y_2) = 0.5$. [56]

- The sign test is mathematically given by $S = \sum_{i=1}^n 1_{(Y_{2,i} < Y_{1,i})}$, with 1_A the indicator variable that is equal to one if A is true and zero otherwise. It represents the number of units for which the first observation is larger than the second observation. [56]
- Under the null hypothesis (and when there are no ties $Y_{1,i} = Y_{2,i}$) the statistic follows a binomial distribution with proportion $p = 0.5$ and n trials (i.e., comparisons between Y_1 and Y_2). Thus the binomial distribution function can be used to determine when S becomes too large or too small. [56]
- The sign test needs a small adjustment when **ties** ($Y_{1,i} = Y_{2,i}$) are present in the data. In the case of ties we cannot judge which component of the pair is larger than the other component. This means that these **pairs cannot be used**. These pairs should be **excluded** from the calculations. The test statistic remains unchanged, but the number of trials n should be reduced by the number of ties. [56]
- The **disadvantage** of the sign test is that it does not consider the size of the distances between the two components. [56]

12.2.5. Wilcoxon's Signed Rank Test for Two Related Samples ($H_0 : m_D = 0$)

- The **advantage** of the sign test is that we did not have to assume anything about the distribution function of the pairs $(Y_{1,i}, Y_{2,i})$. Under the null hypothesis we could determine the distribution function of our test statistic. Therefore, we could determine when the test statistic would result in unlikely results if the null hypothesis is true. [56]
- The Wilcoxon signed rank test takes into account these differences for the two groups of pairs with $Y_{1,i} > Y_{2,i}$ and $Y_{1,i} < Y_{2,i}$. [56]
- steps:
 - Calculate for each pair the difference $D_i = Y_{1,i} - Y_{2,i}$. [56]
 - Create for each pair an indicator $1_{(Y_{1,i} > Y_{2,i})}$. [56]
 - Calculate for each pair the rank R_i of the absolute differences $|D_i|$. [56]
 - Calculate the sum of ranks for the positive differences: $W^+ = \sum_{i=1}^n 1_{(Y_{1,i} > Y_{2,i})}$. [56]
- If the distribution function of the positive differences D_i is equivalent to the distribution function of the negative differences D_i , we would expect that the **average** rank for the positive differences is equal to the **average** rank of the negative differences. [56]
- This translates to a null hypothesis on the median of the difference: $H_0 : m_D = 0$, with m_D the median of the distribution of all differences D_i . If the average rank for the positive differences is really different from the average rank of the negative differences, the median of all differences can no longer be zero. [56]
- we use **asymptotic theory**. If the sample sizes are large enough and the null hypothesis is true, the sum of the ranks for the positive differences W^+ is approximately normal with mean μ_W and variance σ_W^2 .

The mean and variance are determined by $\mu_W = \frac{n(n+1)}{4}$ and the variance is $\sigma_W^2 = \frac{n(n+1)(2n+1)}{24}$, respectively. [56]

7. the mean is just half of the sum of all ranks, as the sum of all ranks is equal to $\frac{n(n+1)}{2}$. [56]
8. we can standardize the Wilcoxon signed rank test to $Z = \frac{W^+ - \mu_W}{\sigma_W}$ and compare this standardized statistic with the quantiles of the standard normal distribution to reject the null hypothesis $H_0 : m_D = 0$ or not. [56]

12.2.6. Levene's Test for Equal Variation ($H_0 : \sigma_1^2 = \sigma_2^2 = \dots = \sigma_k^2$)

1. For non-normal data the variability or variation around the mean or median is not fully described by the standard deviation alone, as is the case for normal data. [56]
2. Here the distance of each observation from its group mean or median is calculated first. Thus for sample h we calculate distances $Z_{h,i} = |Y_{h,i} - m_h|$, with $i = 1, 2, \dots, n_h$. The value m_h represents some kind of location of the sample h , like the average or median. [56]
3. If the distribution functions F_1 and F_2 for the two populations ($h = 1, 2$) are the same, except for a difference in the population mean or median, the distances $Z_{1,i}$ and $Z_{2,i}$ may be viewed as two samples from one and the same population of distances, i.e., the distances $Z_{1,i}$ and $Z_{2,i}$ would be drawn from the same distribution of distances. [56]
4. To investigate if these samples of distances come from one distribution, we may study a difference in means. If the samples on distances $Z_{1,i}$ and $Z_{2,i}$ suggest that they have different means, they do not come from the same distribution of distances and there must exist differences between the two distribution functions F_1 and F_2 that are not induced by a shift in mean or median alone. This would imply that a difference in variation in the two populations $h = 1$ and $h = 2$ is warranted, since differences in the means between the two samples of distances $Z_{1,i}$ and $Z_{2,i}$ indicate that one set of distances are on average larger than the other set of distances. [56]
5. Levene's test statistic just uses the t-test with equal variances on the two independent samples of distances $Z_{1,i}$ and $Z_{2,i}$ to determine if the means of the population of distances that they may represent are different. A **two-sided** t-test is used, since there is no preference in knowing which variable X or Y has a higher or lower variability. [56]
6. Brown & Forsythe demonstrated that the use of the median for m_h gives a somewhat more robust statistic for many different distribution functions F_1 and F_2 than Levene's choice of means. Brown & Forsythe's version of Levene's test is often recommended over Levene's test. [56]
7. What Levene's test does [7]
 - (a) It tests whether two (or more) populations have equal variances. [7]
 - (b) It works by transforming the data into absolute deviations from the mean (or median, in the Brown–Forsythe version), and then running an ANOVA or t-test on those deviations. [7]
8. **Null hypothesis** $H_0 : \sigma_1^2 = \sigma_2^2 = \dots = \sigma_k^2$ (All group/population variances are equal.) [7]
9. **Alternative hypothesis:** At least one variance is different. [7]
10. **Decision Rule** [7]
 - (a) Compute the Levene (or Brown–Forsythe) test statistic. [7]
 - (b) Compare its p-value with α (usually 0.05). [7]
 - (c) If $p \leq \alpha$: Reject $H_0 \rightarrow$ conclude that the variances are not equal. [7]
 - (d) If $p > \alpha$: Fail to reject $H_0 \rightarrow$ no evidence against equal variances. [7]

12.2.7. The χ^2 Test for Categorical Variables

$$(H_0 : P(X = x, Y = y) = P(X = x)P(Y = y))$$

1. the null hypothesis is formulated as $H_0 : P(X = x, Y = y) = P(X = x)P(Y = y)$ against the alternative hypothesis of $H_a : P(X = x, Y = y) \neq P(X = x)P(Y = y)$. [56]
2. The collected data are typically presented by a contingency table and Pearson's chi-square statistic is a proper statistic for testing the independence between X and Y . [56]
3. The statistic was designed to compare the observed cell counts for the contingency table with the expected cell counts for the contingency table under independence. [56]
4. Using K for the number of rows in the contingency table, M for the number of columns, Pearson's chi-square statistic is given by: [56]

$$\chi_P^2 = \sum_{x=1}^K \sum_{y=1}^M \frac{(N_{xy} - (N_x \cdot N_y / N))^2}{N_x \cdot N_y / N} \quad (\text{Yate's "continuity" correction}) \quad [56]$$

with N_{xy} the observed count in cell (x,y) of the contingency table, N_x the row total for $X = x$, N_y the column total for $Y = y$, and N the total count in the contingency table. [56]

5. The product $\frac{N_x \cdot N_y}{N}$ is the expected count for cell (x,y) under independence. Under the null hypothesis of independence, Pearson's chi-square statistic follows the χ^2 -distribution with $(K-1)(M-1)$ degrees of freedom. [56]
6. the p-value for testing the null hypothesis is found by calculating $P(\chi^2 > x_P^2)$, the probability that a chi-square distributed random variable χ^2 exceeds the observed value x_P^2 for χ^2 . [56]

12.2.8. Tests for Outliers

1. An outlier is an observation that seems to deviate from the other observations such that it arouses suspicion that some unintended mechanism has interfered with the random sampling. [56]
2. This means that the observation is not drawn from the same population distribution as the other observations. [56]
3. Outliers are a nuisance for many calculations since they may highly affect the estimator or test statistic, and therefore influence the conclusions strongly. To limit their effect, one may be inclined to remove the outliers and continue with the remaining observations. [56]
4. Outliers should only be removed when a reason or cause has been established; in all other situations the outlier should not be removed. [56]
5. Outlier detection is not an easy task.
 - (a) it is difficult to distinguish whether the outlier observation is truly caused by some kind of unintended mechanism or whether it belongs to a population distribution that is just different than what we anticipated. [56]
 - (b) statistical methods that were developed for checking only one outlier observation are diminished in their detection capacity if multiple outliers are present. This is called the **masking effect** and relates to the type 2 error rate in hypothesis testing, since a real outlier is not being detected. [56]
 - (c) outlier detection requires certain assumptions on the underlying population distribution. Without such assumptions we can never make confident statements like we try to do with hypothesis testing. If we are not willing to make such assumption, we all have to agree to the same definition that an observation is called an outlier whenever the observation satisfies some kind of predefined criterion. [56]
6. The Grubbs test was developed for normally distributed data and it can be formulated in terms of hypothesis testing. [56]

7. Tukey's method has been implemented in many software packages for visualization of the data with a box plot. John Tukey simply provided a definition for outliers, in the sense that an observation is called an outlier when the observation is further away from the median value than a predefined distance. In the box plot they are typically indicated by dots or stars. [56]
8. Fixing the criteria for an outlier observation does not always imply that the occurrence of such observations in the sample will be unlikely. For normal distribution functions, Tukey's criteria for outlier observations is unlikely if no outliers are present, but using Tukey's criteria for other distribution functions should be implemented with care [56]

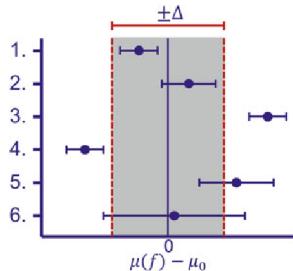
12.2.8.1. Tukey's Method for Outliers (Using IQR)

1. Tukey suggested that an observation is an outlier whenever the observation Y_k is 1.5 times the interquartile range below the first quartile or 1.5 times the interquartile range above the third quartile. [56]
2. An observation is called a lower-tail outlier when $Y_k < Q_1 - 1.5 \cdot \text{IQR}$ and it is called an upper-tail outlier when $Y_k > Q_3 + 1.5 \cdot \text{IQR}$, with $\text{IQR} = Q_3 - Q_1$ the interquartile range. [56]
3. The first observation that may satisfy the lower tail criterion would be the minimum, while for the upper tail it is the maximum value. [56]
4. Tukey's method is often referred to as a method that **does not depend on any assumptions** of the population distribution, since it is only based on quartiles. However, the shape of the population distribution does determine how likely an observation will satisfy the criterion and thus will be declared an outlier [56]
5. If we consider the full population, we may substitute the expected or true values for Q_1 and Q_3 in the criteria and determine the proportion of the population that would satisfy the lower and upper criteria. The lower criterion in the population is given by $F^{-1}(0.25) - 1.5(F^{-1}(0.75) - F^{-1}(0.25))$ and the upper criterion is given by $F^{-1}(0.75) + 1.5(F^{-1}(0.75) - F^{-1}(0.25))$. [56]

12.2.9. Equivalence Testing ($H_0 : |\mu(f) - \mu_0| > \Delta$)

1. Equivalence testing would first formulate a margin Δ that would be used to form a range of values around the value μ_0 that we would see as equivalent settings. [56]
2. The null hypothesis is then formulated as $H_0 : |\mu(f) - \mu_0| > \Delta$ against the alternative hypothesis $H_a : |\mu(f) - \mu_0| \leq \Delta$. The null hypothesis specifies non-equivalence: the difference between $\mu(f)$ and μ_0 is larger than Δ . [56]
3. If the null hypothesis is rejected, we have demonstrated with sufficient confidence $(1 - \alpha)$ that the true population mean $\mu(f)$ is equivalent to the value μ_0 . The level Δ is called the **equivalence margin**. [56]
4. Using confidence intervals, a test for equivalence is reasonably simple. We just need to calculate a $1 - 2\alpha$ confidence interval on $\mu(f) - \mu_0$ and then compare it to the interval $[-\Delta, \Delta]$. If the confidence interval falls fully in the interval $[-\Delta, \Delta]$, the null hypothesis $H_0 : |\mu(f) - \mu_0| > \Delta$ is rejected at significance level α . [56]
5. To test at significance level α , we need to calculate a $1 - 2\alpha$ confidence interval. The reason is that the equivalence test is actually the application of two one-sided tests: one test for showing that $\mu(f) < \mu_0 + \Delta$ and another test for showing that $\mu(f) > \mu_0 - \Delta$. [56]
6. Each test is done at significance level α . This is similar to having the $1 - 2\alpha$ confidence interval on $\mu(f) - \mu_0$ fall inside $[-\Delta, \Delta]$. [56]
7. The one-sided equivalence tests are referred to as **non-inferiority tests**. [56]
8. In the case of equivalence testing, where the equivalence margin is given by $\Delta > 0$, we can formulate the following hypotheses: [56]

Non-inferiority	$H_0 : \mu_1 - \mu_2 \leq -\Delta$	$H_a : \mu_1 - \mu_2 > -\Delta$
Non-inferiority	$H_0 : \mu_1 - \mu_2 \geq \Delta$	$H_a : \mu_1 - \mu_2 < \Delta$
Equivalence	$H_0 : \mu_1 - \mu_2 \geq \Delta$	$H_a : \mu_1 - \mu_2 < \Delta$



Comparison between equivalence testing and traditional hypothesis testing. The vertical dotted lines represent the interval $[-\Delta, \Delta]$ and the vertical solid line represents the difference $\mu(f) - \mu_0 = 0$. The horizontal x-axis represents the difference $\mu(f) - \mu_0$ and each dot in the figure represents an estimate of this difference. The horizontal lines through the dots represent the $1 - 2\alpha$ confidence intervals. [56]

9. Equivalence testing $H_0 : |\mu(f) - \mu_0| > \Delta$ against $H_a : |\mu(f) - \mu_0| \leq \Delta$ is really different in concept from traditional hypothesis testing $H_0 : \mu(f) = \mu_0$ against $H_a : \mu(f) \neq \mu_0$. [56]
 - (a) The $1 - 2\alpha$ confidence interval fall completely within $[-\Delta, \Delta]$, which means that the null hypothesis $H_0 : |\mu(f) - \mu_0| > \Delta$ is being rejected. It is not completely clear whether $H_0 : \mu(f) = \mu_0$ is being rejected, since this can only be established if the $1 - \alpha$ confidence interval has been applied. If we assume that this $1 - \alpha$ interval would fit within $[-\Delta, 0]$, the null hypothesis $H_0 : \mu(f) = \mu_0$ would also be rejected. This is a setting that we described earlier for large datasets, where the traditional hypothesis can be rejected for irrelevant differences. [56]
 - (b) The null hypothesis $H_0 : |\mu(f) - \mu_0| > \Delta$ for equivalence is being rejected, indicating that the population mean $\mu(f)$ is equivalent to μ_0 . The traditional hypothesis $H_0 : \mu(f) = \mu_0$ is not being rejected, which implies that there is not enough evidence to believe that the population mean $\mu(f)$ is different from μ_0 . This means that both conclusions seem to coincide, although this does not imply that $\mu(f) = \mu_0$. [56]
 - (c) The traditional null hypothesis is being rejected (assuming that the 95% confidence interval would not contain zero) and equivalence cannot be claimed. Thus both approaches seem to have similar conclusions: $\mu(f)$ is different and not equivalent to μ_0 . [56]
 - (d) This is the same as in the previous setting, although the results lie on the other side of the vertical line $\mu(f) = \mu_0$. [56]
 - (e) The fact that one side of the confidence interval is contained in the interval $[-\Delta, \Delta]$ does not change the conclusions. [56]
 - (f) Here we cannot demonstrate equivalence, but we cannot reject the traditional null hypothesis either. Thus on the one hand there is not enough evidence to reject $\mu(f) = \mu_0$, but there is not enough evidence either to reject that $|\mu(f) - \mu_0| > \Delta$. This seems contradictory, but it is probably an issue of a lack of information, since the 90% confidence interval is too wide. [56]
10. The non-inferiority hypotheses look somewhat similar to the one-sided hypotheses, but they are not. In the traditional hypothesis testing format we would investigate the null hypothesis $H_0 : \mu_{New} \leq \mu_{Old}$ against $H_a : \mu_{New} > \mu_{Old}$, assuming that a higher mean is a better clinical outcome. [56]
11. The collected data (typically through a randomized controlled clinical trial) would then have to demonstrate that the null hypothesis is rejected. Thus the new treatment must demonstrate **superiority**. [56]
12. Often new treatments are not (much) better in their clinical outcome, but they bring a secondary benefit. For instance, the new treatment is much less invasive. Superiority is then difficult to demonstrate. Instead, we would like to show that the new treatment is not inferior to the old treatment. Thus, the data must demonstrate that the new treatment is not too much worse than the old treatment: $H_a : \mu_{New} > \mu_{Old} - \Delta$, with $\Delta > 0$ the **non-inferiority margin**. Thus, the null hypothesis becomes $H_0 : \mu_{New} \leq \mu_{Old} - \Delta$. If this null hypothesis cannot be rejected, the new treatment is assumed inferior to the old treatment, since non-inferiority could not be proven. [56]

12.2.10. Bayes' Law for Competing Hypotheses

1. Suppose we are trying to learn about the value of a parameter of interest θ , and suppose we have a number of specific hypotheses regarding its value, e.g., "Hypothesis A: $\theta < 0$ " (although we could effectively make any statement we want regarding the parameter values of interest). Let θ_i refer to the first hypothesis regarding θ (i.e., Hypothesis A). [56]
2. If we have k such hypotheses regarding θ (thus we have $\theta_1, \dots, \theta_k$) we can now compute the probability of each of these hypothesis after observing some data by applying Bayes' rule: [56]

$$P(\theta_i|D) = \frac{P(D|\theta_i)P(\theta_i)}{\sum_{j=1}^k P(D|\theta_j)P(\theta_j)}$$

IV

MATHEMATICS: LINEAR ALGEBRA

CHAPTER 13

GROUP THEORY

13.1. Groups

1. **Definition:** Consider a set \mathcal{G} and an (inner) operation $\otimes : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ group defined on G . Then $G := (\mathcal{G}, \otimes)$ is called a group if the following hold: [49]
 - (a) Closure of \mathcal{G} under \otimes : $\forall x, y \in \mathcal{G} : x \otimes y \in \mathcal{G}$ [49]
 - (b) Associativity: $\forall x, y, z \in \mathcal{G} : (x \otimes y) \otimes z = x \otimes (y \otimes z)$ [49]
 - (c) Neutral element: $\exists e \in \mathcal{G} \forall x \in \mathcal{G} : x \otimes e = x \text{ and } e \otimes x = x$ [49]
 - (d) Inverse element: $\forall x \in \mathcal{G} \exists y \in \mathcal{G} : x \otimes y = e \text{ and } y \otimes x = e$, where e is the neutral element. We often write x^{-1} to denote the inverse element of x . [49]
2. The inverse element is defined with respect to the operation \otimes and does not necessarily mean $\frac{1}{x}$. [49]
3. Group allows only inner operation \otimes , means the operands **must be** elements from \mathcal{G} .
4. Examples:
 - (a) $(\mathbb{N}_0, +)$ is **not** a group: Although $(\mathbb{N}_0, +)$ possesses a neutral element (0), the inverse elements are missing. $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ [49]
 - (b) (\mathbb{Z}, \cdot) is not a group: Although (\mathbb{Z}, \cdot) contains a neutral element (1), the inverse elements for any $z \in \mathbb{Z}, z \neq \pm 1$, are missing. [49]
 - (c) (\mathbb{R}, \cdot) is not a group since 0 does not possess an inverse element. [49]

13.2. Abelian Group

1. **Definition:** A group $G = (\mathcal{G}, \otimes)$ is called Abelian group if $\forall x, y \in \mathcal{G} : x \otimes y = y \otimes x$ (commutative) [49]
2. Examples:
 - (a) $(\mathbb{Z}, +)$ is an Abelian group [49]
 - (b) $(\mathbb{R} \setminus \{0\}, \cdot)$ is Abelian [49]
 - (c) $(\mathbb{R}^n, +), (\mathbb{Z}^n, +), n \in \mathbb{N}$ are Abelian if $+$ is defined component-wise:
$$(x_1, \dots, x_n) + (y_1, \dots, y_n) = (x_1 + y_1, \dots, x_n + y_n)$$
Then, $(x_1, \dots, x_n)^{-1} := (-x_1, \dots, -x_n)$ is the inverse element and $e = (0, \dots, 0)$ is the neutral element.
 - (d) $(\mathbb{R}^{m \times n}, +)$, the set of $m \times n$ -matrices is Abelian (with component-wise addition)

13.3. General Linear Group

1. **Definition:** The set of regular (invertible) matrices $A \in \mathbb{R}^{n \times n}$ is a group with respect to matrix multiplication and is called general linear group $GL(n, \mathbb{R})$.
2. However, since matrix multiplication is not commutative, the group is not Abelian.

CHAPTER 14

MATRICES

1. With $m, n \in \mathbb{N}$ a real-valued (m, n) matrix \mathbf{A} is an $m \cdot n$ -tuple of elements $a_{ij}, i = 1, \dots, m, j = 1, \dots, n$, which is ordered according to a rectangular scheme consisting of m rows and n columns:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad (a_{ij} \in \mathbb{R}) \quad [49]$$

2. $\mathbb{R}^{m \times n}$ is the set of all real-valued (m, n) -matrices. $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be equivalently represented as $\mathbf{a} \in \mathbb{R}^{mn}$ by stacking all n columns of the matrix into a long vector. [49]
3. They can be used to compactly represent systems of linear equations, but they also represent linear functions (linear mappings). [49]
4. matrix represents a linear mapping or a collection of vectors. [49]

14.1. Matrix Operations

14.1.1. Matrix Addition (+) [49]

The sum of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{m \times n}$ is defined as the element-wise sum:

$$\mathbf{A} + \mathbf{B} := \begin{bmatrix} a_{11} + b_{11} & \cdots & a_{1n} + b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & \cdots & a_{mn} + b_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad [49]$$

```

1 import numpy as np
2
3 m,n = 4,3
4
5 A = np.random.randint(-10, 10, size=(m,n))
6 B = np.random.randint(-10, 10, size=(m,n))
7
8 C = A + B
9
10 print("A:\n", A)
11 print("B:\n", B)
12 print("C:\n", C)

```

Python Snippet 14.1: Matrix Addition - numPy

14.1.2. Matrix Multiplication (AB OR $@$ OR \cdot) [49]

For matrices $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{n \times k}$, the elements c_{ij} of the product matrices. $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times k}$ are computed as:

$$c_{ij} = \sum_{l=1}^n a_{il} b_{lj} \quad i = 1, \dots, m \quad j = 1, \dots, k \quad [49]$$

```

1 import numpy as np
2
3 m,n,k = 4,3,5
4
5 A = np.random.randint(-10, 10, size=(m, n))
6 B = np.random.randint(-10, 10, size=(n, k))
7
8 C1 = A @ B
9 C2 = np.matmul(A, B)
10 C3 = np.dot(A, B)
11
12 print("A:\n", A)
13 print("B:\n", B)
14 print("C1:\n", C1)
15 print("C2:\n", C2)
16 print("C3:\n", C3)
17 print(np.all(C1 == C2), np.all(C1 == C3))

```

Python Snippet 14.2: Matrix Multiplication - numPy

1. To compute element c_{ij} we multiply the elements of the i th row of \mathbf{A} with the j th column of \mathbf{B} and sum them up. [49]
2. Matrices can only be multiplied if their “neighboring” dimensions match. [49]

$$\underbrace{\mathbf{A}}_{n \times k} \underbrace{\mathbf{B}}_{k \times m} = \underbrace{\mathbf{C}}_{n \times m}$$

The product \mathbf{BA} is not defined if $m \neq n$ since the neighboring dimensions do not match.
3. Matrix multiplication is **not** defined as an element-wise operation on matrix elements, i.e., $c_{ij} \neq a_{ij} b_{ij}$ (even if the size of \mathbf{A} , \mathbf{B} was chosen appropriately). [49]

14.1.2.1. Properties

1. matrix multiplication is **not** commutative: $\mathbf{AB} \neq \mathbf{BA}$ [49]
2. **Associativity:** $\forall \mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{n \times p}, \mathbf{C} \in \mathbb{R}^{p \times q}:$
 - (a) $\mathbf{ABC} = (\mathbf{AB}) \mathbf{C} = \mathbf{A} (\mathbf{BC})$ [49]
3. **Distributivity:** $\forall \mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{C}, \mathbf{D} \in \mathbb{R}^{n \times p}:$
 - (a) $(\mathbf{A} + \mathbf{B}) \mathbf{C} = \mathbf{AC} + \mathbf{BC}$ [49]
 - (b) $\mathbf{A} (\mathbf{C} + \mathbf{D}) = \mathbf{AC} + \mathbf{AD}$ [49]
4. Multiplication with the identity matrix: $\forall \mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}:$
 - (a) $\mathbf{I}_m \mathbf{A} = \mathbf{A} \mathbf{I}_n = \mathbf{A}$ $m \neq n \Rightarrow \mathbf{I}_m \neq \mathbf{I}_n$ [49]

14.1.3. Multiplication by a Scalar ($\lambda \mathbf{A}$)

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\lambda \in \mathbb{R}$. Then $\lambda \mathbf{A} = \mathbf{K} \in \mathbb{R}^{m \times n}$, $k_{ij} = \lambda a_{ij}$. [49]

1. λ scales each element of \mathbf{A} [49]

14.1.3.1. Properties

$\forall \lambda, \psi \in \mathbb{R}$:

$$1. \text{ Associativity: } (\lambda\psi)\mathbf{C} = \lambda(\psi\mathbf{C}) \quad \mathbf{C} \in \mathbb{R}^{m \times n} \quad [49]$$

$$2. \lambda(\mathbf{B}\mathbf{C}) = (\lambda\mathbf{B})\mathbf{C} = \mathbf{B}(\lambda\mathbf{C}) = (\mathbf{B}\mathbf{C})\lambda \quad \mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{C} \in \mathbb{R}^{n \times k} \quad [49]$$

Note that this allows us to move scalar values around. [49]

$$3. (\lambda\mathbf{C})^\top = \mathbf{C}^\top\lambda^\top = \mathbf{C}^\top\lambda = \lambda\mathbf{C}^\top \quad \lambda = \lambda^\top \forall \lambda \in \mathbb{R} \quad [49]$$

4. Distributivity:

$$(a) (\lambda + \psi)\mathbf{C} = \lambda\mathbf{C} + \psi\mathbf{C} \quad \mathbf{C} \in \mathbb{R}^{m \times n} \quad [49]$$

$$(b) \lambda(\mathbf{B} + \mathbf{C}) = \lambda\mathbf{B} + \lambda\mathbf{C} \quad \mathbf{B}, \mathbf{C} \in \mathbb{R}^{m \times n} \quad [49]$$

```

1 import numpy as np
2
3 # Define a matrix or vector
4 A = np.array([
5     [1, 2],
6     [3, 4]
7 ])
8
9 # Define a scalar
10 lambda = 5
11
12 # Multiply the matrix by the scalar
13 result = lambda * A
14
15 print("Original matrix A:\n", A)
16 print("Scalar lambda:", lambda)
17 print("Result of lambda * A:\n", result)

```

Python Snippet 14.3: Multiplication by a Scalar - numPy

14.1.4. Hadamard product (\odot OR $*$) [49]

element-wise multiplication: For $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{m \times n} \quad c_{ij} = a_{ij} b_{ij}$$

```

1 import numpy as np
2
3 m,n = 4,3
4
5 A = np.random.randint(-10, 10, size=(m,n))
6 B = np.random.randint(-10, 10, size=(m,n))
7
8 C = A * B
9
10 print("A:\n", A)
11 print("B:\n", B)
12 print("C:\n", C)

```

Python Snippet 14.4: Hadamard product - numPy

14.1.5. Matrix Transpose (A^\top) [49]

For $\mathbf{A} \in \mathbb{R}^{m \times n}$ the matrix $\mathbf{B} \in \mathbb{R}^{n \times m}$ with $b_{ij} = a_{ji}$ is called the transpose of \mathbf{A} . We write $\mathbf{B} = \mathbf{A}^\top$. [49]

```

1 import numpy as np
2
3 m,n = 4,3
4
5 A = np.random.randint(-10, 10, size=(m,n))
6
7 print("A:\n", A)
8 print("A transpose:\n", A.T)

```

Python Snippet 14.5: Matrix Inverse - numPy

1. In general, \mathbf{A}^\top can be obtained by writing the columns of \mathbf{A} as the rows of \mathbf{A}^\top

[49]

14.1.5.1. Properties

$$\begin{array}{lll} 1. (\mathbf{A}^\top)^\top = \mathbf{A} & [49] & 3. (\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top \\ 2. (\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top & [49] & 4. (\mathbf{A}^{-1})^\top = (\mathbf{A}^\top)^{-1} =: \mathbf{A}^{-\top} \end{array} \quad [49]$$

14.1.6. Matrix Inverse (A^{-1}) [49]

```

1 import numpy as np
2
3 n = 4
4
5 A = np.random.randint(-10, 10, size=(n,n)).astype(float)
6 A_inv = np.linalg.inv(A)
7
8 print("A:\n", A)
9 print("A_inv:\n", A_inv)
10 print(A @ A_inv)
11 print(np.allclose((A @ A_inv), np.eye(n, n)))

```

Python Snippet 14.6: Matrix Inverse - numPy

1. Consider a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Let matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ have the property that $\mathbf{AB} = \mathbf{I}_n = \mathbf{BA}$. \mathbf{B} is called the inverse of \mathbf{A} and denoted by \mathbf{A}^{-1} .

[49]

2.

Theorem 14.1 (Square Matrix: Invertible). For any square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ it holds that \mathbf{A} is invertible if and only if $\det(\mathbf{A}) \neq 0$.

[49]

3. Not every matrix \mathbf{A} possesses an inverse \mathbf{A}^{-1} .
4. Only square matrices might have inverse. Non-square matrices don't have inverse.
5. When the matrix inverse exists, it is unique.

[49]

$$6. \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in \mathbb{R}^{2 \times 2} \quad \text{and} \quad a_{11} a_{22} - a_{12} a_{21} \neq 0 \text{ (determinant of } A)$$

$$\Rightarrow \mathbf{A}^{-1} = \frac{1}{a_{11} a_{22} - a_{12} a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \quad [49]$$

14.1.6.1. Matrix Inverse using Gaussian Elimination

1. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, we need to find $\mathbf{X} = \mathbf{A}^{-1}$ such that $\mathbf{AX} = \mathbf{I}_n$ [49]
2. We can write this down as a set of simultaneous linear equations $\mathbf{AX} = \mathbf{I}_n$, where we solve for $\mathbf{X} = [\mathbf{x}_1 | \dots | \mathbf{x}_n]$. [49]
3. augmented matrix notation for a compact representation of this set of systems of linear equations: [49]
 $[\mathbf{A} | \mathbf{I}_n] \rightsquigarrow \dots \rightsquigarrow [\mathbf{I}_n | \mathbf{A}^{-1}]$ [49]

14.1.6.2. Moore-Penrose pseudo-inverse $((\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top)$

$$\mathbf{AX} = \mathbf{I}$$

$$\Leftrightarrow \mathbf{A}^\top \mathbf{AX} = \mathbf{A}^\top$$

$$\Leftrightarrow \mathbf{X} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top$$
 [49]

1. Disadvantages:
 - (a) requires many computations for the matrix-matrix product and computing the inverse of $\mathbf{A}^\top \mathbf{A}$. [49]
 - (b) for reasons of numerical precision it is generally not recommended [49]

14.1.6.3. Properties

- | | |
|---|---|
| 1. $\mathbf{AA}^{-1} = \mathbf{I} = \mathbf{A}^{-1}\mathbf{A}$ [49] | 4. $(\mathbf{A}^{-1})^\top = (\mathbf{A}^\top)^{-1} =: \mathbf{A}^{-\top}$ [49] |
| 2. $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ [49] | 5. $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$ |
| 3. $(\mathbf{A} + \mathbf{B})^{-1} \neq \mathbf{A}^{-1} + \mathbf{B}^{-1}$ [49] | |

14.2. Rank of a matrix

1.

Definition 14.1 (Rank of a matrix). The number of linearly independent columns of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ equals the number of linearly independent rows and is called the rank of \mathbf{A} and is denoted by $\text{rk}(\mathbf{A})$. [49]

14.2.1. Properties of rank

1. $\text{rk}(\mathbf{A}) = \text{rk}(\mathbf{A}^\top)$, i.e., the column rank equals the row rank. [49]
2. The columns of $\mathbf{A} \in \mathbb{R}^{m \times n}$ span a subspace $U \subseteq \mathbb{R}^m$ with $\dim(U) = \text{rk}(\mathbf{A})$. We call this subspace the **image or range**. A basis of U can be found by applying Gaussian elimination to \mathbf{A} to identify the **pivot columns**. [49]
3. The rows of $\mathbf{A} \in \mathbb{R}^{m \times n}$ span a subspace $W \subseteq \mathbb{R}^n$ with $\dim(W) = \text{rk}(\mathbf{A})$. A basis of W can be found by applying Gaussian elimination to \mathbf{A}^\top . [49]
4. For all $\mathbf{A} \in \mathbb{R}^{n \times n}$ it holds that \mathbf{A} is regular (invertible) if and only if $\text{rk}(\mathbf{A}) = n$. [49]
5. For all $\mathbf{A} \in \mathbb{R}^{m \times n}$ and all $\mathbf{b} \in \mathbb{R}^m$ it holds that the linear equation system $Ax = b$ can be solved if and only if $\text{rk}(\mathbf{A}) = \text{rk}(\mathbf{A}|\mathbf{b})$, where $\mathbf{A}|\mathbf{b}$ denotes the augmented system. [49]
6. For $\mathbf{A} \in \mathbb{R}^{m \times n}$ the subspace of solutions for $\mathbf{Ax} = \mathbf{0}$ possesses dimension $n - \text{rk}(\mathbf{A})$. We call this subspace the **kernel** or the **null space**. [49]
7. A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ has **full rank** if its rank equals the largest possible rank for a matrix of the same dimensions. This means that the rank of a full-rank matrix is the lesser of the number of rows and

columns, i.e., $\text{rk}(\mathbf{A}) = \min(m, n)$. A matrix is said to be **rank deficient** if it does not have full rank. [49]

```

1 import numpy as np
2
3 # Define your matrix
4 A = np.array([
5     [1, 2, 3],
6     [2, 4, 6],
7     [1, 0, 1]
8 ])
9
10 # Compute the rank
11 rank = np.linalg.matrix_rank(A)
12
13 print("Rank of matrix A:", rank)

```

Python Snippet 14.7: Rank of a matrix - numPy

14.3. Null Space and Column Space

1. Let us consider $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a linear mapping $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $\mathbf{x} \mapsto \mathbf{Ax}$. [49]
2. For $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]$, where \mathbf{a}_i are the columns of \mathbf{A} , we obtain [49]
$$\text{Im}(\Phi) = \{\mathbf{Ax} : \mathbf{x} \in \mathbb{R}^n\} = \left\{ \sum_{i=1}^n x_i \mathbf{a}_i : x_1, \dots, x_n \in \mathbb{R} \right\} = \text{span}[\mathbf{a}_1, \dots, \mathbf{a}_n] \subseteq \mathbb{R}^m \quad [49]$$
i.e., the image is the span of the columns of \mathbf{A} , also called the **column space**. Therefore, the column space (image) is a subspace of \mathbb{R}^m , where m is the “height” of the matrix. [49]
3. $\text{rk}(\mathbf{A}) = \dim(\text{Im}(\Phi))$ [49]
4. The kernel/null space $\ker(\Phi)$ is the general solution to the homogeneous system of linear equations $\mathbf{Ax} = \mathbf{0}$ and captures all possible linear combinations of the elements in \mathbb{R}^n that produce $\mathbf{0} \in \mathbb{R}^m$. [49]
5. The kernel is a subspace of \mathbb{R}^n , where n is the “width” of the matrix. [49]
6. The kernel focuses on the relationship among the columns, and we can use it to determine whether/how we can express a column as a linear combination of other columns. [49]

14.4. Inhomogeneous systems of linear equations and affine subspaces

1. For $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^m$, the solution of the system of linear equations $\mathbf{A}\boldsymbol{\lambda} = \mathbf{x}$ is either the empty set or an affine subspace of \mathbb{R}^n of dimension $n - \text{rk}(\mathbf{A})$. In particular, the solution of the linear equation $\lambda_1 \mathbf{b}_1 + \dots + \lambda_n \mathbf{b}_n = \mathbf{x}$, where $(\lambda_1, \dots, \lambda_n) \neq (0, \dots, 0)$, is a hyperplane in \mathbb{R}^n . [49]
2. In \mathbb{R}^n , every k -dimensional affine subspace is the solution of an inhomogeneous system of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\text{rk}(\mathbf{A}) = n - k$. For homogeneous equation systems $\mathbf{Ax} = \mathbf{0}$ the solution was a vector subspace, which we can also think of as a special affine space with support point $\mathbf{x}_0 = \mathbf{0}$. [49]

14.5. Types of matrices

14.5.1. Square matrix

1. $A \in \mathbb{R}^{n \times n}$ we call (n, n) -matrices square matrices because they possess the same number of rows and columns. [49]

2.

Definition 14.2 (Defective (square) matrix). A square matrix $A \in \mathbb{R}^{n \times n}$ is **defective** if it possesses fewer than n linearly independent eigenvectors. [49]

- (a) A non-defective matrix $A \in \mathbb{R}^{n \times n}$ does not necessarily require n distinct eigenvalues, but it does require that the eigenvectors form a basis of \mathbb{R}^n . [49]
- (b) sum of the dimensions of the eigenspaces is less than n [49]
- (c) a defective matrix has at least one eigenvalue λ_i with an algebraic multiplicity $m > 1$ and a geometric multiplicity of less than m . [49]
- (d) A defective matrix cannot have n distinct eigenvalues, as distinct eigenvalues have linearly independent eigenvectors [49]

```

1 import numpy as np
2
3 n = 4
4
5 print(np.random.randint(-10, 10, size=(n, n)))

```

Python Snippet 14.8: Square matrix - numPy

14.5.2. Identity Matrix (I_n)

In $\mathbb{R}^{n \times n}$, we define the identity matrix:

$$I_n := \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{n \times n} \quad [49]$$

as the $n \times n$ -matrix containing 1 on the diagonal and 0 everywhere else.

```

1 import numpy as np
2
3 n = 4
4
5 print(np.eye(n,n))

```

Python Snippet 14.9: Identity Matrix - numPy

14.5.3. Regular/ Invertible/ Non-singular matrix

A matrix A is called regular/ invertible/ non-singular if A^{-1} exists. [49]

1. if A is invertible, then so is A^\top [49]

14.5.4. Singular/ Non-invertible matrix

A matrix A is called singular/ non-invertible if A^{-1} doesn't exist. [49]

14.5.5. Symmetric Matrix

```

1 import numpy as np
2
3 n = 4
4
5 A = np.random.randint(-10, 10, size=(n, n))
6
7 # converting A to a symmetric matrix
8 A = (A + A.T) / 2
9
10 print(A)

```

Python Snippet 14.10: Identity Matrix - numPy

1. A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric if $\mathbf{A} = \mathbf{A}^\top$ [49]
2. only (n, n) -matrices can be symmetric [49]
3. The sum of symmetric matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ is **always symmetric**. [49]
4. although the product of 2 symmetric matrices is **always defined**, it is generally **not symmetric** [49]

14.5.6. Symmetric, Positive Definite (SPD) Matrix

1.

Definition 14.3 (Symmetric, Positive Definite (SPD) Matrix). A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ that satisfies $\forall \mathbf{x} \in V \setminus \{\mathbf{0}\} : \mathbf{x}^\top \mathbf{A} \mathbf{x} > 0$ is called **symmetric, positive definite**, or just **positive definite**. [49]

2. Consider an n -dimensional vector space V with an inner product $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{R}$ and an ordered basis $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of V . Any vectors $\mathbf{x}, \mathbf{y} \in V$ can be written as linear combinations of the basis vectors so that $\mathbf{x} = \sum_{i=1}^n \psi_i \mathbf{b}_i \in V$ and $\mathbf{y} = \sum_{j=1}^n \lambda_j \mathbf{b}_j \in V$ for suitable $\psi_i, \lambda_j \in \mathbb{R}$. Due to the bilinearity of the inner product, it holds for all $\mathbf{x}, \mathbf{y} \in V$ that [49]

$$\langle \mathbf{x}, \mathbf{y} \rangle = \left\langle \sum_{i=1}^n \psi_i \mathbf{b}_i, \sum_{j=1}^n \lambda_j \mathbf{b}_j \right\rangle = \sum_{i=1}^n \sum_{j=1}^n \psi_i \langle \mathbf{b}_i, \mathbf{b}_j \rangle \lambda_j = \hat{\mathbf{x}}^\top \mathbf{A} \hat{\mathbf{y}}$$
 [49]

where $A_{ij} := \langle \mathbf{b}_i, \mathbf{b}_j \rangle$ and $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ are the coordinates of \mathbf{x} and \mathbf{y} with respect to the basis B . This implies that the inner product $\langle \cdot, \cdot \rangle$ is uniquely determined through \mathbf{A} . The symmetry of the inner product also means that \mathbf{A} is symmetric. Furthermore, the positive definiteness of the inner product implies that $\forall \mathbf{x} \in V \setminus \{\mathbf{0}\} : \mathbf{x}^\top \mathbf{A} \mathbf{x} > 0$. [49]

3. If only \geq holds, then \mathbf{A} is called **symmetric, positive semi-definite**. [49]

4. If $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric, positive definite, then [49]

$$\langle \mathbf{x}, \mathbf{y} \rangle = \hat{\mathbf{x}}^\top \mathbf{A} \hat{\mathbf{y}}$$
 [49]

defines an inner product with respect to an ordered basis B , where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are the coordinate representations of $\mathbf{x}, \mathbf{y} \in V$ with respect to B . [49]

5.

Theorem 14.2 (SPD: inner product). For a real-valued, finite-dimensional vector space V and an ordered basis B of V , it holds that $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{R}$ is an inner product if and only if there exists a symmetric, positive definite matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with [49]

$$\langle \mathbf{x}, \mathbf{y} \rangle = \hat{\mathbf{x}}^\top \mathbf{A} \hat{\mathbf{y}}$$
 [49]

The following properties hold if $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric and positive definite: [49]

- (a) The null space (kernel) of \mathbf{A} consists only of $\mathbf{0}$ because $\mathbf{x}^\top \mathbf{A} \mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$. This implies that $\mathbf{A} \mathbf{x} \neq \mathbf{0}$ if $\mathbf{x} \neq \mathbf{0}$. [49]

- (b) The diagonal elements a_{ii} of \mathbf{A} are positive because $a_{ii} = \mathbf{e}_i^\top \mathbf{A} \mathbf{e}_i > 0$, where \mathbf{e}_i is the i -th vector of the standard basis in \mathbb{R}^n . [49]

6.

Theorem 14.3 (SPD: obtaining SPD). Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can always obtain a symmetric, positive semidefinite matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ by defining $\mathbf{S} := \mathbf{A}^\top \mathbf{A}$. [49]

- (a) If $\text{rk}(\mathbf{A}) = n$, then $\mathbf{S} := \mathbf{A}^\top \mathbf{A}$ is symmetric, positive definite. [49]

14.5.7. Equivalent Matrices

1.

Definition 14.4 (Equivalent Matrices). Two matrices $\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{R}^{m \times n}$ are equivalent if there exist regular matrices $S \in \mathbb{R}^{n \times n}$ and $T \in \mathbb{R}^{m \times m}$, such that $\tilde{\mathbf{A}} = T^{-1} \mathbf{A} S$. [49]

2. equivalent matrices are not necessarily similar. [49]

14.5.8. Similar Matrices

1.

Definition 14.5 (Similar Matrices). Two matrices $\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$ are similar if there exists a regular matrix $S \in \mathbb{R}^{n \times n}$ with $\tilde{\mathbf{A}} = S^{-1} \mathbf{A} S$. [49]

2. Similar matrices are always equivalent. [49]

14.5.9. Orthogonal Matrix

1. A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix if and only if its columns are orthonormal so that [49]

$$\mathbf{A} \mathbf{A}^\top = \mathbf{I} = \mathbf{A}^\top \mathbf{A} \quad \Rightarrow \quad \mathbf{A}^{-1} = \mathbf{A}^\top \quad [49]$$

i.e., the inverse is obtained by simply transposing the matrix. [49]

14.5.10. Triangular matrix (T)

1. We call a square matrix \mathbf{T} an **upper-triangular matrix** if $T_{ij} = 0$ for upper-triangular matrix $i > j$, i.e., the matrix is zero below its diagonal. [49]

2. Analogously, we define a **lower-triangular matrix** as a matrix with zeros above its diagonal. [49]

14.5.11. Diagonal Matrix (D)

1. A diagonal matrix is a matrix that has value zero on all off-diagonal elements, i.e., they are of the form: [49]

$$D = \begin{bmatrix} c_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_n \end{bmatrix} \quad [49]$$

2. They allow fast computation of determinants, powers, and inverses.

- (a) The determinant is the product of its diagonal entries [49]

- (b) a matrix power D^k is given by each diagonal element raised to the power k [49]

- (c) the inverse D^{-1} is the reciprocal of its diagonal elements if all of them are nonzero [49]

$$|\mathbf{D}| = \begin{vmatrix} c_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_n \end{vmatrix} = \prod_{i=1}^n c_i \quad \mathbf{D}^k = \begin{bmatrix} c_1^k & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_n^k \end{bmatrix} \quad \mathbf{D}^{-1} = \begin{bmatrix} \frac{1}{c_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{c_n} \end{bmatrix}$$

3.

Definition 14.6 (Diagonalizable). A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is diagonalizable if it is similar to a diagonal matrix, i.e., if there exists an invertible matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ such that $\mathbf{D} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$. [49]

4.

Theorem 14.4 (Symmetric Matrix: always diagonalizable). A symmetric matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ can always be diagonalized. [49]

- (a) follows directly from the spectral theorem [49]
- (b) \mathbf{P} an orthogonal matrix [49]
- (c) $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$ [49]

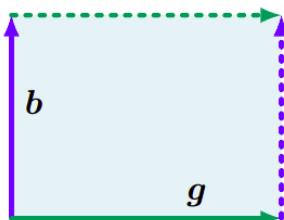
14.6. Norms of a Matrix

1.

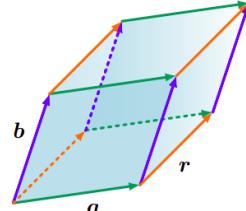
Definition 14.7 (Spectral Norm of a Matrix). For $\mathbf{x} \in \mathbb{R}^n \setminus \{0\}$, the spectral norm of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined as $\|\mathbf{A}\|_2 := \max_{\mathbf{x}} \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2}$ [49]

14.7. Describing/summarizing Matrix

14.7.1. Determinant ($\det(\mathbf{A})$ or $|A|$)



The area of the parallelogram (shaded region) spanned by the vectors \mathbf{b} and \mathbf{g} is $|\det([\mathbf{b}, \mathbf{g}])|$. [49]



The volume of the parallelepiped (shaded volume) spanned by vectors \mathbf{r} , \mathbf{b} , \mathbf{g} is $|\det([\mathbf{r}, \mathbf{b}, \mathbf{g}])|$. [49]

.

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \quad [49]$$

1.

Definition 14.8 (Determinant). A determinant is a mathematical object in the analysis and solution of systems of linear equations. [49]

2. Determinants are **only** defined for square matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$, i.e., matrices with the same number of rows and columns. [49]

3. The determinant of a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a function that maps \mathbf{A} onto a real number. [49]
4. For $n = 1$, $\det(\mathbf{A}) = \det(a_{11}) = a_{11}$ [49]
5. For $n = 2$, $\det(\mathbf{A}) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$ [49]

6.

Definition 14.9 (Determinant: Sarrus' Rule). For $n = 3$, [49]

$$\det(\mathbf{A}) = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{21}a_{32}a_{13} + a_{31}a_{12}a_{23} - a_{31}a_{22}a_{13} - a_{11}a_{32}a_{23} - a_{21}a_{12}a_{33}$$

[49]

7. For a triangular matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$, the determinant is the product of the diagonal elements, i.e., $\det(\mathbf{T}) = \sum_{i=1}^n T_{ii}$ [49]
8. the determinant $\det(\mathbf{A})$ is the signed volume of an n -dimensional parallelepiped formed by columns of the matrix \mathbf{A} . [49]
9. The sign of the determinant indicates the orientation of the spanning vectors. [49]

10.

Theorem 14.5 (Determinant: Laplace Expansion). Consider a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Then, for all $j = 1, \dots, n$: [49]

- (a) Expansion along column j : $\det(\mathbf{A}) = \sum_{k=1}^n (-1)^{k+j} a_{kj} \det(\mathbf{A}_{k,j})$ [49]
- (b) Expansion along row j : $\det(\mathbf{A}) = \sum_{k=1}^n (-1)^{k+j} a_{jk} \det(\mathbf{A}_{j,k})$ [49]
- (c) $\mathbf{A}_{k,j} \in \mathbb{R}^{(n-1) \times (n-1)}$ is the sub-matrix of \mathbf{A} that we obtain when deleting row k and column j . [49]
- (d) $\det(\mathbf{A}_{k,j})$ is called a **minor** and $(-1)^{k+j} \det(\mathbf{A}_{k,j})$ a **cofactor**. [49]

11.

Theorem 14.6 (Determinant: product of eigenvalues). The determinant of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the product of its eigenvalues, i.e., [49]

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

[49]

where $\lambda_i \in \mathbb{C}$ are (possibly repeated) eigenvalues of \mathbf{A} . [49]

```

1 import numpy as np
2
3 # Define a square matrix
4 A = np.array([
5     [1, 2],
6     [3, 4]
7 ])
8
9 # Compute the determinant
10 det_A = np.linalg.det(A)
11
12 print("Matrix A:\n", A)
13 print("Determinant of A:", det_A)

```

Python Snippet 14.11: Determinant of matrix - numPy

14.7.1.1. Properties of Determinant

For $\mathbf{A} \in \mathbb{R}^{n \times n}$ the determinant exhibits the following properties: [49]

1. The determinant of a matrix product is the product of the corresponding determinants, $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$. [49]
2. Determinants are invariant to transposition, i.e., $\det(\mathbf{A}) = \det(\mathbf{A}^\top)$. [49]
3. If \mathbf{A} is regular (invertible), then $\det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})}$. [49]
4. Similar matrices possess the same determinant. Therefore, for a linear mapping $\Phi : V \rightarrow V$ all transformation matrices \mathbf{A}_Φ of Φ have the same determinant. Thus, the determinant is invariant to the choice of basis of a linear mapping. [49]
5. Adding a multiple of a column/row to another one does not change $\det(\mathbf{A})$. [49]
6. Multiplication of a column/row with $\lambda \in \mathbb{R}$ scales $\det(\mathbf{A})$ by λ . In particular, $\det(\lambda\mathbf{A}) = \lambda^n \det(\mathbf{A})$. [49]
7. Swapping two rows/columns changes the sign of $\det(\mathbf{A})$. [49]

Note: Because of the last three properties, we can use Gaussian elimination to compute $\det(\mathbf{A})$ by bringing \mathbf{A} into row-echelon form. We can stop Gaussian elimination when we have \mathbf{A} in a triangular form where the elements below the diagonal are all 0. (the determinant of a triangular matrix is the product of the diagonal elements)

14.7.2. Trace ($\text{tr}(\mathbf{A})$)

```

1 import numpy as np
2
3 # Define a square matrix
4 A = np.array([
5     [1, 2, 3],
6     [4, 5, 6],
7     [7, 8, 9]
8 ])
9
10 # Compute the trace (sum of diagonal elements)
11 trace_A = np.trace(A)
12
13 print("Matrix A:\n", A)
14 print("Trace of A:", trace_A)

```

Python Snippet 14.12: Trace of a matrix - numPy

1.

Definition 14.10 (Trace). The trace of a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined as $\text{tr}(\mathbf{A}) := \sum_{i=1}^n a_{ii}$ i.e. , the trace is the **sum of the diagonal elements of \mathbf{A}** . [49]

2.

Theorem 14.7 (Trace: sum of eigenvalues). The trace of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the sum of its eigenvalues, i.e., [49]

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n \lambda_i \quad [49]$$

where $\lambda_i \in \mathbb{C}$ are (possibly repeated) eigenvalues of \mathbf{A} . [49]

14.7.2.1. Properties of Trace

1. $\text{tr}(\mathbf{A} + \mathbf{B}) = \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B})$ for $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ [49]
2. $\text{tr}(\alpha \mathbf{A}) = \alpha \text{tr}(\mathbf{A}), \alpha \in \mathbb{R}$ for $\mathbf{A} \in \mathbb{R}^{n \times n}$ [49]
3. $\text{tr}(\mathbf{I}_n) = n$ [49]
4. $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$ for $\mathbf{A} \in \mathbb{R}^{n \times k}, \mathbf{B} \in \mathbb{R}^{k \times n}$ [49]
5. The trace is invariant under cyclic permutations, ie, $\text{tr}(\mathbf{AKL}) = \text{tr}(\mathbf{KLA})$ for matrices $\mathbf{A} \in \mathbb{R}^{a \times k}, \mathbf{K} \in \mathbb{R}^{k \times l}, \mathbf{L} \in \mathbb{R}^{l \times a}$. [49]
6. $\text{tr}(\mathbf{xy}^\top) = \text{tr}(\mathbf{y}^\top \mathbf{x}) = \mathbf{y}^\top \mathbf{x} \in \mathbb{R} \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ [49]
7. Given $\Phi : V \rightarrow V$, then $\text{tr}(\Phi) = \text{tr}(\mathbf{A}_\Phi)$. while matrix representations of linear mappings are basis dependent the trace of a linear mapping Φ is independent of the basis. [49]

14.7.3. Characteristic Polynomial ($p_A(\lambda)$)

1.

Definition 14.11 (Characteristic Polynomial). For $\lambda \in \mathbb{R}$ and a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$:

$$p_{\mathbf{A}}(\lambda) := \det(\mathbf{A} - \lambda \mathbf{I})$$

$$= c_0 + c_1 \lambda + c_2 \lambda^2 + \cdots + c_{n-1} \lambda^{(n-1)} + (-1)^n \lambda^n$$

$c_0, \dots, c_{n-1} \in \mathbb{R}$, is the characteristic polynomial of \mathbf{A} .

2. $c_0 = \det(\mathbf{A}), c_{n-1} = (-1)^{n-1} \text{tr}(\mathbf{A})$

14.7.4. Eigenvalues (λ) & Eigenvectors

1. Eigen is a German word meaning “characteristic”, “self”, or “own”. [49]

2.

Definition 14.12 (Eigenvalues & Eigenvectors). Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square matrix. Then $\lambda \in \mathbb{R}$ is an eigenvalue of \mathbf{A} and $\mathbf{x} \in \mathbb{R}^n \setminus \{0\}$ is the corresponding eigenvector of \mathbf{A} if $\mathbf{Ax} = \lambda \mathbf{x}$. [49]

3. $\mathbf{Ax} = \lambda \mathbf{x}$ is called eigenvalue equation. [49]

4. it is often a convention that eigenvalues are sorted in descending order, so that the largest eigenvalue and associated eigenvector are called the **first eigenvalue** and its associated eigenvector, and the second largest called the **second eigenvalue** and its associated eigenvector, and so on. [49]

5. The following statements are equivalent:

(a) λ is an eigenvalue of $\mathbf{A} \in \mathbb{R}^{n \times n}$ [49]

(b) There exists an $\mathbf{x} \in \mathbb{R}^n \setminus \{0\}$ with $\mathbf{Ax} = \lambda \mathbf{x}$, or equivalently, $(\mathbf{A} - \lambda \mathbf{I}_n)\mathbf{x} = 0$ can be solved non-trivially, i.e., $\mathbf{x} \neq 0$. [49]

(c) $\text{rk}(\mathbf{A} - \lambda \mathbf{I}_n) < n$ [49]

(d) $\det(\mathbf{A} - \lambda \mathbf{I}_n) = 0$ [49]

6. **Non-uniqueness of eigenvectors:** If \mathbf{x} is an eigenvector of \mathbf{A} associated with eigenvalue λ , then for any $c \in \mathbb{R} \setminus \{0\}$ it holds that $c\mathbf{x}$ is an eigenvector of \mathbf{A} with the same eigenvalue since $\mathbf{A}(c\mathbf{x}) = c\mathbf{Ax} = c\lambda\mathbf{x} = \lambda(c\mathbf{x})$. All vectors that are collinear to \mathbf{x} are also eigenvectors of \mathbf{A} . [49]

7.

Theorem 14.8 (Eigenvalue: root of the characteristic polynomial). $\lambda \in \mathbb{R}$ is an eigenvalue of $\mathbf{A} \in \mathbb{R}^{n \times n}$ if and only if λ is a root of the characteristic polynomial $p_{\mathbf{A}}(\lambda)$ of \mathbf{A} . [49]

8.

Definition 14.13 (Algebraic Multiplicity). Let a square matrix \mathbf{A} have an eigenvalue λ_i . The **algebraic multiplicity** of λ_i is the number of times the root appears in the characteristic polynomial. [49]

9.

Definition 14.14 (Geometric Multiplicity). Let λ_i be an eigenvalue of a square matrix \mathbf{A} . Then the **geometric multiplicity** of λ_i is the number of linearly independent eigenvectors associated with λ_i . In other words, it is the dimensionality of the eigenspace spanned by the eigenvectors associated with λ_i . [49]

10. A specific eigenvalue's geometric multiplicity must be at least one because every eigenvalue has at least one associated eigenvector. An eigenvalue's geometric multiplicity cannot exceed its algebraic multiplicity, but it may be lower.

11. Geometrically, the eigenvector corresponding to a nonzero eigenvalue points in a direction that is stretched by the linear mapping. The eigenvalue is the factor by which it is stretched. If the eigenvalue is negative, the direction of the stretching is flipped. [49]

12. **The Case of the Identity Matrix:** The identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ has characteristic polynomial $p_{\mathbf{I}}(\lambda) = \det(\mathbf{I} - \lambda \mathbf{I}) = (1 - \lambda)^n = 0$, which has only one eigenvalue $\lambda = 1$ that occurs n times. Moreover, $\mathbf{Ix} = \lambda x = 1x$ holds for all vectors $x \in \mathbb{R}^n \setminus \{0\}$. Because of this, the sole eigenspace E_1 of the identity matrix spans n dimensions, and all n standard basis vectors of \mathbb{R}^n are eigenvectors of \mathbf{I} . [49]

13.

Theorem 14.9 (eigenvectors: linearly independent). The eigenvectors x_1, \dots, x_n of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with n distinct eigenvalues $\lambda_1, \dots, \lambda_n$ are linearly independent. This theorem states that eigenvectors of a matrix with n distinct eigenvalues form a basis of \mathbb{R}^n . [49]

14.

Theorem 14.10 (Spectral Theorem). If $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric, there exists an orthonormal basis of the corresponding vector space V consisting of eigenvectors of \mathbf{A} , and each eigenvalue is real. [49]

(a) A direct implication of the spectral theorem is that the eigen-decomposition of a symmetric matrix \mathbf{A} exists (with real eigenvalues), and that we can find an ONB of eigenvectors so that $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^\top$, where \mathbf{D} is diagonal and the columns of \mathbf{P} contain the eigenvectors. [49]

```

1 import numpy as np
2
3 # Define a square matrix
4 A = np.random.randint(-10, 10, size=(3,3), dtype=int)
5 print(A)
6
7 # Compute eigenvalues and eigenvectors
8 # eigenvalues: 1D array of eigenvalues.
9 # eigenvectors: 2D array where each column is an eigenvector.
10 eigenvalues, eigenvectors = np.linalg.eig(A)
11 print(eigenvalues)
12 print(eigenvectors)

```

Python Snippet 14.13: Eigenvalues & Eigenvectors of a Matrix - numPy

14.7.4.1. Properties of eigenvalues & eigenvectors

1. A matrix \mathbf{A} and its transpose \mathbf{A}^\top possess the same eigenvalues, but not necessarily the same eigenvectors. [49]
2. Similar matrices possess the same eigenvalues. Therefore, a linear mapping Φ has eigenvalues that are independent of the choice of basis of its transformation matrix. [49]

3. Symmetric, positive definite (SPD) matrices **always** have positive, real eigenvalues. [49]

14.7.5. Eigenspace (E_λ) & Eigenspectrum

1.

Definition 14.15 (Eigenspace). For $\mathbf{A} \in \mathbb{R}^{n \times n}$, the set of all eigenvectors of \mathbf{A} associated with an eigenvalue λ spans a subspace of \mathbb{R}^n , which is called the **eigenspace** of \mathbf{A} with respect to λ and is denoted by E_λ . [49]

2.

Definition 14.16 (Eigenspectrum). The set of all eigenvalues of \mathbf{A} is called the **eigenspectrum**, or just spectrum, of \mathbf{A} . [49]

3. If λ is an eigenvalue of $\mathbf{A} \in \mathbb{R}^{n \times n}$, then the corresponding eigenspace E_λ is the solution space of the homogeneous system of linear equations $(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = 0$. [49]

14.7.5.1. Properties of Eigenspace & Eigenspectrum

1. The eigenspace E_λ is the null space of $\mathbf{A} - \lambda \mathbf{I}$ since: [49]

$$\begin{aligned} \mathbf{Ax} = \lambda \mathbf{x} &\iff \mathbf{Ax} - \lambda \mathbf{x} = 0 \\ &\iff (\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = 0 \iff \mathbf{x} \in \ker(\mathbf{A} - \lambda \mathbf{I}) \end{aligned} \quad [49]$$

14.8. Matrix Decomposition/Factorization

14.8.1. Cholesky Decomposition ($A = LL^\top$)

1.

Theorem 14.11 (Cholesky Decomposition). A symmetric, positive definite matrix \mathbf{A} can be factorized into a product $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, where \mathbf{L} is a lower-triangular matrix with positive diagonal elements: [49]

$$\underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}}_A = \underbrace{\begin{bmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \cdots & l_{nn} \end{bmatrix}}_L \underbrace{\begin{bmatrix} l_{11} & \cdots & l_{n1} \\ 0 & \cdots & l_{nn} \end{bmatrix}}_{L^\top} \quad [49]$$

\mathbf{L} is called the **Cholesky factor** of \mathbf{A} , and \mathbf{L} is unique. [49]

2. In machine learning, symmetric positive definite matrices require frequent manipulation, e.g., the covariance matrix of a multivariate Gaussian variable is symmetric, positive definite. The Cholesky factorization of this covariance matrix allows us to generate samples from a Gaussian distribution. [49]
3. It also allows us to perform a linear transformation of random variables, which is heavily exploited when computing gradients in deep stochastic models, such as the variational auto-encoder. [49]
4. The Cholesky decomposition also allows us to compute determinants very efficiently: $\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{L}^\top) = \det(\mathbf{L})^2 = \prod_i l_{ii}^2$ [49]

```

1 import numpy as np
2
3 # generate a random matrix
4 A = np.random.randint(-10, 10, size=(3,3))
5 # Create a symmetric positive definite matrix
6 A = A.T @ A
7
8 # Perform Cholesky decomposition

```

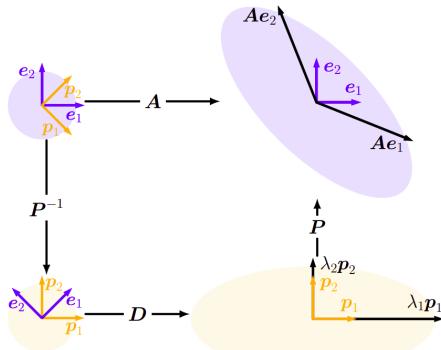
```

9 L = np.linalg.cholesky(A)
10
11 print("Matrix A:")
12 print(A)
13 print("\nCholesky factor L:")
14 print(L)

```

Python Snippet 14.14: Cholesky Decomposition - numPy

14.8.2. Eigendecomposition ($A = PDP^{-1}$)



Intuition behind the eigendecomposition as sequential transformations.

Top-left to bottom-left: P^{-1} performs a basis change (here drawn in \mathbb{R}^2 and depicted as a rotation-like operation) from the standard basis into the eigenbasis.

Bottom-left to bottom-right: D performs a scaling along the remapped orthogonal eigenvectors, depicted here by a circle being stretched to an ellipse.

Bottom-right to top-right: P undoes the basis change (depicted as a reverse rotation) and restores the original coordinate frame.

1.

Theorem 14.12 (Eigendecomposition). A square matrix $A \in \mathbb{R}^{n \times n}$ can be factored into $A = PDP^{-1}$ where $P \in \mathbb{R}^{n \times n}$ and D is a diagonal matrix whose diagonal entries are the eigenvalues of A , if and only if the eigenvectors of A form a basis of \mathbb{R}^n . [49]

2. Steps:

- (a) Compute eigenvalues and eigenvectors
- (b) Check for existence (eigenvectors span \mathbb{R}^n or not)
- (c) Construct the matrix P to diagonalize A
 - i. $P = [p_1, \dots, p_n]$
 - ii. $D = P^{-1}AP = P^TAP$ (if A is symmetric, $P^{-1} = P^T$)

3. we can find a matrix power for a matrix $A \in \mathbb{R}^{n \times n}$ via the eigenvalue decomposition (if it exists) so that: [49]

$$A^k = (PDP^{-1})^k = PDP^{-1}PDP^{-1}\dots PDP^{-1} = PD^kP^{-1} \quad [49]$$

4. Assume that the eigendecomposition $A = (PDP^{-1})$ exists. Then: [49]

$$\det(A) = \det(PDP^{-1}) = \det(P)\det(D)\det(P^{-1}) = \det(D) = \prod_i d_{ii} \quad [49]$$

5. eigendecomposition operates within the same vector space, where the same basis change is applied and then undone. [49]

```

1 import numpy as np
2
3 # Define a square matrix
4 A = np.array([[4, -2],
5               [1, 1]])
6
7 # Perform eigendecomposition
8 eigenvalues, eigenvectors = np.linalg.eig(A)
9
10 # Construct D and P
11 D = np.diag(eigenvalues)
12 P = eigenvectors
13
14 # Inverse of P
15 P_inv = np.linalg.inv(P)
16
17 # Reconstruct A
18 A_reconstructed = P @ D @ P_inv
19
20 # Print results
21 print("Original matrix A:")
22 print(A)
23
24 print("\nEigenvalues (D):")
25 print(D)
26
27 print("\nEigenvectors (P):")
28 print(P)
29
30 print("\nReconstructed A (P D P_inv):")
31 print(A_reconstructed)
32
33 print("\nCheck reconstruction accuracy:", np.allclose(A, A_reconstructed))

```

Python Snippet 14.15: Eigendecomposition - numPy

14.8.3. Singular Value Decomposition (SVD) ($A = U\Sigma V^\top$)

1. The singular value decomposition (SVD) of a matrix is a central matrix decomposition method in linear algebra. [49]
2. It has been referred to as the “fundamental theorem of linear algebra” because it can be applied to all matrices, not only to square matrices, and it always exists. [49]
3. the SVD of a matrix A , which represents a linear mapping $\Phi : V \rightarrow W$, quantifies the change between the underlying geometry of these two vector spaces. [49]
- 4.

Theorem 14.13 (SVD Theorem). Let $A \in \mathbb{R}^{m \times n}$ be a rectangular matrix of rank $r \in [0, \min(m, n)]$. The SVD of A is a decomposition of the form $\underset{m \times n}{A} = \underset{m \times m}{U} \underset{m \times n}{\Sigma} \underset{n \times n}{V}^\top$ with an orthogonal matrix $U \in \mathbb{R}^{m \times m}$ with column vectors u_i , $i = 1, \dots, m$, and an orthogonal matrix $V \in \mathbb{R}^{n \times n}$ with column vectors v_j , $j = 1, \dots, n$. Moreover, Σ is an $m \times n$ matrix with $\Sigma_{ii} = \sigma_i \geq 0$ and $\Sigma_{ij} = 0, i \neq j$. [49]

5. U

- (a) u_i are called the left-singular vectors

[49]

6. Σ :

- (a) The diagonal entries σ_i , $i = 1, \dots, r$, of Σ are called the singular values [49]
- (b) By convention, the singular values are ordered, i.e., $\sigma_1 \geq \sigma_2 \geq \sigma_r \geq 0$. [49]
- (c) The singular value matrix Σ is unique [49]
- (d) the Σ is rectangular, particularly, it is of the same size as A [49]
- (e) Σ has a diagonal submatrix that contains the singular values and needs additional zero padding [49]

if $m > n$, then the matrix Σ has diagonal structure up to row n and then consists of $\mathbf{0}^\top$ row vectors from $n+1$ to m below so that:

$$\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \\ 0 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & 0 \end{bmatrix} \quad [49]$$

If $m < n$, the matrix Σ has a diagonal structure up to column m and columns that consist of $\mathbf{0}$ from $m+1$ to n :

$$\begin{bmatrix} \sigma_1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_m & 0 & 0 & 0 \end{bmatrix} \quad [49]$$

 7. V

- (a) v_j are called the right-singular vectors [49]

8.

Definition 14.17 (Standard SVD/ Full SVD ($A = U\Sigma V^\top$)). SVD notation where the SVD is described as having two square left- and right-singular vector matrices, but a non-square singular value matrix. [49]

$$\underset{m \times n}{A} = \underset{m \times m}{U} \underset{m \times m}{\Sigma} \underset{n \times n}{V}^\top \quad [49]$$

9.

Definition 14.18 (Reduced SVD ($A = U_r \Sigma_r V_r^\top$)). focus on square singular matrices [49]

$$\underset{m \times n}{A} = \underset{m \times n}{U_r} \underset{n \times n}{\Sigma_r} \underset{n \times n}{V_r}^\top \quad (m \geq n) \quad [49]$$

This alternative format changes merely how the matrices are constructed but leaves the mathematical structure of the SVD unchanged. The convenience of this alternative formulation is that Σ is diagonal, as in the eigenvalue decomposition. [49]

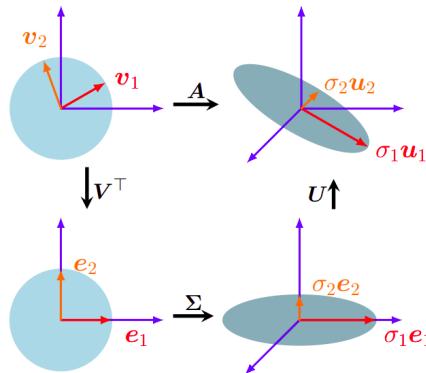
10.

Definition 14.19 (Compact SVD/ Thin SVD/ Truncated SVD ($A = U\Sigma V$)). It is possible to define the SVD of a rank- r matrix A so that U is an $m \times r$ matrix, Σ a diagonal matrix $r \times r$, and V an $r \times n$ matrix. This construction is very similar to our definition, and ensures that the diagonal matrix Σ has only nonzero entries along the diagonal. The main convenience of this alternative notation is that Σ is diagonal, as in the eigenvalue decomposition.

$$\underset{m \times n}{A} = \underset{m \times r}{U} \underset{r \times r}{\Sigma} \underset{r \times n}{V} \quad [49]$$

11. A restriction that the SVD for A only applies to $m \times n$ matrices with $m > n$ is practically unnecessary. When $m < n$, the SVD decomposition will yield Σ with more zero columns than rows and, consequently, the singular values $\sigma_{m+1}, \dots, \sigma_n$ are 0. [49]

14.8.3.1. Geometric Intuitions for the SVD



Intuition behind the SVD of a matrix $\mathbf{A} \in \mathbb{R}^{3 \times 2}$ as sequential transformations. [49]

Top-left to bottom-left: \mathbf{V}^\top performs a basis change in \mathbb{R}^2 . [49]

Bottom-left to bottom-right: Σ scales and maps from \mathbb{R}^2 to \mathbb{R}^3 . The ellipse in the bottom-right lives in \mathbb{R}^3 .

The third dimension is orthogonal to the surface of the elliptical disk. [49]

Bottom-right to top-right: \mathbf{U} performs a basis change within \mathbb{R}^3 . [49]

SVD as sequential linear transformations performed on the bases. The SVD of a matrix can be interpreted as a decomposition of a corresponding linear mapping $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ into three operations. Assume we are given a transformation matrix of a linear mapping $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with respect to the standard bases B and C of \mathbb{R}^n and \mathbb{R}^m , respectively. Moreover, assume a second basis \tilde{B} of \mathbb{R}^n and \tilde{C} of \mathbb{R}^m . Then: [49]

1. The matrix \mathbf{V} performs a basis change in the domain \mathbb{R}^n from \tilde{B} (represented by the red and orange vectors \mathbf{v}_1 and \mathbf{v}_2 in the top-left) to the standard basis B . $\mathbf{V}^\top = \mathbf{V}^{-1}$ performs a basis change from B to \tilde{B} . The red and orange vectors are now aligned with the canonical basis in the bottom-left. [49]
2. Having changed the coordinate system to \tilde{B} , Σ scales the new coordinates by the singular values σ_i (and adds or deletes dimensions), i.e., Σ is the transformation matrix of Φ with respect to \tilde{B} and \tilde{C} , represented by the red and orange vectors being stretched and lying in the $\mathbf{e}_1 - \mathbf{e}_2$ plane, which is now embedded in a third dimension in the bottom-right. [49]
3. \mathbf{U} performs a basis change in the co-domain \mathbb{R}^m from \tilde{C} into the canonical basis of \mathbb{R}^m , represented by a rotation of the red and orange vectors out of the $\mathbf{e}_1 - \mathbf{e}_2$ plane. This is shown in the top-right. [49]
4. The SVD expresses a change of basis in both the domain and codomain. What makes the SVD special is that these two different bases are simultaneously linked by the singular value matrix Σ . [49]

14.8.3.2. Construction of the SVD

1. Finding \mathbf{V} :

$$(a) \mathbf{A}^\top \mathbf{A} = \mathbf{P} \mathbf{D} \mathbf{P}^\top = \mathbf{P} \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix} \mathbf{P}^\top \in \mathbb{R}^{n \times n} \quad [49]$$

(b) \mathbf{P} is an orthogonal matrix, which is composed of the orthonormal eigenbasis. [49]

(c) The $\lambda_i \geq 0$ are the eigenvalues of $\mathbf{A}^\top \mathbf{A}$. [49]

(d) Let SVD exist. Then: [49]

$$\mathbf{A}^\top \mathbf{A} = (\mathbf{U} \Sigma \mathbf{V}^\top)^\top (\mathbf{U} \Sigma \mathbf{V}^\top) = \mathbf{V} \Sigma^\top \mathbf{U}^\top \mathbf{U} \Sigma \mathbf{V}^\top = \mathbf{V} \Sigma^\top \Sigma \mathbf{V}^\top = \mathbf{V} \begin{bmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{bmatrix} \mathbf{V}^\top \in \mathbb{R}^{n \times n} \quad [49]$$

(e) $\mathbf{V}^\top = \mathbf{P}^\top$ and $\sigma_i^2 = \lambda_i$ [49]

(f) Therefore, the eigenvectors of $\mathbf{A}^\top \mathbf{A}$ that compose \mathbf{P} are the right-singular vectors \mathbf{V} of \mathbf{A} [49]

(g) The eigenvalues of $\mathbf{A}^\top \mathbf{A}$ are the squared singular values of Σ [49]

2. Finding \mathbf{U} :

$$(a) \mathbf{A}^\top \mathbf{A} = \mathbf{S}\mathbf{D}\mathbf{S}^\top = \mathbf{S} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_m \end{bmatrix} \mathbf{S}^\top \in \mathbb{R}^{m \times m} \quad [49]$$

(b) \mathbf{P} is an orthogonal matrix, which is composed of the orthonormal eigenbasis. [49]

(c) Let SVD exist. Then: [49]

$$\mathbf{A}\mathbf{A}^\top = (\mathbf{U}\Sigma\mathbf{V}^\top)(\mathbf{U}\Sigma\mathbf{V}^\top)^\top = \mathbf{U}\Sigma\mathbf{V}^\top\mathbf{V}\Sigma^\top\mathbf{U}^\top = \mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top = \mathbf{U} \begin{bmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_m^2 \end{bmatrix} \mathbf{U}^\top \in \mathbb{R}^{m \times m} \quad [49]$$

(d) $\mathbf{U}^\top = \mathbf{S}^\top$ and $\sigma_i^2 = \lambda_i$ [49]

(e) The orthonormal eigenvectors of $\mathbf{A}\mathbf{A}^\top$ are the left-singular vectors \mathbf{U} and form an orthonormal basis in the codomain of the SVD. [49]

3. Finding Σ :

(a) Since $\mathbf{A}\mathbf{A}^\top$ and $\mathbf{A}^\top \mathbf{A}$ have the same nonzero eigenvalues, the nonzero entries of the Σ matrices in the SVD for both cases have to be the same. [49]

(b) the images of the \mathbf{v}_i under \mathbf{A} have to be orthogonal, too. [49]

$$(\mathbf{A}\mathbf{v}_i)^\top(\mathbf{A}\mathbf{v}_j) = \mathbf{v}_i^\top(\mathbf{A}^\top\mathbf{A})\mathbf{v}_j = \mathbf{v}_i^\top(\lambda_j\mathbf{v}_j) = \lambda_j\mathbf{v}_i^\top\mathbf{v}_j = 0 \quad [49]$$

(c) For the case $m \geq r$ (where $r = rk(\mathbf{A})$), it holds that $\{\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_r\}$ is a basis of an r -dimensional subspace of \mathbb{R}^m . [49]

(d) To complete the SVD construction, we need left-singular vectors that are orthonormal: We normalize the images of the right-singular vectors $\mathbf{A}\mathbf{v}_i$ and obtain: [49]

$$\mathbf{u}_i = \frac{\mathbf{A}\mathbf{v}_i}{\|\mathbf{A}\mathbf{v}_i\|} = \frac{\mathbf{A}\mathbf{v}_i}{\sqrt{\lambda_i}} = \frac{\mathbf{A}\mathbf{v}_i}{\sigma_i} \Rightarrow \mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i \quad (i = 1, \dots, r) \quad [49]$$

(e) singular value equation: $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ [49]

(f) **Author's Note:** As $\mathbf{V}^\top = \mathbf{V}^{-1}$,

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top = \mathbf{U}\Sigma\mathbf{V}^{-1} \Rightarrow \mathbf{AV} = \mathbf{U}\Sigma \Rightarrow \mathbf{U} = \mathbf{AV}\Sigma^{-1} \Rightarrow \mathbf{U} = \frac{\mathbf{AV}}{\Sigma}$$

As Σ is a diagonal matrix (more or less), $\Sigma^{-1} = \frac{1}{\Sigma}$ (element-wise)

4. the eigenvectors of $\mathbf{A}^\top \mathbf{A}$, which we know are the right-singular vectors \mathbf{v}_i , and their normalized images under \mathbf{A} , the left-singular vectors \mathbf{u}_i , form two self-consistent ONBs that are connected through the singular value matrix Σ . [49]

5. Case 1: $n < m$:

(a) $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ holds for $i \leq n$, i.e., only for the nonzero singular values. [7, 49]

(b) For $i > n$, \mathbf{u}_i are not defined via $\mathbf{A}\mathbf{v}_i$. [7, 49]

(c) But by construction of SVD, we choose \mathbf{u}_i to complete an orthonormal basis for \mathbb{R}^m , so the extra \mathbf{u}_i 's (for $i > n$) are orthonormal anyway. [7, 49]

6. Case 2: $m < n$

(a) $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ only holds for $i \leq m$. [7, 49]

(b) For $i > m$, $\mathbf{A}\mathbf{v}_i = 0$. So, these \mathbf{v}_i lie in the null space of \mathbf{A} . [7, 49]

(c) The set of all \mathbf{v}_i (even for $i > m$) is still orthonormal. [7, 49]

(d) So the columns \mathbf{v}_i with $i > m$ form an orthonormal basis for the null space of \mathbf{A} . [7, 49]

14.8.4. Eigenvalue Decomposition ($A = PDP^{-1}$) vs. Singular Value Decomposition ($A = U\Sigma V^\top$)

1. The SVD always exists for any matrix $\mathbb{R}^{m \times n}$. The eigendecomposition is only defined for square matrices $\mathbb{R}^{n \times n}$ and only exists if we can find a basis of eigenvectors of \mathbb{R}^n . [49]
2. The vectors in the eigendecomposition matrix P are not necessarily orthogonal, i.e., the change of basis is not a simple rotation and scaling. On the other hand, the vectors in the matrices U and V in the SVD are orthonormal, so they do represent rotations. [49]
3. Both the eigendecomposition and the SVD are compositions of three linear mappings: [49]
 - (a) Change of basis in the domain [49]
 - (b) Independent scaling of each new basis vector and mapping from domain to codomain [49]
 - (c) Change of basis in the codomain [49]

A key difference between the eigendecomposition and the SVD is that in the SVD, domain and codomain can be vector spaces of different dimensions. [49]

4. In the SVD, the left- and right-singular vector matrices U and V are generally not inverse of each other (they perform basis changes in different vector spaces). In the eigendecomposition, the basis change matrices P and P^{-1} are inverses of each other. [49]
5. In the SVD, the entries in the diagonal matrix Σ are all real and nonnegative, which is not generally true for the diagonal matrix in the eigendecomposition. [49]
6. The SVD and the eigendecomposition are closely related through their projections: [49]
 - (a) The left-singular vectors of A are eigenvectors of AA^\top [49]
 - (b) The right-singular vectors of A are eigenvectors of $A^\top A$. [49]
 - (c) The nonzero singular values of A are the square roots of the nonzero eigenvalues of both AA^\top and $A^\top A$. [49]
7. For symmetric matrices $A \in \mathbb{R}^{n \times n}$, the eigenvalue decomposition and the SVD are one and the same, which follows from the spectral theorem. [49]

14.9. Matrix Approximation

1. Instead of doing the full SVD factorization, SVD allows us to represent a matrix A as a sum of simpler (low-rank) matrices A_i , which lends itself to a matrix approximation scheme that is cheaper to compute than the full SVD. [49]
2. We construct a rank-1 matrix $A_i := u_i v_i^\top \in \mathbb{R}^{m \times n}$ which is formed by the outer product of the i -th orthogonal column vector of U and V . [49]
3. A matrix $A \in \mathbb{R}^{m \times n}$ of rank r can be written as a sum of rank-1 matrices A_i so that $A = \sum_{i=1}^r \sigma_i u_i v_i^\top = \sum_{i=1}^r \sigma_i A_i$ where the outer-product matrices A_i are weighted by the i -th singular value σ_i . [49]
4. If the sum does not run over all matrices A_i , $i = 1, \dots, r$, but only up to an intermediate value $k < r$, we obtain a rank- k approximation $\hat{A}(k) = \sum_{i=1}^k \sigma_i u_i v_i^\top = \sum_{i=1}^k \sigma_i A_i$ of A with $rk(\hat{A}(k)) = k$. [49]
- 5.

Theorem 14.14 (Eckart-Young Theorem). Consider a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ of rank r and let $\mathbf{B} \in \mathbb{R}^{m \times n}$ be a matrix of rank k . For any $k \leq r$ with $\hat{\mathbf{A}}(k) = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ it holds that $\hat{\mathbf{A}}(k) = \arg \max_{\text{rk}(\mathbf{B})=k} \|\mathbf{A} - \mathbf{B}\|_2$,
 $\|\mathbf{A} - \hat{\mathbf{A}}(k)\|_2 = \sigma_{k+1}$. [49]

- (a) The Eckart-Young theorem states explicitly how much error we introduce by approximating \mathbf{A} using a rank- k approximation. [49]
 - (b) We can interpret the rank- k approximation obtained with the SVD as a projection of the full-rank matrix \mathbf{A} onto a lower-dimensional space of rank-at-most- k matrices. [49]
 - (c) The SVD minimizes the error (with respect to the spectral norm) between \mathbf{A} and any rank- k approximation. [49]
 - (d) The Eckart-Young theorem implies that we can use SVD to reduce a rank- r matrix \mathbf{A} to a rank- k matrix $\hat{\mathbf{A}}$ in a principled, optimal (in the spectral norm sense) manner. [49]
6. the difference between $\mathbf{A} - \hat{\mathbf{A}}(k)$ is a matrix containing the sum of the remaining rank-1 matrices

$$\mathbf{A} - \hat{\mathbf{A}}(k) = \sum_{i=k+1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$$
 [49]
7. If we assume that there is another matrix \mathbf{B} with $\text{rk}(\mathbf{B}) \leq k$, such that $\|\mathbf{A} - \mathbf{B}\|_2 \leq \|\mathbf{A} - \hat{\mathbf{A}}(k)\|_2$, then there exists an at least $(n - k)$ -dimensional null space $Z \subseteq \mathbb{R}^n$, such that $\mathbf{x} \in Z$ implies that $\mathbf{Bx} = \mathbf{0}$. [49]
 Then it follows that $\|\mathbf{Ax}\|_2 = \|(\mathbf{A} - \mathbf{B})\mathbf{x}\|_2$ and by using a version of the Cauchy-Schwartz inequality that encompasses norms of matrices, we obtain $\|\mathbf{Ax}\|_2 \leq \|\mathbf{A} - \mathbf{B}\|_2 \|\mathbf{x}\|_2 < \sigma_{k+2} \|\mathbf{x}\|_2$. [49]
 However, there exists a $(k + 1)$ -dimensional subspace where $\|\mathbf{Ax}\|_2 \geq \sigma_{k+1} \|\mathbf{x}\|_2$, which is spanned by the right-singular vectors \mathbf{v}_j , $j \leq k + 1$ of \mathbf{A} .
 Adding up dimensions of these two spaces yields a number greater than n , as there must be a nonzero vector in both spaces. This is a contradiction of the rank-nullity theorem [49]

Explaining the contradiction part:

[7]

- (a) $\text{rank}(\mathbf{B}) + \text{nullity}(\mathbf{B}) = n$
- (b) So if $\text{rank}(\mathbf{B}) < k$, then $\text{nullity} > n - k$.
- (c) Consider:
 - i. The subspace where $\mathbf{Bx} = \mathbf{0}$: null space of dimension $> n - k$
 - ii. The subspace spanned by $\mathbf{v}_{k+1}, \dots, \mathbf{v}_r$: has dimension $\geq n - k$
- (d) If both have more than $n - k$ dimensions, their intersection must be nonzero (they must share a direction). That means:
 - i. There exists a nonzero vector \mathbf{x} that's in both: in the null space of \mathbf{B} and in the span of $\mathbf{v}_{k+1}, \dots, \mathbf{v}_r$
 - ii. For that \mathbf{x} , we must have: $\|\mathbf{Ax}\|_2 \geq \sigma_{k+1}$. But this contradicts the earlier assumption that $\|\mathbf{Ax}\|_2 < \sigma_{k+1}$.
- (e) Thus, assuming that $\|\mathbf{A} - \mathbf{B}\|_2 < \sigma_{k+1}$ leads to a contradiction with rank-nullity. Therefore: $\|\mathbf{A} - \mathbf{B}\|_2 \geq \sigma_{k+1}$ which confirms the theorem.
- (f) **in Simple Terms:**
 - i. You're trying to approximate a matrix using fewer components (rank k).
 - ii. The best you can do (smallest error) is using the top k singular values from SVD.
 - iii. If you try to do better, you run into a contradiction with basic linear algebra — namely, the rank-nullity theorem.

- iv. So, SVD gives the best possible low-rank approximation, and the error is exactly the $(k + 1)$ -th singular value.
- 8. We can interpret the approximation of \mathbf{A} by a rank- k matrix as a form of **lossy compression**. [49]

CHAPTER 15

VECTOR SPACES

15.1. Vector Space

1.

Definition 15.1 (Vector Space). A real-valued vector space $V = (\mathcal{V}, +, \cdot)$ is a set \mathcal{V} with two operations: [49]

$$+ : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V} \quad [49]$$

$$\cdot : \mathbb{R} \times \mathcal{V} \rightarrow \mathcal{V} \quad [49]$$

2. addition (+) is inner operation: both operands must be from \mathcal{V} [49]
3. multiplication by scalars (\cdot) is outer operation: one operand is from \mathcal{V} , another from \mathbb{R} [49]
4. The elements $\mathbf{x} \in \mathcal{V}$ are called **vectors**. [49]

15.1.1. Properties of a vector Space

1. $(\mathcal{V}, +)$ is an Abelian group [49]
2. Distributivity:
 - (a) $\forall \lambda \in \mathbb{R}, \mathbf{x}, \mathbf{y} \in \mathcal{V} : \lambda \cdot (\mathbf{x} + \mathbf{y}) = \lambda \cdot \mathbf{x} + \lambda \cdot \mathbf{y}$ [49]
 - (b) $\forall \lambda, \psi \in \mathbb{R}, \mathbf{x} \in \mathcal{V} : (\lambda + \psi) \cdot \mathbf{x} = \lambda \cdot \mathbf{x} + \psi \cdot \mathbf{x}$ [49]
3. Associativity (outer operation): $\forall \lambda, \psi \in \mathbb{R}, \mathbf{x} \in \mathcal{V} : \lambda \cdot (\psi \cdot \mathbf{x}) = (\lambda \psi) \cdot \mathbf{x}$ [49]
4. Neutral element with respect to the outer operation: $\forall \mathbf{x} \in \mathcal{V} : 1 \cdot \mathbf{x} = \mathbf{x}$ [49]
5. The neutral element of $(\mathcal{V}, +)$ is the zero vector $\mathbf{0} = [0, \dots, 0]^\top$ [49]
6. A “vector multiplication” $\mathbf{ab}, \mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, is **not defined**. [49]

15.1.2. Dimension of a vector Space

1. the dimension of V is the number of basis vectors of V , and we write $\dim(V)$. [49]
2. The dimension of a vector space corresponds to the number of its basis vectors. [49]
3. Intuitively, the dimension of a vector space can be thought of as the number of independent directions in this vector space. [49]
4. The dimension of a vector space is not necessarily the number of elements in a vector. For instance, the vector space $V = \text{span}[\begin{bmatrix} 0 \\ 1 \end{bmatrix}]$ is one-dimensional, although the basis vector possesses two elements. [49]

```
1 import numpy as np
2
3 # Define 3 column vectors in R^4
4 v1 = np.array([1, 2, 3, 4]).reshape(-1, 1)
5 v2 = np.array([2, 4, 6, 8]).reshape(-1, 1)
6 v3 = np.array([0, 1, 0, 1]).reshape(-1, 1)
```

```

7
8 # Put them into a matrix as columns => stack column-wise (axis=1)
9 matrix = np.column_stack([v1, v2, v3]) # Shape: (4, 3)
10
11 # Find the dimension = rank of the matrix
12 dimension = np.linalg.matrix_rank(matrix)
13
14 print(f"Dimension of the vector space: {dimension}")

```

Python Snippet 15.1: Dimension of vector space - numPy

15.1.3. Norm

1.

Definition 15.2 (Norm). A norm on a vector space V is a function [49]

$$\|\cdot\| : V \rightarrow \mathbb{R} \text{ such that } \mathbf{x} \mapsto \|\mathbf{x}\| \quad [49]$$

which assigns each vector \mathbf{x} its length $\|\mathbf{x}\| \in \mathbb{R}$, such that for all $\lambda \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in V$ the following hold: [49]

(a) Absolutely homogeneous: $\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\|$ [49]

(b) Triangle inequality: $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ [49]

In geometric terms, the triangle inequality states that for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side. [49]

(c) Positive definite: $\|\mathbf{x}\| \geq 0$ and $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$ [49]

```

1 import numpy as np
2
3 def norm_general(_arr: np.ndarray):
4     """
5         Compute the norm using custom formula/ calculations
6     """
7     raise NotImplementedError()

```

Python Snippet 15.2: General Norm of a vector - numPy

15.1.4. General Inner Product ($\Omega(a, b)$)

1. A bilinear mapping Ω is a mapping with two arguments, and it is linear in each argument, i.e., when we look at a vector space V then it holds that for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V, \lambda, \psi \in \mathbb{R}$ that [49]

$$\Omega(\lambda \mathbf{x} + \psi \mathbf{y}, \mathbf{z}) = \lambda \Omega(\mathbf{x}, \mathbf{z}) + \psi \Omega(\mathbf{y}, \mathbf{z}) \quad [49]$$

$$\Omega(\mathbf{x}, \lambda \mathbf{y} + \psi \mathbf{z}) = \lambda \Omega(\mathbf{x}, \mathbf{y}) + \psi \Omega(\mathbf{x}, \mathbf{z}). \quad [49]$$

2. Let V be a vector space and $\Omega : V \times V \rightarrow \mathbb{R}$ be a bilinear mapping that takes two vectors and maps them onto a real number. Then [49]

(a) Ω is called **symmetric** if $\Omega(\mathbf{x}, \mathbf{y}) = \Omega(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in V$, i.e., the order of the arguments does not matter. [49]

(b) Ω is called **positive definite** if $\forall \mathbf{x} \in V \setminus \{\mathbf{0}\} : \Omega(\mathbf{x}, \mathbf{x}) > 0, \Omega(\mathbf{0}, \mathbf{0}) = 0$. [49]

15.1.5. Inner Product ($\langle \mathbf{x}, \mathbf{y} \rangle$)

1. Let V be a vector space and $\Omega : V \times V \rightarrow \mathbb{R}$ be a bilinear mapping that takes two vectors and maps them onto a real number. Then [49]

- (a) A positive definite, symmetric bilinear mapping $\Omega : V \times V \rightarrow \mathbb{R}$ is called an inner product on V . We typically write $\langle x, y \rangle$ instead of $\Omega(x, y)$. [49]
2. default inner product is dot product: $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$ [49]

15.1.5.1. Inner Product Space

1. Let V be a vector space and $\Omega : V \times V \rightarrow \mathbb{R}$ be a bilinear mapping that takes two vectors and maps them onto a real number. Then the pair $(V, \langle \cdot, \cdot \rangle)$ is called an **inner product space** or **(real) vector space with inner product**.

15.1.5.2. Inner Products of Random Variables

1. If we have two uncorrelated random variables X, Y , then $\mathbb{V}[x+y] = \mathbb{V}[x] + \mathbb{V}[y]$ [49]
2. Random variables can be considered vectors in a vector space, and we can define inner products to obtain geometric properties of random variables. [49]
3. If we define $\langle X, Y \rangle := \text{Cov}[x, y]$ for zero mean random variables X and Y , we obtain an inner product. [49]
4. The length of a random variable is $\|X\| = \sqrt{\text{Cov}[x, x]} = \sqrt{\mathbb{V}[x]} = \sigma[x]$ i.e., its standard deviation. The “longer” the random variable, the more uncertain it is; and a random variable with length 0 is deterministic. [49]
5. The angle θ between two random variables X, Y : $\cos \theta = \frac{\langle X, Y \rangle}{\|X\| \|Y\|} = \frac{\text{Cov}[x, y]}{\sqrt{\mathbb{V}[x] \mathbb{V}[y]}}$ which is the correlation between the two random variables [49]

15.1.6. Euclidean vector space

1. Let V be a vector space and $\Omega : V \times V \rightarrow \mathbb{R}$ be a bilinear mapping that takes two vectors and maps them onto a real number. Then if we use the dot product, we call $(V, \langle \cdot, \cdot \rangle)$ a Euclidean vector space.

15.1.7. Outer Product (\otimes)

1. Given a vector $\mathbf{x} \in \mathbb{R}^n$, we obtain the outer product $\mathbf{x}^m := \mathbf{x} \otimes \overbrace{\cdots \otimes \mathbf{x}}^{m \text{ times}} \in \mathbb{R}^{\underbrace{n \times n \times \cdots \times n}_{m \text{ times}}}$ [49]
2. Example: $\mathbf{x} \in \mathbb{R}^n \Rightarrow \mathbf{x}^2 = \mathbf{x} \otimes \mathbf{x} \in \mathbb{R}^{n \times n}$, $\mathbf{x}^2[i, j] = x[i]x[j]$ [49]

15.1.8. Lengths

1. Inner products and norms are closely related in the sense that any inner product induces a norm $\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ [49] in a natural way, such that we can compute lengths of vectors using the inner product. [49]
2. length depends on norm [7] different norms lead to different lengths [7]
3. Not every norm is induced by an inner product (eg: Manhattan norm). [49]
4. **Definition 15.3** (Cauchy-Schwarz Inequality). For an inner product vector space $(V, \langle \cdot, \cdot \rangle)$ the induced norm $\|\cdot\|$ satisfies the Cauchy-Schwarz inequality [49] $|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|$ [49]

5. Transformations by orthogonal matrices are special because the length of a vector \mathbf{x} is not changed when transforming it using an orthogonal matrix \mathbf{A} . For the dot product, we obtain [49]
- $$\|\mathbf{Ax}\|^2 = (\mathbf{Ax})^\top (\mathbf{Ax}) = \mathbf{x}^\top \mathbf{A}^\top \mathbf{Ax} = \mathbf{x}^\top \mathbf{I} \mathbf{x} = \mathbf{x}^\top \mathbf{x} = \|\mathbf{x}\|^2 \quad [49]$$

15.1.9. Distance

1. Consider an inner product space $(V, \langle \cdot, \cdot \rangle)$. Then [49]
 $d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\| = \sqrt{\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle}$ [49]
 is called the distance between \mathbf{x} and \mathbf{y} for $\mathbf{x}, \mathbf{y} \in V$. [49]
2. If we use the dot product as the inner product, then the distance is called **Euclidean distance**. [49]
3. Similar to the length of a vector, the distance between vectors does not require an inner product: a norm is sufficient. If we have a norm induced by an inner product, the distance may vary depending on the choice of the inner product. [49]
4. Every distance is a metric, but not every metric is necessarily defined as a distance using a norm or inner product. [7]
5. Transformations with orthogonal matrices preserve distances. [49]

15.1.10. Metric

- 1.
- Definition 15.4** (Metric). The mapping $d : V \times V \rightarrow \mathbb{R}$, $(\mathbf{x}, \mathbf{y}) \mapsto d(\mathbf{x}, \mathbf{y})$ is called a **metric**. [49]
2. A metric d satisfies the following: [49]
 - (a) d is **positive definite**, i.e., $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}, \mathbf{y} \in V$ and $d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$. [49]
 - (b) d is **symmetric**, i.e., $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in V$. [49]
 - (c) **Triangle inequality**: $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$. [49]

15.1.11. Inner Product VS Metric

1. the lists of properties of inner products and metrics look very similar. [49]
2. $\langle \mathbf{x}, \mathbf{y} \rangle$ and $d(\mathbf{x}, \mathbf{y})$ behave in opposite directions. [49]
3. Very similar \mathbf{x} and \mathbf{y} will result in a large value for the inner product and a small value for the metric. [49]

15.1.12. Angles

1. We use the Cauchy-Schwarz inequality to define angles ω in inner product spaces between two vectors \mathbf{x}, \mathbf{y} , and this notion coincides with our intuition in \mathbb{R}^2 and \mathbb{R}^3 . Assume that $\mathbf{x} \neq \mathbf{0}$, $\mathbf{y} \neq \mathbf{0}$. Then $-1 \leq \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} \leq 1$. Therefore, there exists a unique $\omega \in [0, \pi]$ with $\cos(\omega) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}$ [49]
 2. The number ω is the angle between the vectors \mathbf{x} and \mathbf{y} . [49]
 3. Intuitively, the angle between two vectors tells us how similar their orientations are. [49]
 4. Transformations with orthogonal matrices preserve angles. [49]
- The angle between any two vectors \mathbf{x}, \mathbf{y} , as measured by their inner product, is also unchanged when transforming both of them using an orthogonal matrix \mathbf{A} . Assuming the dot product as the inner product, the angle of the images \mathbf{Ax} and \mathbf{Ay} is given as [49]

$$\cos \omega = \frac{(\mathbf{Ax})^\top (\mathbf{Ax})}{\|\mathbf{Ax}\| \|\mathbf{Ax}\|} = \frac{\mathbf{x}^\top \mathbf{A}^\top \mathbf{A} \mathbf{y}}{\sqrt{\mathbf{x}^\top \mathbf{A}^\top \mathbf{A} \mathbf{x} \mathbf{y}^\top \mathbf{A}^\top \mathbf{A} \mathbf{y}}} = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

which gives exactly the angle between \mathbf{x} and \mathbf{y} .

[49]

ω	$\cos(\omega)$	Interpretation
0	1	orientation is same
$\pi/2$	0	perpendicular
π	-1	opposite direction

15.1.13. Orthogonal Complement (U^\perp)

- Consider a D -dimensional vector space V and an M -dimensional subspace $U \subseteq V$. Then its orthogonal complement U^\perp is a $(D - M)$ -dimensional subspace of V and contains all vectors in V that are orthogonal to every vector in U . Furthermore, $U \cap U^\perp = \{\mathbf{0}\}$ so that any vector $\mathbf{x} \in V$ can be uniquely decomposed into
- [49]

$$\mathbf{x} = \sum_{m=1}^M \lambda_m \mathbf{b}_m + \sum_{j=1}^{D-M} \psi_j \mathbf{b}_j^\perp \quad \lambda_m, \psi_j \in \mathbb{R}$$
[49]

where $(\mathbf{b}_1, \dots, \mathbf{b}_M)$ is a basis of U and $(\mathbf{b}_1^\perp, \dots, \mathbf{b}_{D-M}^\perp)$ is a basis of U^\perp .

[49]

- the orthogonal complement can also be used to describe a plane U (two-dimensional subspace) in a three-dimensional vector space.
 - the vector \mathbf{w} with $\|\mathbf{w}\| = 1$, which is orthogonal to the plane U , is the basis vector of U^\perp . All vectors that are orthogonal to \mathbf{w} must (by construction) lie in the plane U . The vector \mathbf{w} is called the **normal vector** of U .
 - orthogonal complements can be used to describe hyperplanes in n -dimensional vector and affine spaces.
- [49]

15.2. Vector Subspace/ linear subspace

- 1.

Definition 15.5 (Vector Subspace). Let $V = (\mathcal{V}, +, \cdot)$ be a vector space and $\mathcal{U} \subseteq \mathcal{V}, \mathcal{U} \neq \emptyset$. Then $\mathcal{U} = (\mathcal{U}, +, \cdot)$ is called vector subspace of V (or linear subspace) if \mathcal{U} is a vector space with the vector space operations $+$ and \cdot restricted to $\mathcal{U} \times \mathcal{U}$ and $\mathbb{R} \times \mathcal{U}$. We write $\mathcal{U} \subseteq V$ to denote a subspace \mathcal{U} of V .

[49]

- Intuitively, they are sets contained in the original vector space with the property that when we perform vector space operations on elements within this subspace, we will never leave it. In this sense, they are “closed”.
 - If $\mathcal{U} \subseteq \mathcal{V}$ and V is a vector space, then \mathcal{U} naturally inherits many properties directly from V because they hold for all $\mathbf{x} \in \mathcal{V}$, and in particular for all $\mathbf{x} \in \mathcal{U} \subseteq \mathcal{V}$. This includes the Abelian group properties, the distributivity, the associativity and the neutral element.
 - To determine whether $(\mathcal{U}, +, \cdot)$ is a subspace of V we still do need to show:
 - $\mathcal{U} \neq \emptyset$, in particular: $\mathbf{0} \in \mathcal{U}$
 - Closure of \mathcal{U} :
 - With respect to the outer operation: $\forall \lambda \in \mathbb{R} \forall \mathbf{x} \in \mathcal{U} : \lambda \mathbf{x} \in \mathcal{U}$.
 - With respect to the inner operation: $\forall \mathbf{x}, \mathbf{y} \in \mathcal{U} : \mathbf{x} + \mathbf{y} \in \mathcal{U}$.
- [49]

15.2.1. Dimension of a Vector Subspace

1. If $U \subseteq V$ is a subspace of V , then $\dim(U) \leq \dim(V)$ and $\dim(U) = \dim(V)$ if and only if $U = V$. [49]

15.2.2. Basis of a Vector Subspace

1. A basis of a subspace $U = \text{span}[\mathbf{x}_1, \dots, \mathbf{x}_m] \subseteq \mathbb{R}^n$ can be found by executing the following steps: [49]
 - (a) Write the spanning vectors as columns of a matrix \mathbf{A} [49]
 - (b) Determine the row-echelon form (REF) of \mathbf{A} . [49]
 - (c) The spanning vectors associated with the pivot columns are a basis of U . [49]

15.3. Vector

1. In general, vectors are special objects that can be added together and multiplied by scalars to produce another object of the same kind. From an abstract mathematical viewpoint, any object that satisfies these two properties can be considered a vector. [49]
2. By convention $(1, n)$ -matrices are called rows and $(m, 1)$ -matrices are called row columns. These special matrices are also called row/ column vectors. [49]
3. Geometric vectors directed line segments that start at the origin, then intuitively the length of a vector is the distance of the “end” of this directed line segment from the origin. [49]

15.3.1. Types of vectors

1. \mathbb{R}^n or $\mathbb{R}^{n \times 1}$: column vectors: $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ [49]
2. $\mathbb{R}^{1 \times n}$: row vectors: $\mathbf{x}^\top = [x_1 \quad \dots \quad x_n]$ [49]

```

1 import numpy as np
2
3 row_vec = np.array([1, 2, 3, 4]).reshape(1, -1)
4 print(row_vec)
5 print(row_vec.shape)
6
7 col_vec = np.array([1, 2, 3, 4]).reshape(-1, 1)
8 print(col_vec)
9 print(col_vec.shape)

```

Python Snippet 15.3: Row & Column vectors - numPy

Output:

```

1 [[1 2 3 4]]
2 (1, 4)
3 [[1]
4 [2]
5 [3]
6 [4]]
7 (4, 1)

```

15.3.2. Manhattan Norm/ ℓ_1 Norm

1. The Manhattan norm on \mathbb{R}^n is defined for $\mathbf{x} \in \mathbb{R}^n$ as [49]

$$\|\mathbf{x}\|_1 := \sum_{i=1}^n |x_i| \quad [49]$$

where $|\cdot|$ is the absolute value. [49]

```
1 import numpy as np
2
3 def norm_l1(_arr: np.ndarray):
4     return np.sum(np.abs(_arr))
```

Python Snippet 15.4: L1 Norm of a vector - numPy

15.3.3. Euclidean Norm/ ℓ_2 Norm

1. The Euclidean norm of $\mathbf{x} \in \mathbb{R}^n$ is defined as [49]

$$\|\mathbf{x}\|_2 := \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{\mathbf{x}^\top \mathbf{x}} \quad [49]$$

and computes the Euclidean distance of \mathbf{x} from the origin. [49]

2. default "norm", if not stated otherwise. [49]

```
1 import numpy as np
2
3 def norm_l2(_arr: np.ndarray):
4     return np.sqrt(np.sum(np.square(_arr)))
```

Python Snippet 15.5: L2 Norm of a vector - numPy

15.3.4. Outer Product (ab^\top)

1. $\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n, \mathbf{a}\mathbf{b}^\top \in \mathbb{R}^{n \times n}$ [49]

```
1 import numpy as np
2
3 a = np.array([1, 2, 3]).reshape(-1, 1) # Shape: (3, 1)
4 print(a)
5
6 b = np.array([4, 5]).reshape(1, -1)      # Shape: (1, 2)
7 print(b)
8
9 outer = a @ b
10 print(outer)
```

Python Snippet 15.6: Outer product - numPy

output:

```
1 [[1]
2  [2]
3  [3]]
4 [[4 5]]
5 [[ 4  5]
6  [ 8 10]
7  [12 15]]
```

15.3.5. Scalar/dot product ($a^\top b$)

1. $\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n, \mathbf{a}^\top \mathbf{b} = \sum_{i=1}^n a_i b_i \in \mathbb{R}$ [49]

```

1 import numpy as np
2
3 a = np.array([1, 2, 3])
4 b = np.array([4, 5, 6])
5
6 print(np.dot(a, b))    # 32
7 print(np.inner(a, b))  # 32
8 print(a @ b)          # 32
9 print(np.sum(a * b))  # 32

```

Python Snippet 15.7: Dot product - numPy

15.3.6. Orthogonal ($x \perp y$) & orthonormal vectors

- 1.
2. **Definition 15.6** (Orthogonal Vectors). Two vectors \mathbf{x} and \mathbf{y} are orthogonal if and only if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$, and we write $\mathbf{x} \perp \mathbf{y}$. [49]
3. If additionally $\|\mathbf{x}\| = 1 = \|\mathbf{y}\|$, i.e., the vectors are unit vectors, then \mathbf{x} and \mathbf{y} are orthonormal. [49]
4. **0**-vector is orthogonal to every vector in the vector space. [49]
5. Orthogonality is the generalization of the concept of perpendicularity to bilinear forms that do not have to be the dot product. [49]

15.3.7. Vector as a function

1. We can think of a vector $\mathbf{x} \in \mathbb{R}^n$ as a function with n function values. [49]
2. The concept of an inner product can be generalized to vectors with an infinite number of entries (countably infinite) and also continuous-valued functions (uncountably infinite). Then the sum over individual components of vectors (as in $\mathbf{a}^\top \mathbf{b} = \sum_{i=1}^n a_i b_i$) turns into an integral. [49]

15.3.7.1. Inner Product of Functions

1. An inner product of two functions $u : \mathbb{R} \rightarrow \mathbb{R}$ and $v : \mathbb{R} \rightarrow \mathbb{R}$ can be defined as the definite integral [49].

$$\langle u, v \rangle := \int_a^b u(x)v(x)dx$$
 [49]
 for lower and upper limits $a, b < \infty$, respectively. [49]
 2. As with our usual inner product, we can define norms and orthogonality by looking at the inner product. [49]
 3. If $\langle u, v \rangle = 0$, then the functions u and v are **orthogonal**. [49]
 4. unlike inner products on finite-dimensional vectors, inner products on functions may diverge (have infinite value). [49]
 5. It also holds that the collection of functions

$$\{1, \cos(x), \cos(2x), \cos(3x), \dots\}$$
 [49]
- is orthogonal if we integrate from $-\pi$ to π , i.e., any pair of functions are orthogonal to each other. The collection of functions spans a large subspace of the functions that are even and periodic on $[\pi, \pi]$, and projecting functions onto this subspace is the fundamental idea behind Fourier series. [49]

15.3.8. Collinearity and Codirection

1.

Definition 15.7 (Codirection). Two vectors that point in the same direction are called codirected. [49]

2.

Definition 15.8 (Collinearity). Two vectors are collinear if they point in the same or the opposite direction. [49]

15.4. Linear Combination

1.

Definition 15.9 (Linear Combination of vectors). Consider a vector space V and a finite number of vectors $\mathbf{x}_1, \dots, \mathbf{x}_k \in V$. Then, every $\mathbf{v} \in V$ of the form:

$$\mathbf{v} = \lambda_1 \mathbf{x}_1 + \dots + \lambda_k \mathbf{x}_k = \sum_{i=1}^k \lambda_i \mathbf{x}_i \in V$$

with $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ is a linear combination of the vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$. [49]

2. The **0**-vector can always be written as the linear combination of k vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$ because $\mathbf{0} = \sum_{i=1}^k 0 \mathbf{x}_i$ is always true. [49]

15.5. Affine Subspaces

1.

Definition 15.10 (Affine Subspaces). Let V be a vector space, $\mathbf{x}_0 \in V$ and $U \subseteq V$ a subspace. Then the subset [49]

$$L = \mathbf{x}_0 + U := \{\mathbf{x}_0 + \mathbf{u} : \mathbf{u} \in U\} = \{\mathbf{v} \in V \mid \exists \mathbf{u} \in U : \mathbf{v} = \mathbf{x}_0 + \mathbf{u}\} \subseteq V \quad [49]$$

is called **affine subspace** or **linear manifold** of V . U is called **direction** or **direction space**, and \mathbf{x}_0 is called **support point**. L is also known as **hyperplane**. [49]

2. Note that the definition of an affine subspace excludes **0** if $\mathbf{x}_0 \notin U$. Therefore, an affine subspace is not a (linear) subspace (vector subspace) of V for $\mathbf{x}_0 \notin U$. [49]

3. Consider two affine subspaces $L = \mathbf{x}_0 + U$ and $\tilde{L} = \tilde{\mathbf{x}}_0 + \tilde{U}$ of a vector space V . Then, $L \subseteq \tilde{L}$ if and only if $U \subseteq \tilde{U}$ and $\mathbf{x}_0 - \tilde{\mathbf{x}}_0 \in \tilde{U}$. [49]

4. Affine subspaces are often described by parameters: Consider a k -dimensional affine space $L = \mathbf{x}_0 + U$ of V . If $(\mathbf{b}_1, \dots, \mathbf{b}_k)$ is an ordered basis of U , then every element $\mathbf{x} \in L$ can be uniquely described as [49]

$$\mathbf{x} = \mathbf{x}_0 + \lambda_1 \mathbf{b}_1 + \dots + \lambda_k \mathbf{b}_k \quad [49]$$

where $\lambda_1, \dots, \lambda_k \in \mathbb{R}$. This representation is called parametric equation of L with directional vectors $\mathbf{b}_1, \dots, \mathbf{b}_k$ and parameters $\lambda_1, \dots, \lambda_k$. [49]

5. One-dimensional affine subspaces are called **lines** and can be written as $\mathbf{y} = \mathbf{x}_0 + \lambda \mathbf{b}_1$, where $\lambda \in \mathbb{R}$ and $U = \text{span}[\mathbf{b}_1] \subseteq \mathbb{R}^n$ is a one-dimensional subspace of \mathbb{R}^n . This means that a line is defined by a support point \mathbf{x}_0 and a vector \mathbf{b}_1 that defines the direction. [49]

6. Two-dimensional affine subspaces of \mathbb{R}^n are called **planes**. The parametric equation for planes is $\mathbf{y} = \mathbf{x}_0 + \lambda_1 \mathbf{b}_1 + \lambda_2 \mathbf{b}_2$, where $\lambda_1, \lambda_2 \in \mathbb{R}$ and $U = \text{span}[\mathbf{b}_1, \mathbf{b}_2] \subseteq \mathbb{R}^n$. This means that a plane is defined by a support point \mathbf{x}_0 and two linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2$ that span the direction space. [49]

7. In \mathbb{R}^n , the $(n-1)$ -dimensional affine subspaces are called **hyperplanes**, and the corresponding parametric equation is $\mathbf{y} = \mathbf{x}_0 + \sum_{i=1}^{n-1} \lambda_i \mathbf{b}_i$, where $\mathbf{b}_1, \dots, \mathbf{b}_{n-1}$ form a basis of an $(n-1)$ -dimensional subspace U

of \mathbb{R}^n . This means that a hyperplane is defined by a support point \mathbf{x}_0 and $(n - 1)$ linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_{n-1}$ that span the direction space. In \mathbb{R}^2 , a line is also a hyperplane. In \mathbb{R}^3 , a plane is also a hyperplane. [49]

15.6. Linear (In)dependence

1.

Definition 15.11 (Linear (In)dependence). Let us consider a vector space V with $k \in \mathbb{N}$ and $\mathbf{x}_1, \dots, \mathbf{x}_k \in V$. If there is a non-trivial linear combination, such that $\mathbf{0} = \sum_{i=1}^k \lambda_i \mathbf{x}_i$ with at least one $\lambda_i \neq 0$, the vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$ are linearly dependent. [49]

2. If only the trivial solution exists, i.e., $\lambda_1 = \dots = \lambda_k = 0$ the vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$ are linearly independent. [49]
3. Intuitively, a set of linearly independent vectors consists of vectors that have no redundancy, i.e., if we remove any of those vectors from the set, we will lose something. [49]

15.6.1. Properties of Linear (In)dependence

1. k vectors are either linearly dependent or linearly independent. There is no third option. [49]
2. If at least one of the vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$ is $\mathbf{0}$ then they are linearly dependent. The same holds if two vectors are identical. [49]
3. The vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_k : \mathbf{x}_i \neq \mathbf{0}, i = 1, \dots, k\}$, $k \geq 2$, are linearly dependent if and only if (at least) one of them is a linear combination of the others. In particular, if one vector is a multiple of another vector, i.e., $\mathbf{x}_i = \lambda \mathbf{x}_j, \lambda \in \mathbb{R}$ then the set $\{\mathbf{x}_1, \dots, \mathbf{x}_k : \mathbf{x}_i \neq \mathbf{0}, i = 1, \dots, k\}$ is linearly dependent. [49]
4. A practical way of checking whether vectors $\mathbf{x}_1, \dots, \mathbf{x}_k \in V$ are linearly independent is to use Gaussian elimination: Write all vectors as columns of a matrix A and perform Gaussian elimination until the matrix is in row echelon form (the reduced row-echelon form is unnecessary here):
 - (a) The pivot columns indicate the vectors, which are linearly independent of the vectors on the left. Note that there is an ordering of vectors when the matrix is built. [49]
 - (b) The non-pivot columns can be expressed as linear combinations of the pivot columns on their left. [49]

All column vectors are linearly independent if and only if all columns are pivot columns. If there is at least one non-pivot column, the columns (and, therefore, the corresponding vectors) are linearly dependent. [49]

5. Consider a vector space V with k linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_k$ and m linear combinations: [49]

$$\mathbf{x}_1 = \sum_{i=1}^k \lambda_{1i} \mathbf{b}_i, \quad \dots, \quad \mathbf{x}_m = \sum_{i=1}^k \lambda_{mi} \mathbf{b}_i. \quad [49]$$

Defining $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ as the matrix whose columns are the linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_k$, we can write: [49]

$$\mathbf{x}_j = \mathbf{B} \boldsymbol{\lambda}_j, \quad \boldsymbol{\lambda}_j = \begin{bmatrix} \lambda_{1j} \\ \vdots \\ \lambda_{kj} \end{bmatrix}, \quad j = 1, \dots, m. \quad [49]$$

in a more compact form.

- (a) to test whether $\mathbf{x}_1, \dots, \mathbf{x}_m$ are linearly independent: [49]

- i. general approach of testing: $\sum_{j=1}^m \psi_j \mathbf{x}_j = 0$ [49]
 - ii. $\sum_{j=1}^m \psi_j \mathbf{x}_j = 0 \Rightarrow \sum_{j=1}^m \psi_j \mathbf{B} \lambda_j = 0 \Rightarrow \mathbf{B} \sum_{j=1}^m \psi_j \lambda_j = 0$ [49]
 - iii. This means that $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ are linearly independent if and only if the column vectors $\{\lambda_1, \dots, \lambda_m\}$ are linearly independent. [49]
- (b) In a vector space V , m linear combinations of k vectors $\mathbf{x}_1, \dots, \mathbf{x}_k$ are linearly dependent if $m > k$. [49]

15.7. Generating Set

1.

Definition 15.12 (Generating Set). Consider a vector space $V = (\mathcal{V}, +, \cdot)$ and set of vectors $\mathcal{A} = \{\mathbf{x}_1, \dots, \mathbf{x}_k\} \subseteq \mathcal{V}$. If every vector $\mathbf{v} \in \mathcal{V}$ can be expressed as a linear combination of $\mathbf{x}_1, \dots, \mathbf{x}_k$, \mathcal{A} is called a generating set generating set of V . [49]

2. Generating sets are sets of vectors that span vector (sub)spaces, i.e., every vector can be represented as a linear combination of the vectors in the generating set. [49]

15.8. Span

1.

Definition 15.13 (Span). Consider a vector space $V = (\mathcal{V}, +, \cdot)$ and set of vectors $\mathcal{A} = \{\mathbf{x}_1, \dots, \mathbf{x}_k\} \subseteq \mathcal{V}$. The set of all linear combinations of vectors in \mathcal{A} is span called the span of \mathcal{A} . [49]

2. If \mathcal{A} spans the vector space V , we write $V = \text{span}[\mathcal{A}]$ or $V = \text{span}[\mathbf{x}_1, \dots, \mathbf{x}_k]$. [49]

15.9. Basis

1.

Definition 15.14 (Basis). Consider a vector space $V = (\mathcal{V}, +, \cdot)$ and $\mathcal{A} \subseteq \mathcal{V}$. A generating set \mathcal{A} of V is called minimal if there exists no smaller set $\tilde{\mathcal{A}} \subsetneq \mathcal{A} \subseteq \mathcal{V}$ that spans V . Every linearly independent generating set of V is minimal and is called a basis of V . [49]

2. Let $V = (\mathcal{V}, +, \cdot)$ be a vector space and $\mathcal{B} \subseteq \mathcal{V}, \mathcal{B} \neq \emptyset$. Then, the following statements are equivalent: [49]

- (a) \mathcal{B} is a basis of V . [49]
- (b) \mathcal{B} is a minimal generating set. [49]
- (c) \mathcal{B} is a maximal linearly independent set of vectors in V , i.e., adding any other vector to this set will make it linearly dependent. [49]
- (d) Every vector $\mathbf{x} \in V$ is a linear combination of vectors from \mathcal{B} , and every linear combination is unique, i.e., with

$$\mathbf{x} = \sum_{i=1}^k \lambda_i \mathbf{b}_i = \sum_{i=1}^k \psi_i \mathbf{b}_i$$

and $\lambda_i, \psi_i \in \mathbb{R}, \mathbf{b}_i \in \mathcal{B}$ it follows that $\lambda_i = \psi_i, i = 1, \dots, k$. [49]

- 3. Every vector space V possesses a basis \mathcal{B} . [49]
- 4. There can be many bases of a vector space V , i.e., there is **no unique basis**. However, all bases possess the same number of elements, the **basis vectors**. [49]

5. Unordered basis: $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ [49]
 6. Ordered basis: $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ [49]
 7. matrix whose columns are the vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$: $B = [\mathbf{b}_1, \dots, \mathbf{b}_n]$ [49]

15.9.1. Checking Basis

To check whether two given sets of basis vectors (say \mathcal{B}_1 and \mathcal{B}_2) span the same vector space: [7]

1. **Method 1:** For each $\mathbf{b}_1 \in \mathcal{B}_1$, check if there exist scalars $\lambda_i \in \mathbb{R}$ such that [7]

$$\mathbf{b}_1 = \sum_{\mathbf{b}_{2i} \in \mathcal{B}_2} \lambda_i \mathbf{b}_{2i} \quad [7]$$

and vice versa (i.e., every vector in \mathcal{B}_2 can be written as a linear combination of vectors in \mathcal{B}_1). If both conditions hold, then \mathcal{B}_1 and \mathcal{B}_2 span the same vector space. [7]

2. **Method 2:** Let \mathcal{B}_c be the matrix formed by concatenating all vectors from \mathcal{B}_1 and \mathcal{B}_2 as columns. If [7]

$$\dim(\text{span}(\mathcal{B}_1)) = \dim(\text{span}(\mathcal{B}_2)) = \dim(\text{span}(\mathcal{B}_c)) \quad [7]$$

then both sets span the same vector space. [7]

```

1 import numpy as np
2
3 # Define basis sets as lists of vectors (1D arrays)
4 B1 = [np.array([1, 0]), np.array([0, 1])] # Basis set 1
5 B2 = [np.array([2, 1]), np.array([1, 2])] # Basis set 2
6
7 # Convert basis sets to matrices for method 2 when needed
8 def to_matrix(basis_set):
9     return np.column_stack(basis_set)
10
11 # Method 1: Check linear combination (span equivalence)
12 def can_span(basis_from, basis_to):
13     A = to_matrix(basis_to)
14     for v in basis_from:
15         # x: solution vector (coefficients for linear combination)
16         # residuals: squared error ||Ax - v||^2 (empty if underdetermined)
17         # rank: number of linearly independent columns in A
18         # s: singular values of A
19         x, residuals, rank, s = np.linalg.lstsq(A, v, rcond=None)
20
21         if not np.allclose(A @ x, v):
22             return False
23
24     return True
25
26 def same_vector_space_method1(B1, B2):
27     return can_span(B1, B2) and can_span(B2, B1)
28
29 # Method 2: Compare ranks of spans
30 def same_vector_space_method2(B1, B2):
31     M1 = to_matrix(B1)
32     M2 = to_matrix(B2)
33     M_combined = to_matrix(B1 + B2)
34
35     rank1 = np.linalg.matrix_rank(M1)
36     rank2 = np.linalg.matrix_rank(M2)

```

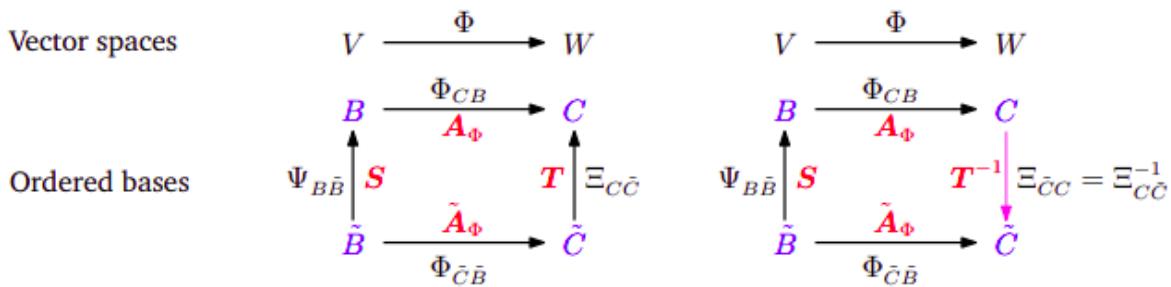
```

37 rank_combined = np.linalg.matrix_rank(M_combined)
38
39     return rank1 == rank2 == rank_combined
40
41 # Run both methods
42 print(same_vector_space_method1(B1, B2))          # True
43 print(same_vector_space_method2(B1, B2))          # True

```

Python Snippet 15.8: Check basis sets span same vector space - numPy

15.9.2. Basis Change



For a homomorphism $\Phi : V \rightarrow W$ and ordered bases B, \tilde{B} of V and C, \tilde{C} of W (marked in blue), we can express the mapping $\Phi_{\tilde{C}\tilde{B}}$ with respect to the bases \tilde{B}, \tilde{C} equivalently as a composition of the homomorphisms $\Phi_{\tilde{C}\tilde{B}} = \Xi_{\tilde{C}C} \circ \Phi_{CB} \circ \Psi_{B\tilde{B}}$ with respect to the bases in the subscripts.

The corresponding transformation matrices are in red.

We use $\Psi_{B\tilde{B}} = \text{id}_V$ and $\Xi_{CC} = \text{id}_W$, i.e., the identity mappings that map vectors onto themselves, but with respect to a different basis. [49]

1.

Theorem 15.1 (Basis Change). For a linear mapping $\Phi : V \rightarrow W$, ordered bases [49]

$$B = (\mathbf{b}_1, \dots, \mathbf{b}_n), \tilde{B} = (\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n) \quad [49]$$

of V and [49]

$$C = (\mathbf{c}_1, \dots, \mathbf{c}_m), \tilde{C} = (\tilde{\mathbf{c}}_1, \dots, \tilde{\mathbf{c}}_m) \quad [49]$$

of W , and a transformation matrix A_Φ of Φ with respect to B and C , the corresponding transformation matrix \tilde{A}_Φ with respect to the bases \tilde{B} and \tilde{C} is given as $\tilde{A}_\Phi = T^{-1} A_\Phi S$. [49]

Here, $S \in \mathbb{R}^{n \times n}$ is the transformation matrix of id_V that maps coordinates with respect to \tilde{B} onto coordinates with respect to B , and $T \in \mathbb{R}^{m \times m}$ is the transformation matrix of id_W that maps coordinates with respect to \tilde{C} onto coordinates with respect to C . [49]

Proof. we can write the vectors of the new basis \tilde{B} of V as a linear combination of the basis vectors of B , such that: [49]

$$\tilde{\mathbf{b}}_j = s_{1j} \mathbf{b}_1 + \dots + s_{nj} \mathbf{b}_n = \sum_{i=1}^n s_{ij} \mathbf{b}_i, j = 1, \dots, n \quad [49]$$

Similarly, we write the new basis vectors \tilde{C} of W as a linear combination of the basis vectors of C , which yields [49]

$$\tilde{\mathbf{c}}_k = t_{1k} \mathbf{c}_1 + \dots + t_{mk} \mathbf{c}_m = \sum_{l=1}^m t_{lk} \mathbf{c}_l, k = 1, \dots, m \quad [49]$$

We define $S = ((s_{ij})) \in \mathbb{R}^{n \times n}$ as the transformation matrix that maps coordinates with respect to \tilde{B} onto coordinates with respect to B and $T = ((t_{lk})) \in \mathbb{R}^{m \times m}$ as the transformation matrix that maps coordinates with respect to \tilde{C} onto coordinates with respect to C . In particular, the j -th column of S is the coordinate representation of $\tilde{\mathbf{b}}_j$ with respect to B and the k -th column of T is the coordinate representation of $\tilde{\mathbf{c}}_k$ with respect to C . Note that both S and T are regular. [49]

We are going to look at $\Phi(\tilde{\mathbf{b}}_j)$ from two perspectives. [49]

(a) Applying the mapping Φ , we get that for all $j = 1, \dots, n$ [49]

$$\Phi(\tilde{\mathbf{b}}_j) = \sum_{k=1}^m \tilde{a}_{kj} \underbrace{\tilde{\mathbf{c}}_k}_{\in W} = \sum_{k=1}^m \tilde{a}_{kj} \sum_{l=1}^m t_{lk} \mathbf{c}_l = \sum_{l=1}^m \left(\sum_{k=1}^m t_{lk} \tilde{a}_{kj} \right) \mathbf{c}_l \quad [49]$$

where we first expressed the new basis vectors $\tilde{\mathbf{c}}_k \in W$ as linear combinations of the basis vectors $\mathbf{c}_l \in W$ and then swapped the order of summation. [49]

(b) when we express the $\tilde{\mathbf{b}}_j \in V$ as linear combinations of $\mathbf{b}_j \in V$, we arrive at ($j = 1, \dots, n$): [49]

$$\Phi(\tilde{\mathbf{b}}_j) = \Phi \left(\sum_{i=1}^n s_{ij} \mathbf{b}_i \right) = \sum_{i=1}^n s_{ij} \Phi(\mathbf{b}_i) = \sum_{i=1}^n s_{ij} \sum_{l=1}^m a_{li} \mathbf{c}_l = \sum_{l=1}^m \left(\sum_{i=1}^n s_{ij} a_{li} \right) \mathbf{c}_l \quad [49]$$

$$(c) \sum_{k=1}^m t_{lk} \tilde{a}_{kj} = \sum_{i=1}^n s_{ij} a_{li} \Rightarrow \mathbf{T} \tilde{\mathbf{A}}_\Phi = \mathbf{A}_\Phi \mathbf{S} \in \mathbb{R}^{m \times n} \Rightarrow \tilde{\mathbf{A}}_\Phi = \mathbf{T}^{-1} \mathbf{A}_\Phi \mathbf{S} \quad [49]$$

□

15.9.3. Orthonormal Basis (ONB) & Orthogonal Basis (OGB)

1.

Definition 15.15 (Orthonormal Basis (ONB)). Consider an n -dimensional vector space V and a basis $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ of V . If [49]

for all $i, j = 1, \dots, n$ then the basis is called an orthonormal basis (ONB). [49]

2. If only $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0$ is satisfied, then the basis is called an **orthogonal basis**. [49]

3. Note: $\langle \mathbf{b}_i, \mathbf{b}_i \rangle = 1$ implies that every basis vector has length/norm 1. [49]

15.9.3.1. Orthogonalization using Gaussian Elimination

1. constructive way to iteratively build an orthonormal basis [49]

2. Assume we are given a set $\{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$ of non-orthogonal and un-normalized basis vectors. [49]

3. We concatenate them into a matrix $\tilde{\mathbf{B}} = [\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n]$ and apply Gaussian elimination to the augmented matrix $[\tilde{\mathbf{B}} \tilde{\mathbf{B}}^\top | \tilde{\mathbf{B}}]$ to obtain an orthonormal basis. [49]

```

1 import numpy as np
2 from scipy.linalg import orth
3
4 def gram_schmidt_from_augmented_matrix(basis_vectors):
5     """
6         Implements Gram-Schmidt as described via augmented matrix [B B^T | B]
7
8     Parameters:
9         basis_vectors (list of np.ndarray): List of non-orthogonal
10            basis vectors (each of same length)
11
12     Returns:
13         np.ndarray: Matrix with orthonormal basis vectors as columns
14     """
15     # Step 1: Form B matrix from input list
16     B = np.column_stack(basis_vectors) # Shape: (d, n)
17
18     # Step 2: Form the augmented matrix [B B^T | B]
19     BB_T = B @ B.T # Shape: (d, d)
20     augmented = np.hstack((BB_T, B)) # Shape: (d, d + n)

```

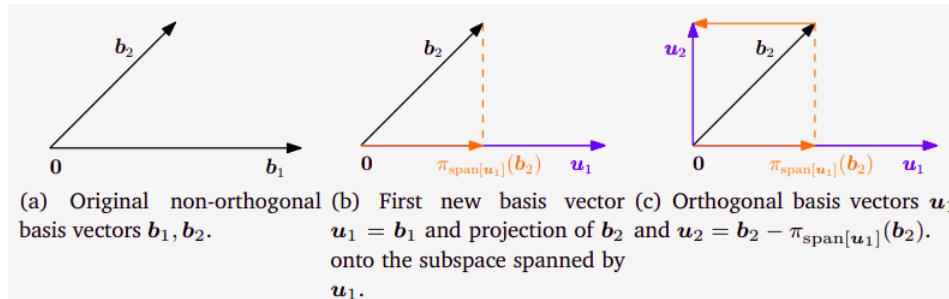
```

21
22 # Step 3: Apply Gaussian elimination to the augmented matrix
23 # Instead of manual row ops, we extract the column space of B using orth()
24 # This gives us an orthonormal basis for the column space of B
25 Q = orth(B) # Shape: (d, r), where r = rank(B)
26
27 return Q # Orthonormal basis matrix
28
29
30 B_tilde = [
31     np.array([1, 1, 0]),
32     np.array([1, 0, 1]),
33     np.array([0, 1, 1])
34 ]
35
36 Q = gram_schmidt_from_augmented_matrix(B_tilde)
37
38 print("Orthonormal basis vectors:")
39 for i in range(Q.shape[1]):
40     print(f"b_{i+1} =", Q[:, i])

```

Python Snippet 15.9: Orthogonalization using Gaussian Elimination - numPy

15.9.3.2. Gram-Schmidt Orthogonalization



Gram-Schmidt orthogonalization.

[\[49\]](#)

 (a) non-orthogonal basis $(\mathbf{b}_1, \mathbf{b}_2)$ of \mathbb{R}^2
[\[49\]](#)

 (b) first constructed basis vector \mathbf{u}_1 and orthogonal projection of \mathbf{b}_2 onto $\text{span}[\mathbf{u}_1]$;

[\[49\]](#)

 (c) orthogonal basis $(\mathbf{u}_1, \mathbf{u}_2)$ of \mathbb{R}^2 .

[\[49\]](#)

1. Method 1

[\[49\]](#)

Projections are at the core of the Gram-Schmidt method that allows us to constructively transform any basis $(\mathbf{b}_1, \dots, \mathbf{b}_n)$ of an n -dimensional vector space V into an orthogonal/orthonormal basis $(\mathbf{u}_1, \dots, \mathbf{u}_n)$ of V . This basis always exists and $\text{span}[\mathbf{b}_1, \dots, \mathbf{b}_n] = \text{span}[\mathbf{u}_1, \dots, \mathbf{u}_n]$.

[\[49\]](#)

$$(a) \mathbf{u}_1 := \mathbf{b}_1$$

[\[49\]](#)

$$(b) \mathbf{u}_k := \mathbf{b}_k - \pi_{\text{span}[\mathbf{u}_1, \dots, \mathbf{u}_{k-1}]}(\mathbf{b}_k) \quad , k = 2, \dots, n$$

[\[49\]](#)

the k -th basis vector \mathbf{b}_k is projected onto the subspace spanned by the first $k-1$ constructed orthogonal vectors $\mathbf{u}_1, \dots, \mathbf{u}_{k-1}$. If we normalize the \mathbf{u}_k , we obtain an ONB where $\|\mathbf{u}_k\| = 1$ for $k = 1, \dots, n$

2. Method 2

[\[51\]](#)

$$(a) \text{proj}_{\mathbf{u}}(\mathbf{b}) = \frac{\langle \mathbf{v}, \mathbf{b} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u}$$

[\[51\]](#)

$$\begin{aligned}
 \mathbf{u}_1 &= \mathbf{b}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
 \mathbf{u}_2 &= \mathbf{b}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{b}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
 \mathbf{u}_3 &= \mathbf{b}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{b}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{b}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
 \text{(b)} \quad \mathbf{u}_4 &= \mathbf{b}_4 - \text{proj}_{\mathbf{u}_1}(\mathbf{b}_4) - \text{proj}_{\mathbf{u}_2}(\mathbf{b}_4) - \text{proj}_{\mathbf{u}_3}(\mathbf{b}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
 &\vdots & &\vdots \\
 \mathbf{u}_k &= \mathbf{b}_k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}_j}(\mathbf{b}_k), & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}
 \end{aligned} \tag{51}$$

The sequence $\mathbf{u}_1, \dots, \mathbf{u}_k$ is the required system of orthogonal vectors, and the normalized vectors $\mathbf{e}_1, \dots, \mathbf{e}_k$ form an orthonormal set. [51]

```

1 import numpy as np
2
3 def gram_schmidt(vectors, inner_product=None):
4     """
5         Perform Gram-Schmidt orthogonalization on a list of vectors.
6
7     Parameters:
8         vectors (list of np.ndarray): List of n vectors in R^d
9             (each a 1D array).
10    inner_product (function): Function to
11        compute inner product, default is np.dot.
12
13 Returns:
14     orthogonal_basis (list of np.ndarray): Orthogonal
15         (not normalized) basis vectors.
16     """
17
18     if inner_product is None:
19         inner_product = lambda u, v: np.dot(u, v)
20
21     orthogonal_basis = []
22
23     for i, v in enumerate(vectors):
24         v_proj = np.zeros_like(v)
25         for u in orthogonal_basis:
26             proj_coeff = inner_product(v, u) / inner_product(u, u)
27             v_proj += proj_coeff * u
28         u_i = v - v_proj
29         orthogonal_basis.append(u_i)
30
31     return orthogonal_basis
32
33
34 def normalize_vectors(vectors, norm=None):
35     """
36         Normalize a list of vectors using the given norm.
37
38     Parameters:
39         vectors (list of np.ndarray): List of vectors to normalize.

```

Chapter 15. Vector Spaces

```
40         norm (function): Norm function, default is np.linalg.norm.
41
42     Returns:
43         list of np.ndarray: Normalized vectors.
44     """
45
46     if norm is None:
47         norm = lambda v: np.linalg.norm(v)
48     return [v / norm(v) for v in vectors]
49
50
51 # Define basis vectors in R^2
52 b1 = np.array([1.0, 1.0])
53 b2 = np.array([1.0, 0.0])
54
55 basis = [b1, b2]
56
57 # Use standard inner product and norm
58 orthogonal_basis = gram_schmidt(basis)
59 orthonormal_basis = normalize_vectors(orthogonal_basis)
60
61 print("Orthogonal basis:")
62 for v in orthogonal_basis:
63     print(v)
64
65 print("\nOrthonormal basis:")
66 for v in orthonormal_basis:
67     print(v)
68 ###### custom inner product/ norm
69
70 # Weighted inner product with weight matrix W
71 W = np.array([[2, 0], [0, 3]])
72 def weighted_inner(u, v):
73     return np.dot(u.T, W @ v)
74
75 # Weighted norm (induced by W)
76 def weighted_norm(v):
77     return np.sqrt(weighted_inner(v, v))
78
79 orthogonal_basis = gram_schmidt(basis, inner_product=weighted_inner)
80 orthonormal_basis = normalize_vectors(orthogonal_basis, norm=weighted_norm)
81
82 print("Orthogonal basis:")
83 for v in orthogonal_basis:
84     print(v)
85
86 print("\nOrthonormal basis:")
87 for v in orthonormal_basis:
88     print(v)
```

Python Snippet 15.10: Gram-Schmidt Orthogonalization & Orthonormalization - numPy

output:

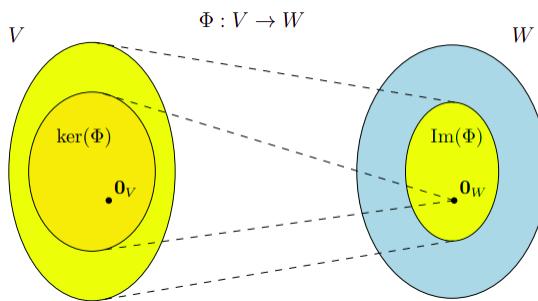
```
1 Orthogonal basis:
2 [1. 1.]
```

```

3 [ 0.5 -0.5]
4
5 Orthonormal basis:
6 [0.70710678 0.70710678]
7 [ 0.70710678 -0.70710678]
8
9 Orthogonal basis:
10 [1. 1.]
11 [ 0.6 -0.4]
12
13 Orthonormal basis:
14 [0.4472136 0.4472136]
15 [ 0.54772256 -0.36514837]

```

15.10. Linear Mappings (Φ)



1.

Definition 15.16 (Linear Mapping). For vector spaces V, W , a mapping $\Phi : V \rightarrow W$ is called a linear mapping (or vector space homomorphism/ linear transformation) if [49]

$$\forall \mathbf{x}, \mathbf{y} \in V \forall \lambda, \psi \in \mathbb{R} : \Phi(\lambda \mathbf{x} + \psi \mathbf{y}) = \lambda \Phi(\mathbf{x}) + \psi \Phi(\mathbf{y}) \quad [49]$$

2. we can represent linear mappings as matrices [49]
3. For linear mappings $\Phi : V \rightarrow W$ and $\Psi : W \rightarrow X$, the mapping $\Psi \circ \Phi : V \rightarrow X$ is also linear. [49]
4. If $\Phi : V \rightarrow W, \Psi : V \rightarrow W$ are linear, then $\Phi + \Psi$ and $\lambda \Phi, \lambda \in \mathbb{R}$, are linear, too. [49]

15.10.1. Matrix Representation of Linear Mappings

1. Any n -dimensional vector space is isomorphic to \mathbb{R}^n . [49]
2. We consider a basis $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ of an n -dimensional vector space V . [49]

15.10.2. Image/Range & Kernel/Null space

1.

Definition 15.17 (Image/ Range). For $\Phi : V \rightarrow W$, we define the image/range [49]

$$\text{Im}(\Phi) := \Phi(V) = \{\mathbf{w} \in W \mid \exists \mathbf{v} \in V : \Phi(\mathbf{v}) = \mathbf{w}\} \quad [49]$$

2.

Definition 15.18 (Kernel/ Null Space). For $\Phi : V \rightarrow W$, we define the kernel/null space [49]

$$\ker(\Phi) := \Phi^{-1}(\mathbf{0}_W) = \{\mathbf{v} \in V : \Phi(\mathbf{v}) = \mathbf{0}_W\} \quad [49]$$

3. We also call V and W also the **domain** and **codomain** of Φ , respectively. [49]

4. The image is the set of vectors $w \in W$ that can be “reached” by Φ from any vector in V . [49]
5. Intuitively, the kernel is the set of vectors $v \in V$ that Φ maps onto the neutral element $\mathbf{0}_W \in W$. [49]
6. It always holds that $\Phi(\mathbf{0}_V) = \mathbf{0}_W$ and, therefore, $\mathbf{0}_V \in \ker(\Phi)$. In particular, the null space is never empty. [49]
7. $\text{Im}(\Phi) \subseteq W$ is a subspace of W , and $\ker(\Phi) \subseteq V$ is a subspace of V . [49]
8. Φ is injective (one-to-one) if and only if $\ker(\Phi) = \{\mathbf{0}\}$. [49]
- 9.

Theorem 15.2 (Rank-Nullity Theorem). For vector spaces V, W and a linear mapping $\Phi : V \rightarrow W$ it holds that $\dim(\ker(\Phi)) + \dim(\text{Im}(\Phi)) = \dim(V)$. [49]

- (a) The rank-nullity theorem is also referred to as the **fundamental theorem of linear mappings** [49]
- (b) If $\dim(\text{Im}(\Phi)) < \dim(V)$, then $\ker(\Phi)$ is non-trivial, i.e., the kernel contains more than $\mathbf{0}_V$ and $\dim(\ker(\Phi)) \geq 1$. [49]
- (c) If A_Φ is the transformation matrix of Φ with respect to an ordered basis and $\dim(\text{Im}(\Phi)) < \dim(V)$, then the system of linear equations $A_\Phi x = \mathbf{0}$ has infinitely many solutions. [49]
- (d) If $\dim(V) = \dim(W)$, then the following three-way equivalence holds:
 - i. Φ is injective
 - ii. Φ is surjective
 - iii. Φ is bijective
 since $\text{Im}(\Phi) \subseteq W$.

15.11. Affine Mapping

- 1.
- Definition 15.19** (Affine Mapping). For two vector spaces V, W , a linear mapping $\Phi : V \rightarrow W$, and $a \in W$, the mapping:
- $$\phi : V \rightarrow W \text{ such that } x \mapsto a + \Phi(x)$$
- is an affine mapping from V to W . The vector a is called the **translation vector** of ϕ . [49]
2. Every affine mapping $\phi : V \rightarrow W$ is also the composition of a linear mapping $\phi : V \rightarrow W$ and a translation $\tau : W \rightarrow W$ in W , such that $\phi = \tau \circ \phi$. The mappings ϕ and τ are uniquely determined. [49]
 3. The composition $\phi' \circ \phi$ of affine mappings $\phi : V \rightarrow W, \phi' : W \rightarrow X$ is affine. [49]
 4. Affine mappings keep the geometric structure invariant. They also preserve the dimension and parallelism. [49]

15.12. Injective, Surjective, Bijective Mappings

Consider a mapping $\Phi : \mathcal{V} \rightarrow \mathcal{W}$, where \mathcal{V}, \mathcal{W} can be arbitrary sets. Then Φ is called: [49]

- 1.
- Definition 15.20** (Injective). if $\forall x, y \in \mathcal{V} : \Phi(x) = \Phi(y) \Rightarrow x = y$ [49]
- 2.
- Definition 15.21** (Surjective). if $\Phi(\mathcal{V}) = \mathcal{W}$ [49]
- (a) If Φ is surjective, then every element in \mathcal{W} can be “reached” from \mathcal{V} using Φ . [49]

3.

Definition 15.22 (Bijective). if it is injective and surjective [49]

- (a) A bijective Φ can be “undone”, i.e., there exists a mapping $\Psi : W \rightarrow V$ so that $\Psi \circ \Phi(\mathbf{x}) = \mathbf{x}$. [49]
- (b) This mapping Ψ is then called the inverse of Φ and normally denoted by Φ^{-1} . [49]

15.13. Special cases of Linear mappings

Consider a mapping $\Phi : V \rightarrow W$, where V, W can be arbitrary vector spaces. Then: [49]

1.

Definition 15.23 (Isomorphism). $\Phi : V \rightarrow W$ linear and bijective [49]

(a)

Theorem 15.3 (isomorphic vector spaces). Finite-dimensional vector spaces V and W are isomorphic if and only if $\dim(V) = \dim(W)$. [49]

- (b) there exists a linear, bijective mapping between two vector spaces of the same dimension. Intuitively, this means that vector spaces of the same dimension are kind of the same thing, as they can be transformed into each other without incurring any loss. [49]
- (c) It gives the justification to treat $\mathbb{R}^{m \times n}$ (the vector space of $m \times n$ -matrices) and \mathbb{R}^{mn} (the vector space of vectors of length mn) the same, as their dimensions are mn , and there exists a linear, bijective mapping that transforms one into the other. [49]
- (d) If $\Phi : V \rightarrow W$ is an isomorphism, then $\Phi^{-1} : W \rightarrow V$ is an isomorphism, too. [49]

2.

Definition 15.24 (Endomorphism). $\Phi : V \rightarrow V$ linear [49]

3.

Definition 15.25 (Automorphism). $\Phi : V \rightarrow V$ linear and bijective [49]

4.

Definition 15.26 (identity mapping/ identity automorphism). $id_V : V \rightarrow V, \mathbf{x} \mapsto \mathbf{x}$ [49]

15.14. Transformation Matrix (A_Φ)

1.

Definition 15.27 (Transformation Matrix). Consider vector spaces V, W with corresponding (ordered) bases $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ and $C = (\mathbf{c}_1, \dots, \mathbf{c}_m)$. Moreover, we consider a linear mapping $\Phi : V \rightarrow W$. For $j \in \{1, \dots, n\}$, [49]

$$\Phi(\mathbf{b}_j) = \alpha_{1j}\mathbf{c}_1 + \dots + \alpha_{mj}\mathbf{c}_m = \sum_{i=1}^m \alpha_{ij}\mathbf{c}_i \quad [49]$$

is the unique representation of $\Phi(\mathbf{b}_j)$ with respect to C . Then, we call the $m \times n$ -matrix \mathbf{A}_Φ , whose elements are given by $\mathbf{A}_\Phi(i, j) = \alpha_{ij}$ the transformation matrix of Φ (with respect to the ordered bases B of V and C of W). [49]

- 2. The transformation matrix can be used to map coordinates with respect to an ordered basis in V to coordinates with respect to an ordered basis in W . [49]
- 3. Consider vector spaces V, W, X . We already know that for linear mappings $\Phi : V \rightarrow W$ and $\Psi : W \rightarrow X$ the mapping $\Psi \circ \Phi : V \rightarrow X$ is also linear. With transformation matrices \mathbf{A}_Φ and \mathbf{A}_Ψ of the corresponding mappings, the overall transformation matrix is $\mathbf{A}_{\Psi \circ \Phi} = \mathbf{A}_\Psi \mathbf{A}_\Phi$. [49]

4. \mathbf{A}_Φ is the transformation matrix of a linear mapping $\Phi_{CB} : V \rightarrow W$ with respect to the bases B, C . [49]

$\tilde{\mathbf{A}}_\Phi$ is the transformation matrix of the linear mapping $\Phi_{\tilde{C}\tilde{B}} : V \rightarrow W$ with respect to the bases \tilde{B}, \tilde{C} . [49]

\mathbf{S} is the transformation matrix of a linear mapping $\Psi_{B\tilde{B}} : V \rightarrow V$ (automorphism) that represents \tilde{B} in terms of B . Normally, $\Psi = \text{id}_V$ is the identity mapping in V . [49]

\mathbf{T} is the transformation matrix of a linear mapping $\Xi_{C\tilde{C}} : W \rightarrow W$ (automorphism) that represents \tilde{C} in terms of C . Normally, $\Xi = \text{id}_W$ is the identity mapping in W . [49]

If we (informally) write down the transformations just in terms of bases, then $\mathbf{A}_\Phi : B \rightarrow C$, $\tilde{\mathbf{A}}_\Phi : \tilde{B} \rightarrow \tilde{C}$, $\mathbf{S} : \tilde{B} \rightarrow B$, $\mathbf{T} : \tilde{C} \rightarrow C$ and $\mathbf{T}^{-1} : C \rightarrow \tilde{C}$, and [49]

$$\tilde{B} \rightarrow \tilde{C} = \tilde{\mathbf{B}} \rightarrow B \rightarrow C \rightarrow \tilde{C} \Rightarrow \tilde{\mathbf{A}}_\Phi = \mathbf{T}^{-1} \mathbf{A}_\Phi \mathbf{S} \quad [49]$$

Note that the execution order is from **right to left** because vectors are multiplied at the right-hand side so that [49]

$$x \mapsto \mathbf{S}x \mapsto \mathbf{A}_\Phi(\mathbf{S}x) \mapsto \mathbf{T}^{-1}(\mathbf{A}_\Phi(\mathbf{S}x)) = \tilde{\mathbf{A}}_\Phi x. \quad [49]$$

5. Orthogonal matrices define transformations that are rotations (with the possibility of flips). [49]

15.15. Coordinates

1. Consider a vector space V and an ordered basis $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of V . For any $\mathbf{x} \in V$ we obtain a unique representation (linear combination): [49]

$$\mathbf{x} = \alpha_1 \mathbf{b}_1 + \dots + \alpha_n \mathbf{b}_n \quad [49]$$

of \mathbf{x} with respect to B . Then $\alpha_1, \dots, \alpha_n$ are the coordinates of \mathbf{x} with respect to B , and the vector: [49]

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \quad [49]$$

is the **coordinate vector/ coordinate representation** of \mathbf{x} with respect to the ordered basis B . [49]

2. A basis effectively defines a coordinate system. [49]

3. For an n -dimensional vector space V and an ordered basis B of V , the mapping $\Phi : \mathbb{R}^n \rightarrow V$, $\Phi(\mathbf{e}_i) = \mathbf{b}_i, i = 1, \dots, n$, is linear, where $(\mathbf{e}_1, \dots, \mathbf{e}_n)$ is the standard basis of \mathbb{R}^n . [49]

4. The coordinates of $\Phi(\mathbf{b}_j)$ with respect to the ordered basis C of W are the j -th column of \mathbf{A}_Φ . [49]
[49]

(a) Consider (finite-dimensional) vector spaces V, W with ordered bases B, C and a linear mapping $\Phi : V \rightarrow W$ with transformation matrix \mathbf{A}_Φ . [49]

(b) If $\hat{\mathbf{x}}$ is the coordinate vector of $\mathbf{x} \in V$ with respect to B and $\hat{\mathbf{y}}$ the coordinate vector of $\mathbf{y} = \Phi(\mathbf{x}) \in W$ with respect to C , then $\hat{\mathbf{y}} = \mathbf{A}_\Phi \hat{\mathbf{x}}$. [49]

15.16. Projections (π)

1. Projections are an important class of linear transformations (besides rotations and reflections) and play an important role in graphics, coding theory, statistics and machine learning. [49]

2. High-dimensional data is often hard to analyze or visualize. However, high-dimensional data quite often possesses the property that only a few dimensions contain most information, and most other dimensions are not essential to describe key properties of the data. [49]

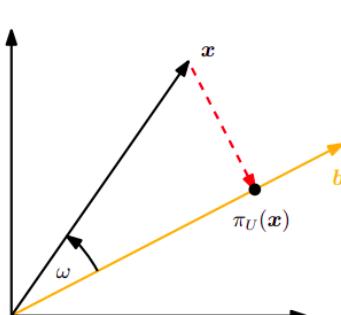
3. When we compress or visualize high-dimensional data, we will lose information. To minimize this compression loss, we ideally find the most informative dimensions in the data. [49]

4. we can project the original high-dimensional data onto a lower-dimensional **feature space** and work in this lower-dimensional space to learn more about the dataset and extract relevant patterns. [49]
5. its a technique for dimensionality reduction. [49]
6. For a given lower-dimensional subspace, orthogonal projections of high-dimensional data retain as much information as possible and minimize the difference/ error between the original data and the corresponding projection. [49]
- 7.
- Definition 15.28** (projection). Let V be a vector space and $U \subseteq V$ a subspace of V . A linear mapping $\pi : V \rightarrow U$ is called a projection if $\pi^2 = \pi \circ \pi = \pi$. [49]
8. Any projection $\pi_U(\mathbf{x})$ onto U is necessarily an element of U . Therefore, they can be represented as linear combinations of the basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_m$ of U , such that $\pi_U(\mathbf{x}) = \sum_{i=1}^m \lambda_i \mathbf{b}_i$. [49]
- 9.
- Definition 15.29** (Projection error/ reconstruction error). The corresponding projection error is the norm of the difference vector between the original vector and its projection onto U : $\|\mathbf{x} - \pi_U(\mathbf{x})\|$ [49]
10. The projections $\pi_U(\mathbf{x})$ are still vectors in \mathbb{R}^n although they lie in an m -dimensional subspace $U \subseteq \mathbb{R}^n$. However, to represent a projected vector we only need the m coordinates $\lambda_1, \dots, \lambda_m$ with respect to the basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_m$ of U . [49]
11. If the basis \mathbf{B} is an ONB, the projection equation $\pi_U(\mathbf{x}) = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{x}$ simplifies greatly to $\pi_U(\mathbf{x}) = \mathbf{B}\mathbf{B}^\top \mathbf{x}$ since $\mathbf{B}^\top \mathbf{B} = \mathbf{I}$ with coordinates $\boldsymbol{\lambda} = \mathbf{B}^\top \mathbf{x}$

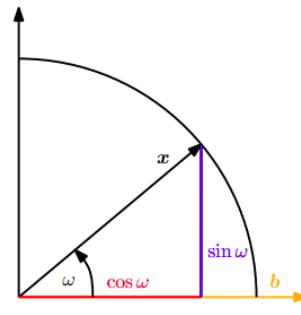
15.16.1. Projection Matrix (P_π)

1. Since linear mappings can be expressed by transformation matrices, the definition of projection applies equally to a special kind of transformation matrices, the projection matrices \mathbf{P}_π , which exhibit the property that $\mathbf{P}_\pi^2 = \mathbf{P}_\pi$. [49]
2. Projection matrices are always **symmetric**. [49]
3. The projection matrix \mathbf{P}_π projects any vector $\mathbf{x} \in \mathbb{R}^n$ onto the line through the origin with direction \mathbf{b} (equivalently, the subspace U spanned by \mathbf{b}). [49]
4. $\pi_U(\mathbf{x})$ is an eigenvector of \mathbf{P}_π , and the corresponding eigenvalue is 1. [49]

15.16.2. Projection onto One-Dimensional Subspaces (Lines)



Projection of $\mathbf{x} \in \mathbb{R}^2$ onto a subspace U with basis vector \mathbf{b} . [49]



Projection of a two-dimensional vector \mathbf{x} with $\|\mathbf{x}\| = 1$ onto a one-dimensional subspace spanned by \mathbf{b} . [49]

1. Assume we are given a line (one-dimensional subspace) through the origin with basis vector $\mathbf{b} \in \mathbb{R}^n$. The line is a one-dimensional subspace $U \subseteq \mathbb{R}^n$ spanned by \mathbf{b} . [49]
2. When we project $\mathbf{x} \in \mathbb{R}^n$ onto U , we seek the vector $\pi_U(\mathbf{x}) \in U$ that is closest to \mathbf{x} . [49]
3. properties of the projection $\pi_U(\mathbf{x})$:
 - (a) The projection $\pi_U(\mathbf{x})$ is closest to \mathbf{x} , where “closest” implies that the distance $\|\mathbf{x} - \pi_U(\mathbf{x})\|$ is minimal. It follows that the segment $\pi_U(\mathbf{x}) - \mathbf{x}$ from $\pi_U(\mathbf{x})$ to \mathbf{x} is orthogonal to U , and therefore the basis vector \mathbf{b} of U . The orthogonality condition yields $\langle \pi_U(\mathbf{x}) - \mathbf{x}, \mathbf{b} \rangle = 0$ since angles between vectors are defined via the inner product. [49]
 - (b) The projection $\pi_U(\mathbf{x})$ of \mathbf{x} onto U must be an element of U and, therefore, a multiple of the basis vector \mathbf{b} that spans U . Hence, $\pi_U(\mathbf{x}) = \lambda \mathbf{b}$, for some $\lambda \in \mathbb{R}$. [49]
4. Steps to determine λ , the projection $\pi_U(\mathbf{x}) \in U$, and the projection matrix \mathbf{P}_π that maps any $\mathbf{x} \in \mathbb{R}^n$ onto U :
 - (a) Finding the coordinate λ : [49]

$$\langle \mathbf{x} - \pi_U(\mathbf{x}), \mathbf{b} \rangle = 0$$

$$\iff \langle \mathbf{x} - \lambda \mathbf{b}, \mathbf{b} \rangle = 0 \iff \langle \mathbf{x}, \mathbf{b} \rangle - \lambda \langle \mathbf{b}, \mathbf{b} \rangle = 0$$

$$\iff \lambda = \frac{\langle \mathbf{x}, \mathbf{b} \rangle}{\langle \mathbf{b}, \mathbf{b} \rangle} = \frac{\langle \mathbf{b}, \mathbf{x} \rangle}{\|\mathbf{b}\|^2} \iff \lambda = \frac{\mathbf{b}^\top \mathbf{x}}{\mathbf{b}^\top \mathbf{b}} = \frac{\mathbf{b}^\top \mathbf{x}}{\|\mathbf{b}\|^2}$$

If $\|\mathbf{b}\| = 1$, then the coordinate λ of the projection is given by $\mathbf{b}^\top \mathbf{x}$. [49]
 - (b) Finding the projection point $\pi_U(\mathbf{x}) \in U$: [49]

$$\pi_U(\mathbf{x}) = \lambda \mathbf{b} = \frac{\langle \mathbf{x}, \mathbf{b} \rangle}{\|\mathbf{b}\|^2} \mathbf{b} = \frac{\mathbf{b}^\top \mathbf{x}}{\|\mathbf{b}\|^2} \mathbf{b}$$

$$\Rightarrow \|\pi_U(\mathbf{x})\| = \|\lambda \mathbf{b}\| = |\lambda| \|\mathbf{b}\| = \frac{|\mathbf{b}^\top \mathbf{x}|}{\|\mathbf{b}\|^2} \|\mathbf{b}\| = |\cos \omega| \|\mathbf{x}\| \|\mathbf{b}\| \frac{\|\mathbf{b}\|}{\|\mathbf{b}\|^2} = |\cos \omega| \|\mathbf{x}\|$$

ω is the angle between \mathbf{x} and \mathbf{b} [49]
 - (c) Finding the projection matrix \mathbf{P}_π : [49]

$$\pi_U(\mathbf{x}) = \lambda \mathbf{b} = \mathbf{b} \lambda = \mathbf{b} \frac{\mathbf{b}^\top \mathbf{x}}{\|\mathbf{b}\|^2} = \frac{\mathbf{b} \mathbf{b}^\top}{\|\mathbf{b}\|^2} \mathbf{x}$$

$$\pi_U(\mathbf{x}) = \mathbf{P}_\pi \mathbf{x}$$

$$\iff \frac{\mathbf{b} \mathbf{b}^\top}{\|\mathbf{b}\|^2} \mathbf{x} = \mathbf{P}_\pi \mathbf{x}$$

$$\iff \mathbf{P}_\pi = \frac{\mathbf{b} \mathbf{b}^\top}{\|\mathbf{b}\|^2}$$

Note that $\mathbf{b} \mathbf{b}^\top$ (and, consequently, \mathbf{P}_π) is a symmetric matrix (of rank 1), and $\|\mathbf{b}\|^2 = \langle \mathbf{b}, \mathbf{b} \rangle$ is a scalar.

The projection $\pi_U(\mathbf{x}) \in \mathbb{R}^n$ is still an n -dimensional vector and not a scalar. However, we no longer require n coordinates to represent the projection, but only a single one if we want to express it with respect to the basis vector \mathbf{b} that spans the subspace U : λ . [49]

```

1 import numpy as np
2
3 def project_onto_line(x, b):
4     """
5         Projects vector x onto the line spanned by vector b.
6
7     Returns:
8         lambda_scalar: the coordinate (scalar)
9             lambda such that projection = lambda * b

```

```

10     projection: the projection vector (x projected onto b)
11     projection_matrix: the projection matrix P such that P @ x = projection
12 """
13
14     b = b.reshape(-1, 1)
15     x = x.reshape(-1, 1)
16
17     # Coordinate lambda (scalar projection)
18     lambda_scalar = (b.T @ x / (b.T @ b)).item()
19
20     # Projection vector
21     projection = lambda_scalar * b # Alternatively: (b @ b.T @ x) / (b.T @ b)
22
23     # Projection matrix
24     projection_matrix = (b @ b.T) / (b.T @ b)
25
26
27     return lambda_scalar, projection, projection_matrix
28
29
30
31 # Example usage
32 x = np.array([3, 4])
33 b = np.array([1, 2])
34
35 lambda_val, projection_vec, projection_mat = project_onto_line(x, b)
36
37 print("Coordinate lambda:", lambda_val)
38 print("Projection vector:\n", projection_vec)
39 print("Projection matrix:\n", projection_mat)
40
41 # should equal projection_vec
42 print("P @ x:\n", projection_mat @ x.reshape(-1, 1))

```

Python Snippet 15.11: Projection onto One-Dimensional Subspaces (Lines) - numPy

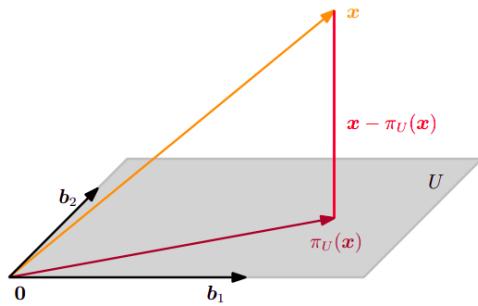
output:

```

1 Coordinate lambda: 2.2
2 Projection vector:
3 [[2.2]
4 [4.4]]
5 Projection matrix:
6 [[0.2 0.4]
7 [0.4 0.8]]
8 P @ x:
9 [[2.2]
10 [4.4]]

```

15.16.3. Projection onto General Subspaces



Projection onto a two-dimensional subspace U with basis $\mathbf{b}_1, \mathbf{b}_2$. The projection $\pi_U(\mathbf{x})$ of $\mathbf{x} \in \mathbb{R}^3$ onto U can be expressed as a linear combination of $\mathbf{b}_1, \mathbf{b}_2$ and the displacement vector $\mathbf{x} - \pi_U(\mathbf{x})$ is orthogonal to both \mathbf{b}_1 and \mathbf{b}_2 . [49]

1. orthogonal projections of vectors $\mathbf{x} \in \mathbb{R}^n$ onto lower-dimensional subspaces $U \subseteq \mathbb{R}^n$ with $\dim(U) = m \geq 1$.
2. Assume that $(\mathbf{b}_1, \dots, \mathbf{b}_m)$ is an ordered basis of U . If U is given by a set of spanning vectors, which are not a basis, make sure you determine a basis $\mathbf{b}_1, \dots, \mathbf{b}_m$ before proceeding.
3. Steps:

- (a) Find the coordinates $\lambda_1, \dots, \lambda_m$ of the projection (with respect to the basis of U), such that the linear combination: [49]

$$\pi_U(\mathbf{x}) = \sum_{i=1}^m \lambda_i \mathbf{b}_i = \mathbf{B}\boldsymbol{\lambda} \quad [49]$$

$$\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_m] \in \mathbb{R}^{n \times m}, \boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_m]^\top \in \mathbb{R}^m \quad [49]$$

is closest to $\mathbf{x} \in \mathbb{R}^n$. “Closest” means “minimum distance”, which implies that the vector connecting $\pi_U(\mathbf{x}) \in U$ and $\mathbf{x} \in \mathbb{R}^n$ must be orthogonal to all basis vectors of U . [49]

we obtain m simultaneous conditions (assuming the dot product as the inner product)

$$\langle \mathbf{b}_1, \mathbf{x} - \pi_U(\mathbf{x}) \rangle = \mathbf{b}_1^\top (\mathbf{x} - \pi_U(\mathbf{x})) = 0$$

$$\vdots \quad [49]$$

$$\langle \mathbf{b}_m, \mathbf{x} - \pi_U(\mathbf{x}) \rangle = \mathbf{b}_m^\top (\mathbf{x} - \pi_U(\mathbf{x})) = 0$$

which, with $\pi_U(\mathbf{x}) = \mathbf{B}\boldsymbol{\lambda}$, can be written as [49]

$$\mathbf{b}_1^\top (\mathbf{x} - \mathbf{B}\boldsymbol{\lambda}) = 0 \quad \dots \quad \mathbf{b}_m^\top (\mathbf{x} - \mathbf{B}\boldsymbol{\lambda}) = 0 \quad [49]$$

such that we obtain a homogeneous linear equation system [49]

$$\begin{bmatrix} \mathbf{b}_1^\top \\ \vdots \\ \mathbf{b}_m^\top \end{bmatrix} \begin{bmatrix} \mathbf{x} - \mathbf{B}\boldsymbol{\lambda} \end{bmatrix} = \mathbf{0} \iff \mathbf{B}^\top (\mathbf{x} - \mathbf{B}\boldsymbol{\lambda}) = \mathbf{0} \iff \mathbf{B}^\top \mathbf{B}\boldsymbol{\lambda} = \mathbf{B}^\top \mathbf{x} \quad [49]$$

$\mathbf{B}^\top \mathbf{B}\boldsymbol{\lambda} = \mathbf{B}^\top \mathbf{x}$ is called **normal equation**. Since $\mathbf{b}_1, \dots, \mathbf{b}_m$ are a basis of U and, therefore, linearly independent, $\mathbf{B}^\top \mathbf{B} \in \mathbb{R}^{m \times m}$ is regular and can be inverted. [49]

$$\boldsymbol{\lambda} = (\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{x} \quad [49]$$

$(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top$ is also called the pseudo-inverse of \mathbf{B} [49]

In practical applications (e.g., linear regression), we often add a “jitter term” εI to $\mathbf{B}^\top \mathbf{B}$ to guarantee increased numerical stability and positive definiteness. This “ridge” can be rigorously derived using Bayesian inference. [49]

A ridge term is essentially the same as a jitter term but used in the context of regularization, especially in ridge regression (also called Tikhonov regularization). [7]

- (b) Find the projection $\pi_U(\mathbf{x}) \in U$: [49]

$$\pi_U(\mathbf{x}) = \mathbf{B}\boldsymbol{\lambda} = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{x} \quad [49]$$

- (c) Find the projection matrix \mathbf{P}_π : [49]

$$\pi_U(\mathbf{x}) = \mathbf{P}_\pi \mathbf{x} = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{x} \iff \mathbf{P}_\pi = \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top = \frac{\mathbf{B}\mathbf{B}^\top}{\mathbf{B}^\top \mathbf{B}} \quad [49]$$

```

1 import numpy as np
2
3 def project_onto_subspace(x, B):
4     """
5         Projects vector x onto the subspace spanned by the column vectors of matrix B.
6
7     Args:
8         x: A numpy array of shape (n,) representing the vector to project
9         B: A numpy array of shape (n, m) whose
10            columns are basis vectors of the subspace
11
12    Returns:
13        lambda_vec: Coordinate vector lambda such that projection = B @ lambda_vec
14        projection: The projection vector of x onto the subspace spanned by B
15        projection_matrix: The projection matrix P such that P @ x = projection
16    """
17
18    # Reshape x as column vector
19    x = x.reshape(-1, 1)
20
21    # Compute (B^T B)^(-1) B^T x
22    BtB_inv = np.linalg.inv(B.T @ B)
23    lambda_vec = BtB_inv @ B.T @ x
24
25    # Compute projection: B @ lambda
26    projection = B @ lambda_vec
27
28    # Projection matrix: B (B^T B)^(-1) B^T
29    projection_matrix = B @ BtB_inv @ B.T
30
31    return lambda_vec, projection, projection_matrix
32
33 # Example usage
34 x = np.array([2, 3, 4])
35
36 # Define individual basis vectors (as 1D arrays or column vectors)
37 b1 = np.array([1, 0, 1])
38 b2 = np.array([0, 1, 1])
39
40 # Stack them as columns to form matrix B
41 B = np.column_stack((b1, b2)) # Equivalent to np.stack((b1, b2), axis=1)
42
43 lambda_vec, projection_vec, projection_mat = project_onto_subspace(x, B)
44
45 print("Coordinate vector lambda:\n", lambda_vec)
46 print("Projection vector:\n", projection_vec)
47 print("Projection matrix:\n", projection_mat)
48
49 # should equal projection_vec
50 print("P @ x:\n", projection_mat @ x.reshape(-1, 1))

```

Python Snippet 15.12: Projection onto General Subspaces - numPy

Output:

```

1 Coordinate vector lambda:
2 [[1.66666667]

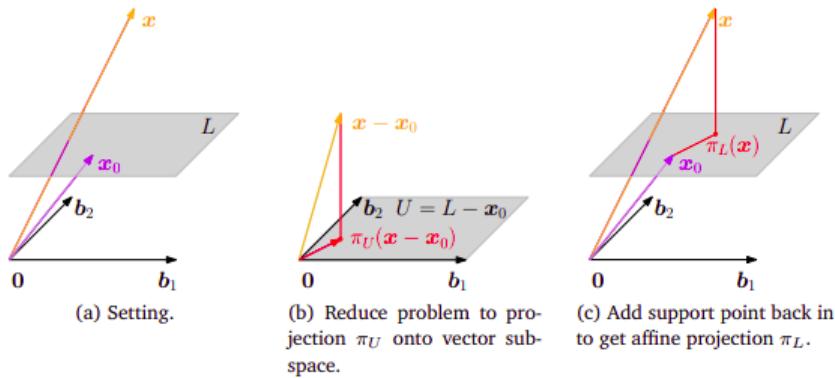
```

```

3 [2.66666667]]
4 Projection vector:
5 [[1.66666667]
6 [2.66666667]
7 [4.33333333]]
8 Projection matrix:
9 [[ 0.66666667 -0.33333333  0.33333333]
10 [-0.33333333  0.66666667  0.33333333]
11 [ 0.33333333  0.33333333  0.66666667]]
12 P @ x:
13 [[1.66666667]
14 [2.66666667]
15 [4.33333333]]

```

15.16.4. Projection onto Affine Subspaces



Projection onto an affine space.

- (a) original setting; [49]
- (b) setting shifted by $-x_0$ so that $x - x_0$ can be projected onto the direction space U ; [49]
- (c) projection is translated back to $x_0 + \pi_U(x - x_0)$, which gives the final orthogonal projection $\pi_L(x)$. [49]

1. We are given an affine space $L = x_0 + U$, where b_1, b_2 are basis vectors of U . [49]
2. To determine the orthogonal projection $\pi_L(x)$ of x onto L , we transform the problem into a problem that we know how to solve: the projection onto a vector subspace. [49]
3. In order to get there, we subtract the support point x_0 from x and from L , so that $L - x_0 = U$ is exactly the vector subspace U . [49]
4. We can now use the orthogonal projections onto a subspace and obtain the projection $\pi_U(x - x_0)$. [49]
5. This projection can now be translated back into L by adding x_0 , such that we obtain the orthogonal projection onto an affine space L as: [49]

$$\pi_L(x) = x_0 + \pi_U(x - x_0)$$
 [49]
 where $\pi_U(\cdot)$ is the orthogonal projection onto the subspace U , i.e., the direction space of L . [49]
6. it is also evident that the distance of x from the affine space L is identical to the distance of $x - x_0$ from U , i.e., [49]

$$d(x, L) = \|x - \pi_L(x)\| = \|x - (x_0 + \pi_U(x - x_0))\| = d(x - x_0, \pi_U(x - x_0)) = d(x - x_0, U)$$
 [49]

15.17. Rotations (Φ) & Rotation Matrix ($R(\theta)$)

1. A rotation is a linear mapping (more specifically, an automorphism of rotation a Euclidean vector space) that rotates a plane by an angle θ about the origin, i.e., the origin is a fixed point. [49]
2. For a positive angle $\theta > 0$, by common convention, we rotate in a counterclockwise direction. [49]
3. The rotated vectors are still linearly independent and, therefore, are a basis of \mathbb{R}^2 . This means that the rotation performs a basis change. [49]
4. Rotations Φ are linear mappings so that we can express them by a rotation matrix $R(\theta)$. [49]

5. Properties:

- (a) Rotations preserve distances, i.e., $\|\mathbf{x} - \mathbf{y}\| = \|R_\theta(\mathbf{x}) - R_\theta(\mathbf{y})\|$. In other words, rotations leave the distance between any two points unchanged after the transformation. [49]
- (b) Rotations preserve angles, i.e., the angle between $R_\theta \mathbf{x}$ and $R_\theta \mathbf{y}$ equals the angle between \mathbf{x} and \mathbf{y} . [49]
- (c) Rotations in three (or more) dimensions are generally not commutative. Therefore, the order in which rotations are applied is important, even if they rotate about the same point. [49]
- (d) Only in two dimensions vector rotations are commutative, such that $R(\phi)R(\theta) = R(\theta)R(\phi) \quad \forall \phi, \theta \in [0, 2\pi)$. They form an Abelian group (with multiplication) only if they rotate about the same point (e.g., the origin). [49]
- (e) Its columns form an orthonormal basis of \mathbb{R}^n . [7]
- (f) It has determinant +1 (to distinguish from reflections, which have determinant -1). [7]

15.17.1. Rotations in \mathbb{R}^2

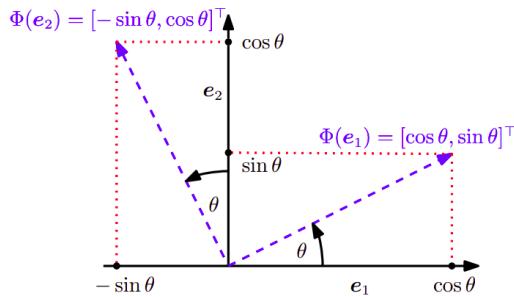


Fig. 15.1: Rotation of the standard basis in \mathbb{R}^2 by an angle θ . [49]

1. Consider the standard basis $\left\{ \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$ of \mathbb{R}^2 , which defines the standard coordinate system in \mathbb{R}^2 . [49]
 2. using trigonometry: $\Phi(\mathbf{e}_1) = \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$, $\Phi(\mathbf{e}_2) = \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix}$ [49]
 3. the rotation matrix that performs the basis change into the rotated coordinates $R(\theta)$ is given as: [49]
- $$R(\theta) = [\Phi(\mathbf{e}_1) \quad \Phi(\mathbf{e}_2)] = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad [49]$$

15.17.2. Rotations in \mathbb{R}^3

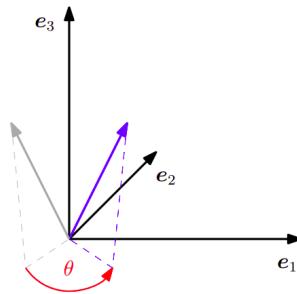


Fig. 15.2: Rotation of a vector (gray) in \mathbb{R}^3 by an angle θ about the e_3 -axis. The rotated vector is shown in blue. [49]

1. In contrast to the \mathbb{R}^2 case, in \mathbb{R}^3 we can rotate any two-dimensional plane about a one-dimensional axis. [49]
2. The easiest way to specify the general rotation matrix is to specify how the images of the standard basis e_1, e_2, e_3 are supposed to be rotated, and making sure these images Re_1, Re_2, Re_3 are orthonormal to each other. We can then obtain a general rotation matrix R by combining the images of the standard basis. [49]
3. We use the convention that a “counterclockwise” (planar) rotation about an axis refers to a rotation about an axis when we look at the axis “head on, from the end toward the origin”. [49]
4. Rotation about the e_1 -axis: [49]

$$R_1(\theta) = [\Phi(e_1) \quad \Phi(e_2) \quad \Phi(e_3)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \quad [49]$$

Here, the e_1 coordinate is fixed, and the counterclockwise rotation is performed in the e_2e_3 plane. [49]

5. Rotation about the e_2 -axis: [49]

$$R_2(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \quad [49]$$

If we rotate the e_1e_3 plane about the e_2 axis, we need to look at the e_2 axis from its “tip” toward the origin. [49]

6. Rotation about the e_3 -axis: [49]

$$R_3(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [49]$$

15.17.3. Rotations in n Dimensions (Givens Rotation)

1. The generalization of rotations from 2D and 3D to n -dimensional Euclidean vector spaces can be intuitively described as fixing $n - 2$ dimensions and restrict the rotation to a two-dimensional plane in the n -dimensional space.

2.

Definition 15.30 (Givens Rotation). Let V be an n -dimensional Euclidean vector space and $\Phi : V \rightarrow V$ an automorphism with transformation matrix:

$$R_{ij}(\theta) := \begin{bmatrix} I_{i-1} & 0 & \dots & \dots & 0 \\ 0 & \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 0 & I_{j-i-1} & 0 & 0 \\ 0 & \sin \theta & 0 & \cos \theta & 0 \\ 0 & \dots & \dots & 0 & I_{n-j} \end{bmatrix} \in \mathbb{R}^{n \times n} \quad [49]$$

for $1 \leq i < j \leq n$ and $\theta \in \mathbb{R}$. Then $R_{ij}(\theta)$ is called a Givens rotation. Essentially, $R_{ij}(\theta)$ is the identity

matrix I_n with

[49]

$$\cdot \quad r_{ii} = \cos \theta, \quad r_{ij} = -\sin \theta, \quad r_{ji} = \sin \theta, \quad r_{jj} = \cos \theta$$

[49]

3. **Note:** 2D is special case

CHAPTER 16

VECTOR CALCULUS

16.1. Function ($f(x)$)

1. A function f is a quantity that relates two quantities to each other. [49]
2. These quantities are typically inputs $\mathbf{x} \in \mathbb{R}^D$ and targets (function values) $f(\mathbf{x})$, which we assume are real-valued if not stated otherwise. [49]
3. \mathbb{R}^D is the domain of f , and the function values $f(\mathbf{x})$ are the image/codomain of f . [49]
4. We often write [49]
 - (a) $f : \mathbb{R}^D \rightarrow \mathbb{R}$ (specifies that f is a mapping from \mathbb{R}^D to \mathbb{R}) [49]
 - (b) $\mathbf{x} \mapsto f(\mathbf{x})$ (specifies the explicit assignment of an input \mathbf{x} to a function value $f(\mathbf{x})$) [49]to specify a function. [49]
5. A function f assigns every input \mathbf{x} exactly one function value $f(\mathbf{x})$. [49]

ALSO SEE/ RELATED:

1. (15.10) Linear Mappings (Φ)

16.1.1. Differentiation of Uni-variate Functions

1.
Definition 16.1 (Univariate function ($f : \mathbb{R} \rightarrow \mathbb{R}$)). Univariate function: $y = f(x)$, $x, y \in \mathbb{R}$ [49]
2.
Definition 16.2 (Difference Quotient). The difference quotient $\frac{\delta y}{\delta x} := \frac{f(x + \delta x) - f(x)}{\delta x}$ computes the slope of the secant line through two points on the graph of f . [49]
3. The derivative of f points in the direction of steepest ascent of f . [49]
4.
Definition 16.3 (Power Series). $f(x) = \sum_{k=0}^{\infty} a_k (x - c)^k$
where a_k are coefficients and c is a constant

16.1.2. Differentiation of multi-variate Functions

1.
Definition 16.4 (multi-variate Functions ($f : \mathbb{R}^n \rightarrow \mathbb{R}$)). The general case where the function f depends on one or more variables $\mathbf{x} \in \mathbb{R}^n$ [49]
2.
Definition 16.5 (Partial Derivative ($\frac{\partial f}{\partial x_i}$)). For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mathbf{x} \mapsto f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$ of n variables x_1, \dots, x_n we define the partial derivatives as [49]

$$\begin{aligned}\frac{\partial f}{\partial x_1} &= \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(\mathbf{x})}{h} \\ &\vdots \\ \frac{\partial f}{\partial x_n} &= \lim_{h \rightarrow 0} \frac{f(x_1, x_2, \dots, x_n + h) - f(\mathbf{x})}{h}\end{aligned}\quad [49]$$

16.1.2.1. Gradient

1. The generalization of the derivative to functions of several variables is the **gradient**. [49]
2. We find the gradient of the function f with respect to \mathbf{x} by varying one variable at a time and keeping the others constant. [49]
3. The gradient is then the **collection** of these partial derivatives. [49]
- 4.

Definition 16.6 (Gradient ($\nabla_{\mathbf{x}} f \in \mathbb{R}^{1 \times n}$)). For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mathbf{x} \mapsto f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$ of n variables x_1, \dots, x_n :

$$\nabla_{\mathbf{x}} f = \text{grad } f = \frac{df}{d\mathbf{x}} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \quad \frac{\partial f(\mathbf{x})}{\partial x_2} \quad \dots \frac{\partial f(\mathbf{x})}{\partial x_n} \right] \in \mathbb{R}^{1 \times n} \quad [49]$$

where n is the number of variables and 1 is the dimension of the image/range/codomain of f . The row vector is called the gradient of f or the Jacobian. [49]

5. This definition of Jacobian is a special case of the general definition of the Jacobian for vector-valued functions as the collection of partial derivatives. [49]
6. The reason why we define the gradient vector as a row vector: [49]
 - (a) we can consistently generalize the gradient to vector-valued functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (then the gradient becomes a matrix) [49]
 - (b) we can immediately apply the multi-variate chain rule without paying attention to the dimension of the gradient. [49]

16.2. Gradients of Vector-Valued Functions ($J = \nabla_{\mathbf{x}} f$)

1.

Definition 16.7 (vector-valued functions/ vector fields ($\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$)). vector-valued functions (vector fields) are defined as $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where $n \geq 1$ and $m > 1$. For a function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and a vector

$\mathbf{x} = [x_1, \dots, x_n]^T \in \mathbb{R}^n$, the corresponding vector of function values is given as $\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^m$ [49]

2. Writing the vector-valued function in this way allows us to view a vector-valued function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ as a vector of functions $[f_1, \dots, f_m]^T$, $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ that map onto \mathbb{R} . [49]

$$3. \quad \frac{\partial \mathbf{f}}{\partial x_i} = \begin{bmatrix} \frac{\partial f_1}{\partial x_i} \\ \vdots \\ \frac{\partial f_m}{\partial x_i} \end{bmatrix} = \begin{bmatrix} \lim_{h \rightarrow 0} \frac{f_1(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_n) - f_1(\mathbf{x})}{h} \\ \vdots \\ \lim_{h \rightarrow 0} \frac{f_m(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_n) - f_m(\mathbf{x})}{h} \end{bmatrix} \in \mathbb{R}^m \quad [49]$$

4.

Definition 16.8 (Jacobian ($J = \nabla_{\mathbf{x}} \mathbf{f} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}}$)). The collection of all first-order partial derivatives of a vector-valued function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called the Jacobian. The Jacobian J is an $m \times n$ matrix, which we define and arrange as follows: [49]

$$\mathbf{J} = \nabla_{\mathbf{x}} \mathbf{f} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad [49]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad J(i, j) = \frac{\partial f_i}{\partial x_j} \quad [49]$$

- (a) As a special case, a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^1$, which maps a vector $\mathbf{x} \in \mathbb{R}^n$ onto a scalar (e.g., $f(\mathbf{x}) = \sum_{i=1}^n x_i$), possesses a Jacobian that is a row vector (matrix of dimension $1 \times n$) [49]
- (b) Jacobian is used in the change-of-variable method for probability distributions. The amount of scaling due to the transformation of a variable is provided by the determinant. [49]
- (c) The absolute value of the Jacobian determinant $|det(\mathbf{J})|$ is the factor by which areas or volumes are scaled when coordinates are transformed. [49]
- (d) Geometrically, the Jacobian determinant gives the magnification/ scaling factor when we transform an area or volume. [49]

16.3. Higher-Order Derivatives

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x} \left(\frac{\partial f}{\partial x} \right) \quad \mid \quad \frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y} \left(\frac{\partial f}{\partial x} \right) \quad \mid \quad \frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x} \left(\frac{\partial f}{\partial y} \right) \quad [49]$$

$$\frac{\partial^n f}{\partial x^n} = \frac{\partial}{\partial x} \left(\underset{n \text{ times}}{\dots} \left(\frac{\partial}{\partial x} \left(\frac{\partial f}{\partial x} \right) \right) \dots \right) \quad [49]$$

1.

Definition 16.9 (Hessian ($\mathbf{H} = \nabla_{x,y}^2 f(x,y)$)). The Hessian is the collection of all second-order partial derivatives. [49]

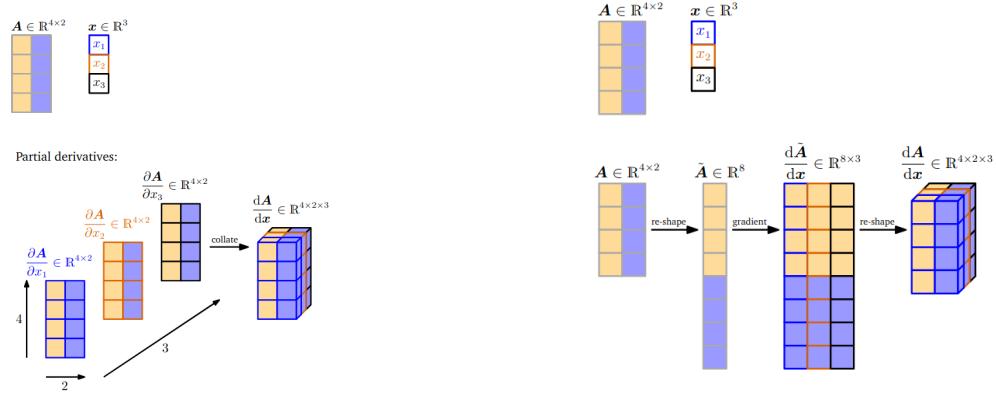
2. If $f(x,y)$ is a twice (continuously) differentiable function, then $\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial^2 f}{\partial x \partial y}$ [49]

$$3. \mathbf{H} = \nabla_{x,y}^2 f(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \quad [49]$$

4. Generally, for $\mathbf{x} \in \mathbb{R}^n$ and $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the Hessian is an $n \times n$ matrix. The Hessian measures the curvature of the function locally around (x,y) . [49]

5. (Hessian of a Vector Field) If $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a vector field, the Hessian is an $(m \times n \times n)$ -tensor. [49]

16.4. Gradients of Matrices



1.

Definition 16.10 (Tensor). Tensor is a multidimensional array [49]

2.

Definition 16.11 (Flattening (matrix \rightarrow vector)). Matrices can be transformed into vectors by stacking the columns of the matrix [49]

3. if we compute the gradient of an $m \times n$ matrix \mathbf{A} with respect to a $p \times q$ matrix \mathbf{B} , the resulting Jacobian would be $(m \times n) \times (p \times q)$, i.e., a four-dimensional tensor \mathbf{J} , whose entries are given as $J_{ijkl} = \frac{\partial A_{ij}}{\partial B_{kl}}$. [49]

16.5. Differentiation Rules

1. Product rule:

$$(a) (f(x)g(x))' = \frac{d}{dx}(f(x)g(x)) = g(x)\frac{d}{dx}f(x) + f(x)\frac{d}{dx}g(x) = f'(x)g(x) + f(x)g'(x) \quad [49]$$

$$(b) \frac{\partial}{\partial \mathbf{x}}(f(\mathbf{x})g(\mathbf{x})) = g(\mathbf{x})\frac{\partial}{\partial \mathbf{x}}f(\mathbf{x}) + f(\mathbf{x})\frac{\partial}{\partial \mathbf{x}}g(\mathbf{x}) \quad [49]$$

$$2. \text{ Quotient rule: } \left(\frac{f(x)}{g(x)}\right)' = \frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2} \quad [49]$$

3. Sum Rule:

$$(a) (f(x) + g(x))' = f'(x) + g'(x) \quad [49]$$

$$(b) \frac{\partial}{\partial \mathbf{x}}(f(\mathbf{x}) + g(\mathbf{x})) = \frac{\partial}{\partial \mathbf{x}}f(\mathbf{x}) + \frac{\partial}{\partial \mathbf{x}}g(\mathbf{x}) \quad [49]$$

4. Chain Rule:

$$(a) (g(f(x)))' = (g \circ f)'(x) = g'(f(x))f'(x) \quad [49]$$

$$(b) \frac{\partial}{\partial \mathbf{x}}(g(f(\mathbf{x}))) = \frac{\partial}{\partial \mathbf{x}}(g \circ f)(\mathbf{x}) = \frac{\partial g(\mathbf{x})}{\partial f(\mathbf{x})} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \quad [49]$$

$g \circ f$ denotes function composition $x \mapsto f(x) \mapsto g(f(x))$. [49]

Gradients of Matrices & Vectors:

5. $\frac{\partial \mathbf{f}(\mathbf{A})^\top}{\partial \mathbf{A}} = \left(\frac{\partial \mathbf{f}(\mathbf{A})}{\partial \mathbf{A}} \right)^\top$ [49]	9. $\frac{\partial \mathbf{x}^\top \mathbf{A} \mathbf{y}}{\partial \mathbf{A}} = \mathbf{x} \mathbf{y}^\top$ [49]
6. $\frac{\partial \text{tr}(\mathbf{f}(\mathbf{A}))}{\partial \mathbf{A}} = \text{tr} \left(\frac{\partial \mathbf{f}(\mathbf{A})}{\partial \mathbf{A}} \right)$ [49]	10. $\frac{\partial \mathbf{x}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} = \mathbf{x}^\top (\mathbf{A} + \mathbf{A}^\top)$ [49]
7. $\frac{\partial \mathbf{f}(\mathbf{A})^{-1}}{\partial \mathbf{A}} = -\mathbf{f}(\mathbf{A})^{-1} \frac{\partial \mathbf{f}(\mathbf{A})}{\partial \mathbf{A}} \mathbf{f}(\mathbf{A})^{-1}$ [49]	11. $\frac{\partial \mathbf{x}^\top \mathbf{y}}{\partial \mathbf{x}} = \mathbf{y}^\top$ [49]
8. $\frac{\partial \mathbf{x}^\top \mathbf{A}^{-1} \mathbf{y}}{\partial \mathbf{A}} = -(\mathbf{A}^{-1})^\top \mathbf{x} \mathbf{y}^\top (\mathbf{A}^{-1})^\top$ [49]	12. $\frac{\partial \mathbf{y}^\top \mathbf{x}}{\partial \mathbf{x}} = \mathbf{y}^\top$ [49]
13. $\frac{\partial \det(\mathbf{f}(\mathbf{A}))}{\partial \mathbf{A}} = \det(\mathbf{f}(\mathbf{A})) \text{tr} \left(\mathbf{f}(\mathbf{A})^{-1} \frac{\partial \mathbf{f}(\mathbf{A})}{\partial \mathbf{A}} \right)$ [49]	
14. $\frac{\partial (\mathbf{x} - \mathbf{As})^\top \mathbf{W} (\mathbf{x} - \mathbf{As})}{\partial \mathbf{s}} = -2(\mathbf{x} - \mathbf{As})^\top \mathbf{W} \mathbf{A}$ (for symmetric \mathbf{W}) [49]	

Examples

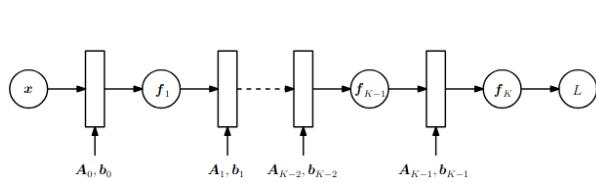
1. Consider a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ of two variables x_1, x_2 . Furthermore, $x_1(t)$ and $x_2(t)$ are themselves functions of t .

$$\nabla_t f(x_1, x_2) = \frac{df}{dt} = \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1(t)} & \frac{\partial f(x_1, x_2)}{\partial x_2(t)} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x_2(t)}{\partial t} \end{bmatrix} = \frac{\partial f(x_1, x_2)}{\partial x_1(t)} \frac{\partial x_1(t)}{\partial t} + \frac{\partial f(x_1, x_2)}{\partial x_2(t)} \frac{\partial x_2(t)}{\partial t} \quad [49]$$

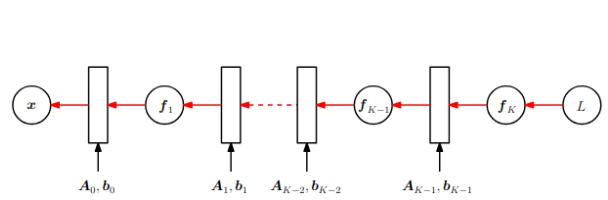
2. If $f(x_1, x_2)$ is a function of x_1 and x_2 , where $x_1(s, t)$ and $x_2(s, t)$ are themselves functions of two variables s and t :

$$\begin{aligned} \nabla_{(s,t)} f(x_1, x_2) &= \frac{df(x_1, x_2)}{d(s, t)} = \frac{\partial f}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial (s, t)} = \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1(s, t)} & \frac{\partial f(x_1, x_2)}{\partial x_2(s, t)} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(s, t)}{\partial s} & \frac{\partial x_1(s, t)}{\partial t} \\ \frac{\partial x_2(s, t)}{\partial s} & \frac{\partial x_2(s, t)}{\partial t} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1(s, t)} \frac{\partial x_1(s, t)}{\partial s} + \frac{\partial f(x_1, x_2)}{\partial x_2(s, t)} \frac{\partial x_2(s, t)}{\partial s} \\ \frac{\partial f(x_1, x_2)}{\partial x_1(s, t)} \frac{\partial x_1(s, t)}{\partial t} + \frac{\partial f(x_1, x_2)}{\partial x_2(s, t)} \frac{\partial x_2(s, t)}{\partial t} \end{bmatrix} \end{aligned} \quad [49]$$

16.6. Backpropagation: Gradients in DNN



Forward pass in a multi-layer neural network to compute the loss L as a function of the inputs \mathbf{x} and the parameters $\mathbf{A}_i, \mathbf{b}_i$. [49]



Backward pass in a multi-layer neural network to compute the gradients of the loss function. [49]

1. An area where the chain rule is used to an extreme is deep learning, where the function value \mathbf{y} is computed as a many-level function composition [49]

$$\mathbf{y} = (f_K \circ f_{K-1} \circ \dots \circ f_1)(\mathbf{x}) = f_K(f_{K-1}(\dots(f_1(\mathbf{x})\dots))) \quad [49]$$

where \mathbf{x} are the inputs (e.g., images), \mathbf{y} are the observations (e.g., class labels), and every function $f_i, i = 1, \dots, K$, possesses its own parameters. [49]

(a) $\mathbf{f}_0 := \mathbf{x}$	[49]	(c) $\mathbf{f}_i(\mathbf{x}_{i-1}) = \sigma_i(\mathbf{A}_{i-1}\mathbf{x}_{i-1} + \mathbf{b}_{i-1})$	[49]
(b) $\mathbf{f}_i := \sigma_i(\mathbf{A}_{i-1}\mathbf{f}_{i-1} + \mathbf{b}_{i-1})$	[49]	(d) $\mathbf{y} = \mathbf{f}_K(\mathbf{x}_{K-1})$	[49]
(e) $L(\boldsymbol{\theta}) = \ \mathbf{y} - \mathbf{f}_K(\boldsymbol{\theta}, \mathbf{x})\ ^2$	[49]	(g) $\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}}$	[49]
(f) $\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{K-1}}$	[49]	(h) $\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{K-1}} \cdots \frac{\partial \mathbf{f}_{i+2}}{\partial \boldsymbol{\theta}_{i+1}} \frac{\partial \mathbf{f}_{i+1}}{\partial \boldsymbol{\theta}_i}$	[49]

orange partial derivatives of the output of a layer with respect to its inputs

blue partial derivatives of the output of a layer with respect to its parameters

where:

(a) i : layer ($i = 1, \dots, K$)	[49]	(d) L : loss function	[49]
(b) \mathbf{x}_{i-1} : output of layer $i-1$	[49]	(e) $\mathbf{A}_j, \mathbf{b}_j$: parameters ($j = 0, \dots, K-1$)	[49]
(c) σ_i : activation function of layer i	[49]	(f) $\boldsymbol{\theta} = \{\mathbf{A}_0, \mathbf{b}_0, \dots, \mathbf{A}_{K-1}, \mathbf{b}_{K-1}\}$	[49]

16.7. Computational Graph

1.

Definition 16.12 (Computational Graph). Computational graphs are a type of graph that can be used to represent mathematical expressions. This is similar to descriptive language in the case of deep learning models, providing a functional description of the required computation. [50]

2. In general, the computational graph is a **directed graph** that is used for expressing and evaluating mathematical expressions. [50]
3. These can be used for two different types of calculations: [50]
 - (a) Forward computation [50]
 - (b) Backward computation [50]
4. Structure of a computational graph:
 - (a) A variable is represented by a **node** in a graph. It could be a scalar, vector, matrix, tensor, or even another type of variable. [50]
 - (b) A function argument and data dependency are both represented by an **edge**. These are similar to node pointers. [50]
 - (c) A simple function of one or more variables is called an operation. There is a set of operations that are permitted. Functions that are more complex than these operations in this set can be represented by combining multiple operations. [50]
5. Types of computational graphs:
 - (a) Static Computational Graphs [50]
 - (b) Dynamic Computational Graphs [50]

16.7.1. Static Computational Graphs

1. Involves two phases
 - (a) Phase 1:- Make a plan for your architecture [50]
 - (b) Phase 2:- To train the model and generate predictions, feed it a lot of data. [50]

2. The benefit of utilizing this graph is that it enables powerful offline graph optimization and scheduling. As a result, they should be faster than dynamic graphs in general. [50]
3. The drawback is that dealing with structured and even variable-sized data is unsightly. [50]

16.7.2. Dynamic Computational Graphs

1. As the forward computation is performed, the graph is implicitly defined.
2. This graph has the advantage of being more adaptable. The library is less intrusive and enables interleaved graph generation and evaluation. The forward computation is implemented in your preferred programming language, complete with all of its features and algorithms. Debugging dynamic graphs is simple because it permits line-by-line execution of the code and access to all variables, finding bugs in your code is considerably easier.
3. The disadvantage of employing this graph is that there is limited time for graph optimization, and the effort may be wasted if the graph does not change.

16.8. Automatic Differentiation

Example: $x \mapsto a \mapsto b \mapsto y$ $\frac{dy}{dx} = \frac{dy}{db} \frac{db}{da} \frac{da}{dx}$ [49]

- 1.
- Definition 16.13** (Automatic differentiation). Automatic differentiation is different from symbolic differentiation and numerical approximations of the gradient, e.g., by using finite differences. We can think of automatic differentiation as a set of techniques to numerically (in contrast to symbolically) evaluate the exact (up to machine precision) gradient of a function by working with intermediate variables and applying the chain rule. [49]
2. Automatic differentiation applies a series of elementary arithmetic operations, e.g., addition and multiplication and elementary functions, e.g., sin, cos, exp, log. [49]
3. By applying the chain rule to these operations, the gradient of quite complicated functions can be computed automatically. [49]
4. Automatic differentiation applies to general computer programs and has forward and reverse modes. Intuitively, the forward and reverse mode differ in the order of multiplication.
 $\frac{dy}{dx} = \left(\frac{dy}{db} \frac{db}{da} \right) \frac{da}{dx}$ would be the reverse mode because gradients are propagated backward through the graph, i.e., reverse to the data flow. $\frac{dy}{dx} = \frac{dy}{db} \left(\frac{db}{da} \frac{da}{dx} \right)$ would be the forward mode, where the gradients flow with the data from left to right through the graph.
5. In the context of neural networks, where the input dimensionality is often much higher than the dimensionality of the labels, the reverse mode is computationally significantly cheaper than the forward mode. [49]
6. Let x_1, \dots, x_d be the input variables to the function, x_{d+1}, \dots, x_{D-1} be the intermediate variables, and x_D the output variable. Then the computation graph can be expressed as follows: [49]

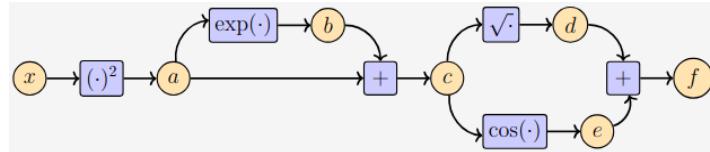
Forward propagation: $x_i = g_i(x_{Pa(x_i)}) \quad i = d + 1, \dots, D \quad f = x_D$ [49]

where $g_i(\cdot)$ are elementary functions $Pa(x_i)$ are indices (ex: $Pa(x_0) = \{1, 2\}$) of parent nodes of the variable x_i in the graph and $x_{Pa(x_i)}$ are the parent nodes (ex: $x_{Pa(x_0)} = \{x_1, x_2\}$) of the variable x_i in the graph. [49]

Backward propagation: $\frac{\partial f}{\partial x_D} = 1 \quad \frac{\partial f}{\partial x_i} = \sum_{j; i \in Pa(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{j; i \in Pa(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial g_j}{\partial x_i}$ [49]

7. The automatic differentiation approach above works whenever we have a function that can be expressed as a computation graph, where the elementary functions are differentiable. In fact, the function may not even be a mathematical function but a computer program. However, not all computer programs can be automatically differentiated, e.g., if we cannot find differential elementary functions. Programming structures, such as `for` loops and `if` statements, require more care as well. [49]

8. Example: $f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$



Computation graph with inputs x , function values f , and intermediate variables a, b, c, d, e . [49]

$$\begin{array}{lll|lll|lll}
 a = x^2 & b = \exp(a) & c = a + b & d = \sqrt{c} & e = \cos(c) & f = d + e \\
 \frac{\partial a}{\partial x} = 2x & \frac{\partial b}{\partial a} = \exp(a) & \frac{\partial c}{\partial a} = \frac{\partial c}{\partial b} = 1 & \frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}} & \frac{\partial e}{\partial c} = -\sin(c) & \frac{\partial f}{\partial d} = \frac{\partial f}{\partial e} = 1 \\
 \frac{\partial f}{\partial c} = \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} + \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} & \frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} + \frac{\partial f}{\partial c} \frac{\partial c}{\partial a} & \frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} & \frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x} & &
 \end{array}$$

16.9. Taylor Series & Polynomial

1.

Definition 16.14 (Taylor Series ($T_\infty(x)$)). For a smooth function $f \in C^\infty$, $f : \mathbb{R} \rightarrow \mathbb{R}$, the Taylor series of f at x_0 is defined as $f(x) = T_\infty(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$ [49]
 $f \in C^\infty$ means that f is continuously differentiable infinitely many times [49]

(a)

Definition 16.15 (Maclaurin series). For $x_0 = 0$, we obtain the **Maclaurin series** as a special instance of the Taylor series. [49]

(b)

Definition 16.16 (Analytic). If $f(x) = T_\infty(x)$, then f is called **analytic**. [49]

2.

Definition 16.17 (Multivariate Taylor Series). We consider a function $f : \mathbb{R}^D \rightarrow \mathbb{R}, \mathbf{x} \mapsto f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^D$ that is smooth at \mathbf{x}_0 . When we define the difference vector $\boldsymbol{\delta} := \mathbf{x} - \mathbf{x}_0$, the multivariate Taylor series of f at (\mathbf{x}_0) is defined as [49]

$$f(\mathbf{x}) = \sum_{k=0}^{\infty} \frac{D_{\mathbf{x}}^k f(\mathbf{x}_0)}{k!} \boldsymbol{\delta}^k \quad [49]$$

where $D_{\mathbf{x}}^k f(\mathbf{x}_0)$ is the k -th (total) derivative of f with respect to \mathbf{x} , evaluated at \mathbf{x}_0 .

SEE: Outer product

[49]

3.

Definition 16.18 (Taylor Polynomial ($T_n(x)$)). The Taylor polynomial of degree n of $f : \mathbb{R} \rightarrow \mathbb{R}$ at x_0 is defined as $T_n(x) := \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$ [49]

where $f^{(k)}(x_0)$ is the k -th derivative of f at x_0 (which we assume exists) and $\frac{f^{(k)}(x_0)}{k!}$ are the coefficients of the polynomial. [49]

We define $t^0 := 1$ for all $t \in \mathbb{R}$. [49]

- (a) In general, a Taylor polynomial of degree n is an **approximation** of a function, which does not need to be a polynomial. The Taylor polynomial is similar to f in a neighborhood around x_0 . Higher-order Taylor polynomials approximate the function f better and more globally. [49]
- (b) A Taylor polynomial of degree n is an **exact** representation of a polynomial f of degree $k \leq n$ since all derivatives $f(i)$, $i > k$ vanish. [49]

4.

Definition 16.19 (Multivariate Taylor Polynomial ($T_n(\mathbf{x})$)). The Taylor polynomial of degree n of $f : \mathbb{R}^D \rightarrow \mathbb{R}$ at \mathbf{x}_0 is defined as $T_n(x) := \sum_{k=0}^n \frac{D_{\mathbf{x}}^k f(\mathbf{x}_0)}{k!} \boldsymbol{\delta}^k$ [49]

where $D_{\mathbf{x}}^k f(\mathbf{x}_0)$ is the k -th (total) derivative of f with respect to \mathbf{x} , evaluated at \mathbf{x}_0 .

SEE: Outer product

[49]

CHAPTER 17

SYSTEMS OF LINEAR EQUATIONS

coefficients	a_{ij}	$\in \mathbb{R}$
constants	b_i	$\in \mathbb{R}$
unknowns	x_i	$\in \mathbb{R}$

Notations

1. **general form** of a system of linear equations:

$$a_{11}x_1 + \dots + a_{1n}x_n = b_1$$

⋮

[49]

$$a_{m1}x_1 + \dots + a_{mn}x_n = b_m$$

(a) x_1, \dots, x_n are the **unknowns** of this system.

(b) Every n -tuple $(x_1, \dots, x_n) \in \mathbb{R}^n$ that satisfies this system is a **solution** of the linear equation system.

2. **compact notation:**

$$\begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \end{bmatrix} \mathbf{x}_1 + \begin{bmatrix} a_{12} \\ \vdots \\ a_{m2} \end{bmatrix} \mathbf{x}_2 + \dots + \begin{bmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix} \mathbf{x}_n = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \iff \underbrace{\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_x = \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_b$$

Note:

1. In general, for a real-valued system of linear equations we obtain either no, exactly one, or infinitely many solutions. [49]

2. **Geometric Interpretation of Systems of Linear Equations:** In a system of linear equations with two variables x_1, x_2 , each linear equation defines a line on the x_1x_2 -plane. Since a solution to a system of linear equations must satisfy all equations simultaneously, the solution set is the intersection of these lines. This intersection set can be a line (if the linear equations describe the same line), a point, or empty (when the lines are parallel). [49]

Similarly, for three variables, each linear equation determines a plane in three-dimensional space. When we intersect these planes, i.e., satisfy all linear equations at the same time, we can obtain a solution set that is a plane, a line, a point or empty (when the planes have no common intersection). [49]

3. the product Ax is a (linear) combination of the columns of A [49]

4. **Augmented Matrix** ($[A|b]$):

$$\left[\begin{array}{ccc|c} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} & b_m \end{array} \right]$$

5. The solution set of a homogeneous system of linear equations $A\mathbf{x} = \mathbf{0}$ with n unknowns $\mathbf{x} = [x_1, \dots, x_n]^\top$ is a subspace of \mathbb{R}^n . [49]

6. Every subspace $U \subseteq (\mathbb{R}^n, +, \cdot)$ is the solution space of a homogeneous system of linear equations $\mathbf{Ax} = \mathbf{0}$ for $\mathbf{x} \in \mathbb{R}^n$. [49]
7. The solution of an inhomogeneous system of linear equations $\mathbf{Ax} = \mathbf{b}, \mathbf{b} \neq \mathbf{0}$ is not a subspace of \mathbb{R}^n . [49]

17.1. Types of Solutions

1. Unique Solution:

- (a) The system has exactly one solution. [7]
- (b) **Graphically:** The lines (or planes) intersect at a single point. [7]
- (c) **Algebraically:** The equations are independent and consistent. [7]

2. Infinite Solutions:

- (a) The system has infinitely many solutions. [7]
- (b) **Graphically:** The lines (or planes) are exactly the same (i.e., they overlap). [7]
- (c) **Algebraically:** The equations are dependent and consistent. [7]

3. No Solution:

- (a) The system has no common solution. [7]
- (b) **Graphically:** The lines are parallel (in 2D) and never meet. [7]
- (c) **Algebraically:** The system is inconsistent. [7]

17.2. Finding Solutions using Matrix Inverse

1. if \mathbf{A} is a square matrix and invertible (strict conditions): $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ [49]
2. if \mathbf{A} has linearly independent columns (mild assumptions): $\mathbf{x} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{b}$ [49]
3. for reasons of numerical precision it is generally not recommended to compute the inverse or pseudo-inverse [49]

17.3. Finding Solutions using Elementary Transformations

1. keep the solution set the same, but that transform the equation system into a simpler form [49]
2. Operations:
 - (a) Exchange of two equations (rows in the matrix representing the system of equations) [49]
 - (b) Multiplication of an equation (row) with a constant $\lambda \in \mathbb{R} \setminus \{0\}$ [49]
 - (c) Addition of two equations (rows) [49]
3. Disadvantages:
 - (a) for systems with millions of variables, it is impractical as the required number of arithmetic operations scales cubically in the number of simultaneous equations. [49]

17.3.1. Row-Echelon Form (REF)

1. **Definition 17.1** (REF: pivot). The leading coefficient of a row (first nonzero number from the left) is called the pivot and is always strictly to the right of the pivot of the row above it. [49]
2. any equation system in row-echelon form always has a “staircase” structure [49]
3. A matrix is in row-echelon form if
 - (a) All rows that contain only zeros are at the bottom of the matrix; correspondingly, all rows that contain at least one nonzero element are on top of rows that contain only zeros. [49]
 - (b) Looking at nonzero rows only, the first nonzero number from the left (also called the **pivot** or the **leading coefficient**) is always strictly to the right of the pivot of the row above it. [49]
4. The variables corresponding to the pivots in the row-echelon form are called basic variables and the other variables are free variables. [49]
5. we express the right-hand side of the equation system using the pivot columns, such that $\mathbf{b} = \sum_{i=1}^P \lambda_i \mathbf{p}_i$, where \mathbf{p}_i , $i = 1, \dots, P$, are the pivot columns.
The λ_i are determined easiest if we start with the rightmost pivot column and work our way to the left. [49]

17.3.2. Reduced Row-Echelon Form (RREF)/ row-reduced echelon form/ row canonical form [49]

1. An equation system is in reduced row-echelon form if:
 - (a) It is in row-echelon form. [49]
 - (b) Every pivot is 1. [49]
 - (c) The pivot is the only non-zero entry in its column. [49]
2. Gaussian elimination is an algorithm that performs elementary transformations to bring a system of linear equations into reduced row-echelon form. [49]

17.3.3. Particular Solution/ Special solution

Term	Scope	Context
Unique Solution	One and only one solution	Systems of equations
Particular Solution	One of possibly many solutions	Differential equations, infinite solution systems

Unique Solution VS Particular Solution [7]

1. this is not the only solution of this system of linear equations. [49]
2. To capture all the other solutions, we need to be creative in generating 0 in a non-trivial way using the columns of the matrix: Adding 0 to our special solution **does not** change the special solution. [49]

17.3.4. Finding Solutions to $Ax = 0$: Minus-1 Trick [49]

Let \mathbf{A} is in reduced row-echelon form without any rows that just contain zeros:

$$A = \begin{bmatrix} 0 & \cdots & 0 & \mathbf{1} & * & \cdots & * & 0 & * & \cdots & * & 0 & * & \cdots & * \\ \vdots & & \vdots & 0 & 0 & \cdots & 0 & \mathbf{1} & * & \cdots & * & \vdots & \vdots & \vdots & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & 0 & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & 0 & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \mathbf{1} & * & \cdots & * \end{bmatrix} \in \mathbb{R}^{k \times n} \quad [49]$$

1. * can be an arbitrary real number [49]
2. constraints:
 - (a) first nonzero entry per row must be 1 [49]
 - (b) all other entries in the corresponding column must be 0 [49]
3. The columns j_1, \dots, j_k with the pivots (marked in **bold**) are the standard unit vectors $e_1, \dots, e_k \in \mathbb{R}^k$. [49]
4. We extend this matrix to an $n \times n$ -matrix \tilde{A} by adding $n - k$ rows of the form $[0 \ \cdots \ 0 \ -1 \ 0 \ \cdots \ 0]$ so that the diagonal of the augmented matrix \tilde{A} contains either 1 or -1. [49]
5. columns of \tilde{A} that contain the -1 as pivots are solutions of the homogeneous equation system $Ax = 0$. [49]
6. To be more precise, these columns form a basis of the solution space of $Ax = 0$, called the kernel or null space. [49]

17.3.5. General Solution (= Particular Solution + Solutions to $Ax = 0$)

Term	What it tells you	Context
Infinite Solutions	The quantity of solutions	Systems of equations
General Solution	A formula for all solutions	Differential equations, algebra

Infinite Solutions VS General Solution [\[7\]](#)

1. The general approach we followed consisted of the following three steps:
 - (a) Find a particular solution to $Ax = b$ [49]
 - (b) Find all solutions to $Ax = 0$ [49]
 - (c) Combine the solutions from steps 1. and 2. to the general solution. [49]

17.4. Approximate Solution

1. Projections allow us to look at situations where we have a linear system $Ax = b$ without a solution. This means that b does not lie in the span of A , i.e., the vector b does not lie in the subspace spanned by the columns of A . [49]
2. The idea is to find the vector in the subspace spanned by the columns of A that is closest to b , i.e., we compute the orthogonal projection of b onto the subspace spanned by the columns of A . This problem arises often in practice, and the solution is called the **least-squares solution** (assuming the dot product as the inner product) of an overdetermined system. [49]

17.5. Finding Solutions using stationary iterative methods - TODO

1. Let \mathbf{x}_* be a solution of $\mathbf{Ax} = \mathbf{b}$. [49]
2. The key idea of these iterative methods is to set up an iteration of the form $\mathbf{x}^{(k+1)} = C\mathbf{x}^{(k)} + d$ for suitable C and d that reduces the residual error $\|\mathbf{x}^{(k+1)} - \mathbf{x}_*\|$ in every iteration and converges to \mathbf{x}_* . [49]

17.5.1. Richardson method - TODO

17.5.2. Jacobi method - TODO

17.5.3. Gauß-Seidel method - TODO

17.5.4. successive over-relaxation method - TODO

17.6. Finding Solutions using Krylov subspace methods - TODO

17.6.1. conjugate gradients - TODO

17.6.2. generalized minimal residual - TODO

17.6.3. biconjugate gradients - TODO

V

ARTIFICIAL INTELLIGENCE
(AI)

CHAPTER 18

AI: INTRODUCTION

1. The field of artificial intelligence, or AI, attempts not just to **understand** but also to **build** intelligent entities. [8]
2. AI currently encompasses a huge variety of subfields, ranging from the general (learning and perception) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street, and diagnosing diseases. [8]
3. AI is relevant to any intellectual task; it is truly a universal field. [8]
4. A system is **rational** if it does the “right thing,” given what it knows. [8]

18.1. Approaches to AI

18.1.1. Acting humanly: The Turing Test approach

1. The **Turing Test**, proposed by **Alan Turing** (1950), was designed to provide a satisfactory operational definition of intelligence. [8]
2. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. [8]
3. The computer would need to possess the following capabilities:
 - (a) **Natural Language Processing** to enable it to communicate successfully in English
 - (b) **Knowledge Representation** to store what it knows or hears
 - (c) **Automated Reasoning** to use the stored information to answer questions and to draw new conclusions
 - (d) **Machine Learning** to adapt to new circumstances and to detect and extrapolate patterns
4. Turing’s test deliberately *avoided direct physical* interaction between the interrogator and the computer, because physical simulation of a person is unnecessary for intelligence. [8]
5. **Total Turing Test** includes a video signal so that the interrogator can test the subject’s perceptual abilities, as well as the opportunity for the interrogator to pass physical objects “through the hatch”. To pass the total Turing Test, the computer will additionally need:
 - (a) **computer vision** to perceive objects
 - (b) **robotics** to manipulate objects and move about

18.1.2. Thinking humanly: The cognitive modeling approach

1. Knowing the actual workings of human minds:
 - (a) **through introspection**: trying to catch our own thoughts as they go by [8]
 - (b) **through psychological experiments**: observing a person in action [8]
 - (c) **through brain imaging**: observing the brain in action [8]
2. If the program’s input–output behavior matches corresponding human behavior, that is evidence that some of the program’s mechanisms could also be operating in humans. [8]

3. The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind. [8]

18.1.3. Thinking rationally: The “laws of thought” approach

1. **Syllogisms:** It is a kind of logical argument that applies deductive reasoning to arrive at a conclusion based on two propositions that are asserted or assumed to be true. [10]
Example: All men are mortal; Socrates is a man; Therefore, Socrates is mortal. [10]
2. **Logic:** Logic is the study of correct reasoning. [9]
3. Emphasis is on correct inferences. [8]
4. **Challenges** with logicist approach:
 - (a) it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. [8]
 - (b) there is a big difference between solving a problem “in principle” and solving it in practice. Even problems with just a few hundred facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. [8]

18.1.4. Acting rationally: The rational agent approach

1. **Agent:** An agent is just something that acts: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals. [8]
2. **Rational Agent:** A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.
3. **Limited Rationality:** acting appropriately when there is not enough time to do all the computations one might like. [8]
4. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one’s goals and then to act on that conclusion. [8]
5. Correct inference is **not all** of rationality; in some situations, there is no provably correct thing to do, but something must still be done. [8]
6. There are also ways of acting rationally that cannot be said to involve inference.
Example: recoiling from a hot stove is a **reflex action** that is usually more successful than a slower action taken after careful deliberation. [8]
7. All the skills needed for the Turing Test also allow an agent to act rationally. [8]
8. **Knowledge representation** and **reasoning** enable agents to reach good decisions. [8]
9. We need learning not only for erudition, but also because it improves our ability to generate effective behavior. [8]
10. The standard of rationality is mathematically well defined and completely general, and can be “unpacked” to generate agent designs that provably achieve it. [8]
Human behavior, on the other hand, is well adapted for one specific environment and is defined by, well, the sum total of all the things that humans do. [8]
11. The rational-agent approach has two **advantages** over the other approaches:
 - (a) it is more general than the “laws of thought” approach because correct inference is just one of several possible mechanisms for achieving rationality [8]

Chapter 18. AI: Introduction

- (b) it is more amenable to scientific development than are approaches based on human behavior or human thought. [8]
- 12. Achieving *perfect rationality* - always doing the right thing - is **not feasible** in complicated environments. [8]

18.2. AI: Disciplines

18.2.1. Philosophy

Questions

- 1. Can formal rules be used to draw valid conclusions? [8]
- 2. How does the mind arise from a physical brain? [8]
- 3. Where does knowledge come from? [8]
- 4. How does knowledge lead to action? [8]

Notes

- 1. It's one thing to say that the mind operates, at least in part, according to logical rules, and to build physical systems that emulate some of those rules; it's another to say that the mind itself is such a physical system. [8]
- 2. One problem with a purely physical conception of the mind is that it seems to leave little room for free will: if the mind is governed entirely by physical laws, then it has no more free will than a rock "deciding" to fall toward the center of the earth. [8]
- 3. **Rationalism:** *Descartes* was a strong advocate of the power of reasoning in understanding the world [8]
- 4. **Dualism:** There is a part of the human mind (or soul or spirit) that is outside of nature, exempt from physical laws. Animals, on the other hand, did not possess this dual quality; they could be treated as machines. [8]
- 5. **Materialism:** It holds that the brain's operation according to the laws of physics constitutes the mind. Free will is simply the way that the perception of available choices appears to the choosing entity. [8]
- 6. **Empiricism Movement:** The empiricism movement, starting with **Francis Bacon**'s (1561–1626) *Novum Organum*, is characterized by a dictum of **John Locke** (1632–1704): "*Nothing is in the understanding, which was not first in the senses.*" [8]
- 7. **Principle of Induction:** general rules are acquired by exposure to repeated associations between their elements. [8]
- 8. **Logical Positivism:** This doctrine holds that all knowledge can be characterized by logical theories connected, ultimately, to observation sentences that correspond to sensory inputs; thus logical positivism combines rationalism and empiricism. [8]
- 9. **Confirmation Theory:** The confirmation theory of Carnap and Carl Hempel (1905–1997) attempted to analyze the acquisition of knowledge from experience. [8]

18.2.2. Mathematics

Questions

- 1. What are the formal rules to draw valid conclusions? [8]

- 2. What can be computed? [8]
- 3. How do we reason with uncertain information? [8]

Parts: Logic, Computation, Probability [8]

Notes

- 1. The idea of formal logic can be traced back to the philosophers of ancient Greece, but its mathematical development really began with the work of **George Boole** (1815–1864), who worked out the details of propositional, or Boolean, logic (Boole, 1847). [8]
- 2. In 1879, **Gottlob Frege** (1848–1925) extended Boole's logic to include objects and relations, creating the first-order logic that is used today. [8]
- 3. **Alfred Tarski** (1902–1983) introduced a theory of reference that shows how to relate the objects in a logic to objects in the real world. [8]

18.2.3. Economics

Questions

- 1. How should we make decisions so as to maximize payoff? [8]
- 2. How should we do this when others may not go along? [8]
- 3. How should we do this when the payoff may be far in the future? [8]

18.2.4. Neuroscience

Questions

- 1. How do brains process information? [8]

18.2.5. Psychology

Questions

- 1. How do humans and animals think and act? [8]

18.2.6. Computer engineering

Questions

- 1. How can we build an efficient computer? [8]

18.2.7. Control theory and cybernetics

Questions

- 1. How can artifacts operate under their own control? [8]

18.2.8. Linguistics

Questions

- 1. How does language relate to thought? [8]

18.3. AI: History

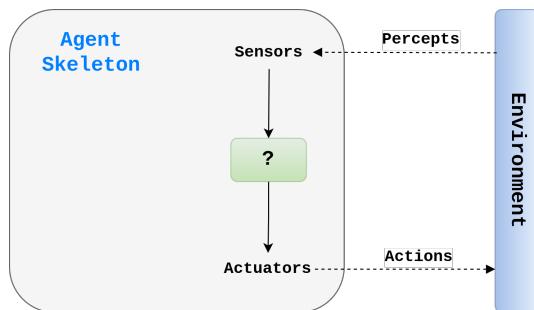
1. Minsky supervised a series of students who chose limited problems that appeared to require intelligence to solve. These limited domains became known as **microworlds**.

Date/ Time ↑	Events	Ref(s)
384–322 B.C.	Aristotle formulate a precise set of laws governing the rational part of the mind. He developed an informal system of syllogisms for proper reasoning, which in principle allowed one to generate conclusions mechanically, given initial premises.	[8]
350 BC	The Greek philosopher Aristotle was one of the first to attempt to codify “right thinking,” that is, irrefutable reasoning processes.	[8]
1315	Ramon Lull (d. 1315) had the idea that useful reasoning could actually be carried out by a mechanical artifact.	[8]
1452–1519	Leonardo da Vinci (1452–1519) designed but did not build a mechanical calculator; recent reconstructions have shown the design to be functional.	[8]
1588–1679	Thomas Hobbes (1588–1679) proposed that reasoning was like numerical computation, that “we add and subtract in our silent thoughts.”	[8]
1596–1650	Rene Descartes (1596–1650) gave the first clear discussion of the distinction between mind and matter and of the problems that arise.	[8]
1623	The first known calculating machine was constructed around 1623 by the German scientist Wilhelm Schickard (1592–1635)	[8]
1642	<i>Pascaline</i> , built in 1642 by Blaise Pascal (1623–1662), is more famous. Pascal wrote that “the arithmetical machine produces effects which appear nearer to thought than all the actions of animals.” Pascaline could only add and subtract.	[8]
1646–1716	Gottfried Wilhelm Leibniz (1646–1716) built a mechanical device intended to carry out operations on concepts rather than numbers, but its scope was rather limited. Leibniz did surpass Pascal by building a calculator that could add, subtract, multiply, and take roots.	[8]
1939–1945	Work in AI started after World War II	[8]
1943	The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943).	[8]
1950	Two undergraduate students at Harvard, Marvin Minsky and Dean Edmonds , built the first neural network computer in 1950. The <i>SNARC</i> , as it was called, used 3000 vacuum tubes and a surplus automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons.	[8]
1952	Arthur Samuel wrote a series of programs for checkers (draughts) that eventually learned to play at a strong amateur level. He disproved the idea that computers can do only what they are told to: his program quickly learned to play a better game than its creator.	[8]
1956	AI name was coined	[8]
1958	In MIT AI Lab Memo No. 1, McCarthy defined the high-level language Lisp , which was to become the dominant AI programming language for the next 30 years.	[8]

Date/ Time ↑	Events	Ref(s)
1959	Herbert Gelernter (1959) constructed the Geometry Theorem Prover , which was able to prove theorems that many students of mathematics would find quite tricky.	[8]
1961	General Problem Solver (GPS) : <i>Allen Newell and Herbert Simon</i> : They were not content merely to have their program solve problems correctly. They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems.	[8]
	1. McCarthy started the AI lab at Stanford.	
1963	2. James Slagle 's <i>Saint program</i> (1963) was able to solve closed-form calculus integration problems typical of first-year college courses.	[8]
1965	McCarthy's plan to use logic to build the ultimate Advice Taker was advanced by J. A. Robinson 's discovery in 1965 of the resolution method (a complete theorem-proving algorithm for first-order logic).	[8]
1967	Daniel Bobrow 's <i>Student program</i> (1967) solved algebra story problems.	[8]
1968	Tom Evans 's <i>Analogy program</i> (1968) solved geometric analogy problems that appear in IQ tests.	[8]

CHAPTER 19

AI: AGENTS



Agents interact with environments through sensors and actuators. [60]

1. **Agent:** An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. [8]
agent = architecture + (agent) program [8]
2. **Environment:** environment refers to everything outside the agent that it interacts with. [7]
The “geography” of the environment is known **a priori**. [8]
“A priori”: Knowledge or assumptions made before data is observed (e.g., predefined rules, constraints). [7]
3. **Sensors:** A sensor is any mechanism that allows an AI agent to perceive its environment by gathering information. This can be physical (hardware) or virtual (software). [7]
4. **Actuators:** An actuator is any mechanism that allows an AI agent to affect or change its environment by performing actions. [7]
5. **Percept:** agent’s perceptual inputs at any given instant [8]
6. **Percept Sequence:** it is the complete history of everything the agent has ever perceived. In general, an agent’s choice of action at any given instant can depend on the entire percept sequence observed to date, but **not** on anything it hasn’t perceived. [8]
7. **Agent Function:** maps any given percept sequence to an action; describes agent’s behavior; an external characterization of the agent; The agent function is an abstract mathematical description; takes the entire percept history [8]
8. **Agent Program:** internal implementation of the agent function for an artificial agent; the agent program is a concrete implementation, running within some physical system. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent’s actions need to depend on the entire percept sequence, the agent will have to remember the percepts. [8]
9. **Architecture:** computing device with physical sensors and actuators on which the agent program is running. [8]
10. **Performance Measure:** It evaluates any given sequence of environment states. [8]
11. **Rational Agent:** A rational agent is one that does the right thing [8]
For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. [8]
Rationality is not the same as perfection. [8]
Rationality maximizes **expected** performance, while perfection maximizes **actual** performance. [8]

Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence to date [8]

12. **Omniscient Agent:** An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality. [8]
13. **Information Gathering:** It is doing actions in order to modify future percepts. [8]
14. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. [8]
15. If we define success in terms of agent's opinion of its own performance, an agent could achieve perfect rationality simply by deluding itself that its performance was perfect. [8]
16. Human agents in particular are notorious for "sour grapes" - believing they did not really want something (e.g., a Nobel Prize) after not getting it. [8]
17. As a general rule, it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave. [8]
18. The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented. There are extreme cases in which the environment is completely known **a priori**. In such cases, the agent need not perceive or learn; it simply acts correctly. Such agents are fragile. [8]
19. To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent **lacks autonomy**. A rational agent should be **autonomous** - it should learn what it can to compensate for partial or incorrect prior knowledge. [8]
20. After sufficient experience of its environment, the behavior of a rational agent can become effectively **independent** of its prior knowledge. Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments. [8]

19.1. Task Environment/ Problem [8]

1. Task environments are essentially the "problems" to which rational agents are the "solutions." [8]

19.1.1. PEAS: Defining Problem

1. Performance:

- (a) Desirable qualities [8]
- (b) Defines how the success of the agent is evaluated. [7]

Example: In a self-driving car, performance can be measured by safety, fuel efficiency, and reaching the destination on time. [7]

2. Environment:

- (a) The surroundings in which the agent operates. [7]
- Example:** For a self-driving car, the environment includes roads, traffic, pedestrians, and weather conditions. [7]

3. Actuators:

- (a) The mechanisms that allow the agent to take action. [7]

Can be hardware or software.

Example: A self-driving car uses its steering wheel, accelerator, and brakes as actuators. [7]

4. Sensors:

(a) The components that allow the agent to perceive its environment. [7]

Can be hardware or software.

Example: A self-driving car has cameras, LiDAR, GPS, and speed sensors. [7]

Note:

1. some **software agents** (or **software robots** or **softbots**) exist in rich, unlimited domains. [8]

19.1.2. Properties of Task Environments [8]

19.1.2.1. Fully observable, partially observable, unobservable

1. **fully observable:** If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are **relevant** to the choice of action; relevance, in turn, depends on the performance measure. Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world. [8]
2. **partially observable:** An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data. [8]
3. **unobservable:** If the agent has no sensors at all then the environment is unobservable. The agent's goals may still be achievable, sometimes with certainty. [8]

19.1.2.2. Single agent, multi-agent

1. **Single agent:** Only 1 agent is interacting in the given environment.
 2. **Multi-agent:** More than 1 agent (of same type or different types) interact in the given environment.
 - (a) **Competitive Multi-agent:** Agents maximize their own performance at cost of other agents' performance.
Examples: 2 agents playing Chess against each other; taxi-driving (competing for parking space)
 - (b) **Cooperative Multi-agent:** Agents take actions to maximize collective performance.
Example: taxi-driving (avoiding collisions)
 - (c) **communication** often emerges as a rational behavior in multiagent environments; in some competitive environments, **randomized behavior** is rational because it avoids the pitfalls of predictability.
- [8]

19.1.2.3. Deterministic, stochastic, uncertain, nondeterministic

1. If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is **deterministic**; otherwise, it is **stochastic**. [8]
2. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. [8]
3. If the environment is partially observable, however, then it could appear to be stochastic. Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as **stochastic**.
It implies that uncertainty about outcomes is quantified in terms of *probabilities*. [8]
4. We say an environment is **uncertain** if it is not fully observable or not deterministic. [8]
5. A **nondeterministic** environment is one in which actions are characterized by their possible outcomes, but no probabilities are attached to them. Non-deterministic environment descriptions are usually associated with performance measures that require the agent to succeed for all possible outcomes of its actions. [8]

19.1.2.4. Episodic, sequential

1. In an **episodic** task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead. [8]

2. In **sequential** environments, on the other hand, the current decision could affect all future decisions. [8]

19.1.2.5. Static, dynamic, semidynamic

1. If the environment can change while an agent is deliberating, then we say the environment is **dynamic** for that agent; otherwise, it is **static**. [8]
2. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. [8]
3. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing. [8]
4. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**. [8]

19.1.2.6. Discrete, continuous

1. The discrete/continuous distinction applies to the state of the environment, to the way *time* is handled, and to the percepts and actions of the agent. [8]

19.1.2.7. Known, unknown

1. Strictly speaking, this distinction refers not to the environment itself but to the agent's (or designer's) state of knowledge about the "laws of physics" of the environment. [8]
2. In a **known** environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given. [8]
3. if the environment is **unknown**, the agent will have to learn how it works in order to make good decisions. [8]

Examples [8]

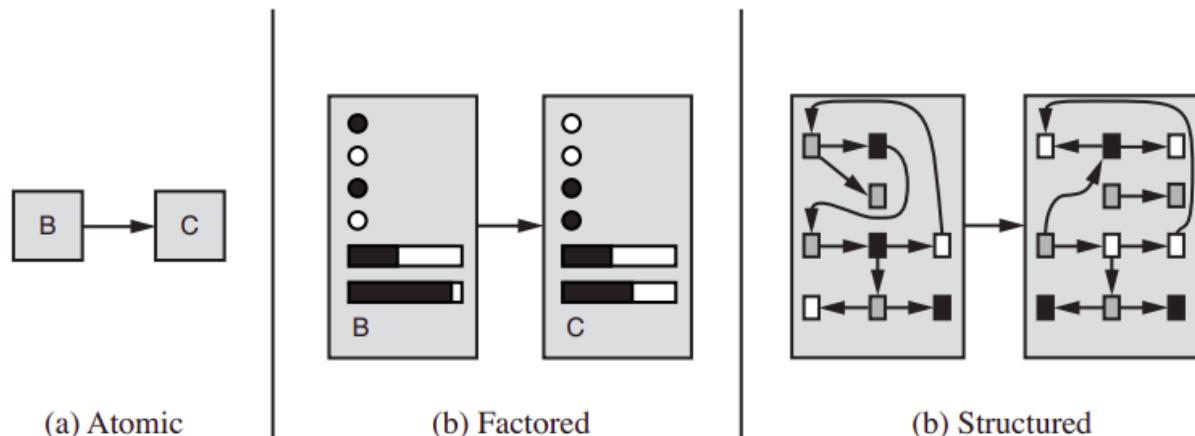
Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Note

1. the distinction between known and unknown environments is not the same as the one between fully and partially observable environments. [8]
2. It is quite possible for a **known** environment to be **partially observable**—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over. [8]
3. an **unknown** environment can be **fully observable**—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them. [8]

4. **environment class:** A category or set of related environments from which specific environments are drawn for testing or training agents. [7, 8]
5. **environment generator:** An environment generator is a tool or function that creates individual environments sampled from an environment class. It automates the creation of diverse scenarios for training or evaluating agents. [7, 8]

19.2. Agent State Representations



Three ways to represent states and the transitions between them.

- (a) **Atomic representation:** a state (such as B or C) is a black box with no internal structure; [8]
- (b) **Factored representation:** a state consists of a vector of attribute values; values can be Boolean, realvalued, or one of a fixed set of symbols. [8]
- (c) **Structured representation:** a state includes objects, each of which may have attributes of its own as well as relationships to other objects. [8]

1. Roughly speaking, a more expressive representation can capture, at least as concisely, everything a less expressive one can capture, plus some more. [8]
2. Often, the more expressive language is much more concise; for example, the rules of chess can be written in a page or two of a structured-representation language such as first-order logic but require thousands of pages when written in a factored-representation language such as propositional logic. [8]
3. reasoning and learning become more complex as the expressive power of the representation increases. [8]
4. To gain the benefits of expressive representations while avoiding their drawbacks, intelligent systems for the real world may need to operate at all points along the axis simultaneously. [8]

19.2.1. Atomic representation

1. each state of the world is indivisible - it has no internal structure. [8]
2. two different atomic states have nothing in common—they are just different black boxes [8]
3. Algorithms/ areas that work with atomic representations - or, at least, they treat representations as if they were atomic:
 - (a) Search and game-playing [8]
 - (b) Hidden Markov models [8]
 - (c) Markov decision processes [8]

19.2.2. Factored representation

1. A factored representation splits up each state into a fixed set of **variables** or **attributes**, each of which can have a **value**. [8]
2. two different factored states can **share** some attributes (such as being at some particular GPS location) and not others (such as having lots of gas or having no gas)
this makes it much easier to work out how to turn one state into another [8]
3. With factored representations, we can also represent uncertainty - for example, ignorance about the amount of gas in the tank can be represented by leaving that attribute blank. [8]
4. Algorithms/ areas based on factored representations:
 - (a) constraint satisfaction algorithms [8]
 - (b) propositional logic [8]
 - (c) planning [8]
 - (d) Bayesian networks [8]

19.2.3. Structured representation

1. the world as having things in it that are related to each other, not just variables with values. [8]
2. almost everything that humans express in natural language concerns objects and their relationships [8]
3. Algorithms/ areas based on factored representations:
 - (a) relational databases [8]
 - (b) first-order logic [8]
 - (c) first-order probability models [8]
 - (d) knowledge-based learning [8]
 - (e) natural language understanding [8]

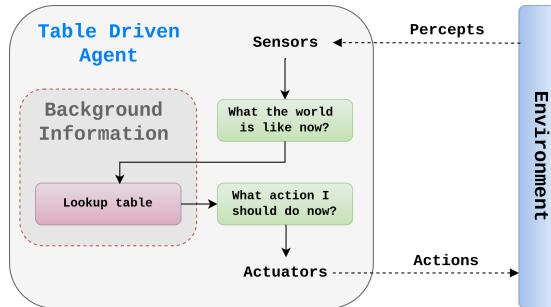
```
1 class State:  
2     def copy(self):  
3         raise NotImplementedError()
```

Python Snippet 19.1: Problem Solving Agent - State skeleton

CHAPTER 20

AI: AGENT PROGRAMS

20.1. Table Driven Agent



Schematic diagram of a table driven agent. [60]

\mathcal{P} set of possible percepts
 T lifetime of agent (total number of percepts)

1. Number of entries in table: $\sum_{t=1}^T |\mathcal{P}|^t$ [8]

2. Sometimes the number of entries can be ridiculously large number due to the combinations of percepts and length of percept sequence.

Example: Consider the *automated taxi*: the visual input from a single camera comes in at the rate of roughly 27 megabytes per second (30 frames per second, 640×480 pixels with 24 bits of color information). This gives a lookup table with over $10^{250,000,000,000}$ entries for an hour's driving. [8]
It means:

- (a) no physical agent in this universe will have the space to store the table [8]
- (b) the designer would not have time to create the table [8]
- (c) no agent could ever learn all the right table entries from its experience [8]
- (d) even if the environment is simple enough to yield a feasible table size, the designer still has no guidance about how to fill in the table entries [8]

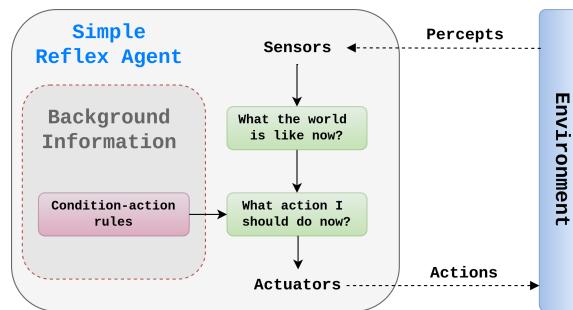
Algorithm 20.1: The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory. [8]

```

1 function TABLE-DRIVEN-AGENT(percept) returns an action
2   persistent:
3     percepts, a sequence, initially empty
4     table, a table of actions, indexed by percept sequences, initially fully specified
5
6   append percept to the end of percepts
7   action  $\leftarrow$  LOOKUP(percepts, table)
8   return action

```

20.2. Simple Reflex Agent



Schematic diagram of a simple reflex agent. [60]

1. simplest kind of agent [8]
2. These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. [8]
3. **condition-action rule/ situation-action rules/ productions/ if-then rules:**
A condition-action rule is a basic decision-making structure used in reflex agents and rule-based systems. [7, 8]
4. the description in terms of “rules” and “matching” is purely conceptual; actual implementations can be as simple as a collection of logic gates implementing a Boolean circuit. [8]
5. Escape from infinite loops is possible if the agent can **randomize** its actions. A randomized simple reflex agent *might outperform* a deterministic simple reflex agent. Randomized behavior of the right kind can be rational in some multi-agent environments. In single-agent environments, randomization is usually **not** rational. [8]
6. **Disadvantage:** limited intelligence [8]
7. **Disadvantage:** will work only if the correct decision can be made on the basis of only the current percept—that is, only if the environment is fully observable. Even a little bit of un-observability can cause serious trouble. [8]
8. **Disadvantage:** we would have to rewrite many condition–action rules for supporting more situations/scenarios.

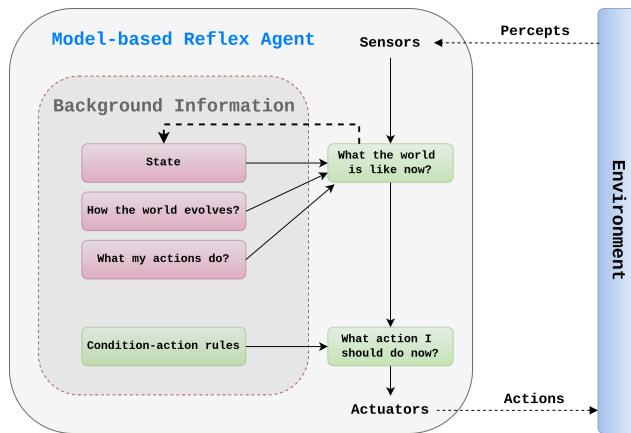
Algorithm 20.2: SIMPLE-REFLEX-AGENT: It acts according to a rule whose condition matches the current state, as defined by the percept. [8]

```

1 function SIMPLE-REFLEX-AGENT( percept ) returns an action
2   persistent: rules, a set of condition-action rules
3
4   state  $\leftarrow$  INTERPRET-INPUT( percept )
5   rule  $\leftarrow$  RULE-MATCH( state, rules )
6   action  $\leftarrow$  rule.ACTION
7   return action
  
```

INTERPRET-INPUT generates an abstracted description of the current state from the percept [8]
 RULE-MATCH returns the first rule in the set of rules that matches the given state description [8]

20.3. Model-based reflex agents



A model-based reflex agent. [60]

1. The most effective way to handle partial observability is for the agent to keep track of the part of the world it can't see now. [8]
2. the agent should maintain some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. [8]
3. Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program.
 - (a) we need some information about how the world evolves independently of the agent [8]
 - (b) we need some information about how the agent's own actions affect the world [8]
4. knowledge about “how the world works” - whether implemented in simple Boolean circuits or in complete scientific theories - is called a model of the world. [8]
5. An agent that uses such a model is called a model-based agent. [8]
6. Regardless of the kind of representation used, it is **seldom** possible for the agent to determine the current state of a partially observable environment *exactly*. [8]
7. internal “state” maintained by a model-based agent **does not** have to describe “what the world is like now” in a literal sense. [8]
8. **Disadvantage:** Knowing something about the current state of the environment is **not always enough** to decide what to do. [8]
9. **Disadvantage:** we would have to rewrite many condition–action rules for supporting more situations/scenarios.

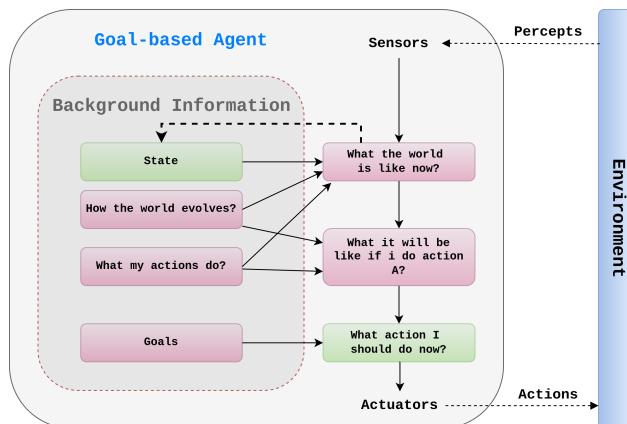
Algorithm 20.3: MODEL-BASED-REFLEX-AGENT: A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent. [8]

```

1 function MODEL-BASED-REFLEX-AGENT(percept) returns an action
2   persistent:
3     state, the agent's current conception of the world state
4     model, a description of how the next state depends on current state and action
5     rules, a set of condition-action rules
6     action, the most recent action, initially none
7
8   state  $\leftarrow$  UPDATE-STATE( state, action, percept, model )
9   rule  $\leftarrow$  RULE-MATCH( state, rules )
10  action  $\leftarrow$  rule.ACTION
11  return action
```

1. UPDATE-STATE: responsible for creating the new internal state description. The details of how models and states are represented vary widely depending on the type of environment and the particular technology used in the agent design. [8]

20.4. (Model-based) Goal-based Agents

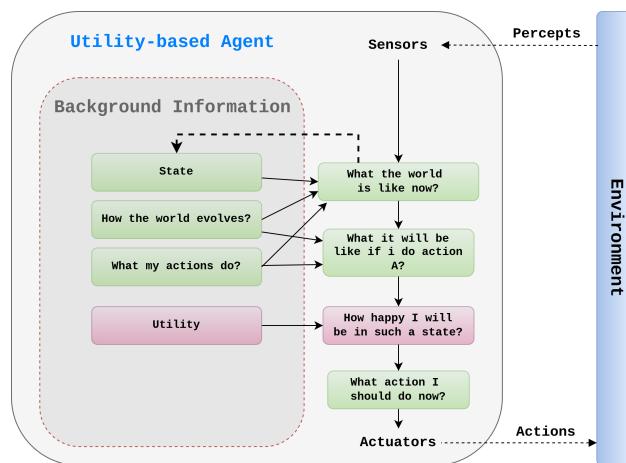


A model-based, goal-based agent. [60]

1. As well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable. [8]
2. The agent program can combine this with the model (the same information as was used in the model-based reflex agent) to choose actions that achieve the goal. [8]
3. Sometimes goal-based action selection is straightforward - for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky - for example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal. [8]
4. decision making of this kind is fundamentally different from the condition-action rules
decision making involves consideration of the future - both “*What will happen if I do such-and-such?*” and “*Will that make me happy?*”. [8]
5. It is **not reflexive**. It takes calculated decisions.
6. Although the goal-based agent appears less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified. [8]

7. Goals just provide a crude binary distinction between “happy” and “unhappy” states. [8]
8. Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider. [8]
9. SEE: subfields of AI devoted to finding action sequences that achieve the agent’s goals: (TODO)
 - (a) Search
 - (b) Planning

20.5. (Model-based Goal-based) Utility-based Agents



A model-based, goal-based, utility-based agent. [60]

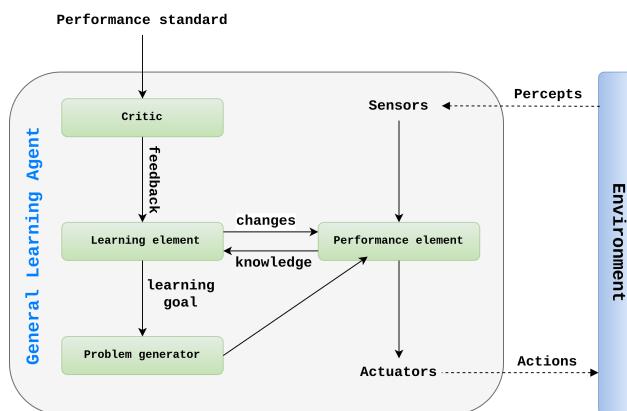
1. A more general performance measure than Goal (happy/ unhappy) should allow a comparison of different world states according to exactly how happy they would make the agent. [8]
2. An agent’s **utility function** is essentially an internalization of the performance measure. [8]
(The word “utility” here refers to “*the quality of being useful*”)
3. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure. [8]
4. this is **not the only** way to be rational but, like goal-based agents, a utility-based agent has many advantages in terms of flexibility and learning [8]
5. (**Advantage**) in two kinds of cases, goals are inadequate but a utility-based agent can still make rational decisions:
 - (a) when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff [8]
 - (b) when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals. [8]
6. Partial observability and stochasticity are ubiquitous in the real world, and so, therefore, is decision making under uncertainty. [8]
7. a rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes - that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome. [8]

8. An agent that possesses an explicit utility function can make rational decisions with a general-purpose algorithm that does not depend on the specific utility function being maximized. In this way, the “global” definition of rationality - designating as rational those agent functions that have the highest performance - is turned into a “local” constraint on rational-agent designs that can be expressed in a simple program. [8]
9. A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning. [8]
10. Choosing the utility-maximizing course of action is also a difficult task, requiring ingenious algorithms. [8]

Aspect	Utility Function	Performance Measure
Perspective	From the agent’s point of view	From the external evaluator’s point of view
Goal	Maximize expected utility	Maximize measurable performance
Usage	Guides decision-making	Evaluates overall system success

Utility Function VS Performance Measure

20.6. Learning agents



A general learning agent. [60]

1. In his famous early paper, **Turing** (1950) considers the idea of actually programming his intelligent machines by hand. [8]
2. **advantage:** it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. [8]
3. **conceptual components:**
 - (a) **learning element:** responsible for making improvements [8]
 - (b) **performance element:** responsible for selecting external actions. it takes in percepts and decides on actions. [8]

The design of the learning element depends very much on the design of the performance element. When trying to design an agent that learns a certain capability, the first question is **not** “How am I going to get it to learn this?” **but** “What kind of performance element will my agent need to do this once it has learned how?” [8]

It is important that the performance standard be fixed. Conceptually, one should think of it as being outside the agent altogether because the agent **must not** modify it to fit its own behavior. [8]

- (c) **critic:** The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future. [8]
The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide **no** indication of the agent's success. [8]
- (d) **problem generator:** It is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. [8]
The problem generator might identify certain areas of behavior in need of improvement and suggest experiments. [8]
4. The learning element can make changes to any of the “*knowledge*” components in the previous agents(simple reflex, model based, goal based, utility based). The simplest cases involve learning directly from the percept sequence. Observation of pairs of successive states of the environment can allow the agent to learn “How the world evolves,” and observation of the results of its actions can allow the agent to learn “What my actions do.”. The forms of learning do not need to access the external performance standard. [8]
 5. In a sense, the **external performance standard** is the *universal* one of making predictions that agree with experiment. [8]
 6. In a sense, the performance standard distinguishes part of the incoming percept as a **reward** (or **penalty**) that provides direct feedback on the quality of the agent’s behavior. Hard-wired performance standards such as pain and hunger in animals can be understood in this way. [8]
 7. Learning in intelligent agents can be summarized as a process of modification of each component of the agent to bring the components into closer agreement with the available feedback information, thereby improving the overall performance of the agent. [8]
 8. Intelligent agents are supposed to maximize their performance measure. [8]

20.7. Problem-Solving Agent

1. one kind of *goal-based agent* that use **atomic** representations [8]
2. **uninformed search algorithms:** algorithms that are given **no information** about the problem other than its definition [8]
some of these algorithms can solve any solvable problem, none of them can do so efficiently [8]
3. **Informed search algorithms:** can do quite well given some guidance on where to look for solutions. [8]
4. **Goal formulation**, based on the current situation and the agent’s performance measure, is the first step in problem solving. [8]
5. We consider a goal to be a set of world states—exactly those states in which the goal is satisfied. The agent’s task is to find out how to act, now and in the future, so that it reaches a goal state. [8]

Simple “formulate, search, execute” design

6. **Problem formulation** is the process of deciding what actions and states to consider, given a goal. [8]

7. An agent with several immediate options of unknown value can decide what to do by first examining future actions that eventually lead to states of known value. [8]
8. The process of looking for a sequence of actions that reaches the goal is called **search**. [8]
9. A *search algorithm* takes a problem as input and returns a **solution** in the form of an action sequence. [8]
10. Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase. While the agent is executing the solution sequence it *ignores* its percepts when choosing an action because it knows in advance what they will be. [8]
11. Once the solution has been executed, the agent will formulate a new goal. [8]
12. An agent that carries out its plans with its eyes closed, so to speak, must be quite certain of what is going on. Control theorists call this an **open-loop system**, because ignoring the percepts breaks the loop between agent and environment. [8]

Algorithm 20.4: A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over [8]

```

1 function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
2   persistent:
3     seq, an action sequence, initially empty
4     state, some description of the current world state
5     goal, a goal, initially null
6     problem, a problem formulation
7
8     state  $\leftarrow$  UPDATE-STATE(state, percept)
9     if seq is empty then
10      goal  $\leftarrow$  FORMULATE-GOAL(state)
11      problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
12      seq  $\leftarrow$  SEARCH(problem)
13      if seq = failure then
14        return a null action
15
16      action  $\leftarrow$  FIRST(seq)
17      seq  $\leftarrow$  REST(seq)
18      return action
```

20.7.1. Defining & Formulating Problem

1. The **initial state** that the agent starts in. [8]
2. A description of the possible **actions** available to the agent. Given a particular state *s*, ACTIONS(*s*) returns the set of actions that can be executed in *s*. We say that each of these actions is *applicable* in *s*. [8]
3. A description of what each action does; the formal name for this is the **transition model**, specified by a function RESULT(*s*, *a*) that returns the state that results from doing action *a* in state *s*. We also use the term **successor** to refer to any state reachable from a given state by a single action. [8]
4. The **goal test**, which determines whether a given state is a goal state. Sometimes there is an *explicit* set of possible goal states, and the test simply checks whether the given state is one of them. Sometimes the goal is specified by an *abstract property* rather than an explicitly enumerated set of states. [8]

5. A **path cost** function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure. We assume that the cost of a path can be described as the *sum* of the costs of the individual actions along the path. [8]

Note:

1. Together, the initial state, actions, and transition model implicitly define the **state space** of the problem—the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or **graph** in which the nodes are states and the links between nodes are actions. [8]
 2. **diameter of the state space:** The maximum number of steps (or actions) required to go from any one state to any other reachable state in the state space. [8]
 3. A **path** in the state space is a sequence of states connected by a sequence of actions. [8]
 4. The **step cost** of taking action a in state s to reach state s' is denoted by $c(s, a, s')$. [8]
 5. A **solution** to a problem is an *action sequence* that leads from the initial state to a goal state. [8]
 6. Solution quality is measured by the path cost function, and an **optimal solution** has the lowest path cost among all solutions. [8]
 7. In addition to abstracting the state description, we must abstract the actions themselves. [8]
 8. The abstraction is **valid** if we can expand any abstract solution into a solution in the more detailed world [8]
 9. The abstraction is **useful** if carrying out each of the actions in the solution is easier than the original problem [8]
 10. **meta-level state space:** Each **state** in a meta-level state space captures the internal (computational) state of a program that is searching in an **object-level state space**. Each **action** in the meta-level state space is a computation step that alters the internal state. A sequence of larger and larger search trees, can be seen as depicting a path in the meta-level state space where each state on the path is an object-level search tree [8]
- Example:** the internal state of the A* algorithm consists of the current search tree. each computation step in A* expands a leaf node and adds its successors to the tree [8]
11. **meta-level learning algorithm** can learn from experiences (missteps in harder problems) to avoid exploring unpromising sub-trees. The goal of learning is to *minimize* the **total cost** of problem solving, trading off computational expense and path cost. [8]
 12. **relaxed problem:** A problem with *fewer restrictions* on the actions [8]
The state-space graph of the relaxed problem is a **super-graph** of the original state space because the removal of restrictions creates added edges in the graph. [8]
Because the relaxed problem adds edges to the state space, any optimal solution in the original problem is, by definition, also a solution in the relaxed problem; but the relaxed problem may have **better solutions** if the added edges provide **short cuts**. Hence, *the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.* [8]
because the derived heuristic is an exact cost for the relaxed problem, it must obey the triangle inequality and is therefore **consistent** [8]
 13. it is crucial that the relaxed problems generated by this technique can be solved essentially **without search**, because the relaxed rules allow the problem to be decomposed into some independent subproblems. If the relaxed problem is hard to solve, then the values of the corresponding heuristic will be **expensive** to obtain. [8]

```

1 class Problem:
2     def __init__(self, initial_state: State):

```

```

3     self.initial_state = initial_state
4     # 1. initial_state: The initial state that the agent starts in.
5
6     def actions(self, state: State):
7         """
8             2. A description of the possible actions available to the agent.
9                 Given a particular state s, ACTIONS(s) returns the set of actions
10                that can be executed in s.
11
12            """
13
14     def result(self, state: State, action):
15         """
16             3. A description of what each action does; the formal name for this
17                 is the transition model, specified by a function RESULT(s, a) that
18                 returns the state that results from doing action a in state s.
19
20            """
21
22     def goal_test(self, state: State):
23         # 4. The goal test, which determines whether a given state is a goal state.
24         raise NotImplementedError()
25
26     def step_cost(self, state: State, action, new_state: State):
27         """
28             5. The step cost of taking action
29                 a in state s to reach state s' is denoted by c(s, a, s').
30
31            """
32
33     def heuristic(self, state: State):
34         """
35             Returns estimated cost from state to goal.
36
37            """
38
39            raise NotImplementedError()

```

Python Snippet 20.1: Problem Solving Agent - Problem Skeleton

20.7.2. Measuring problem-solving performance

1. **Completeness:** Is the algorithm guaranteed to find a solution when there is one? [8]
2. **Optimality:** Does the strategy find the optimal solution? [8]
3. **Time complexity:** How long does it take to find a solution? [8]
4. **Space complexity:** How much memory is needed to perform the search? [8]

V	set	set of vertices (nodes) of the graph
E	set	set of edges (links) of the graph
b	$\in \mathbb{R}$	branching factor or maximum number of successors of any node
d	$\in \mathbb{R}$	depth of the shallowest goal node (i.e., the number of steps along the path from the root)
m	$\in \mathbb{R}$	maximum length of any path in the state space

Notations

Note:

1. the typical measure is the **size** of the **state space graph**: $|V| + |E|$ [8]
2. Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory. [8]
3. **effectiveness of a search algorithm:**
 - (a) **search cost**: depends on the time complexity but can also include a term for memory usage [8]
 - (b) **total cost**: combines the search cost and the path cost of the solution found [8]
4. In general, exponential-complexity search problems **cannot be solved** by uninformed methods for any but the smallest instances.

20.8. Planning Agents

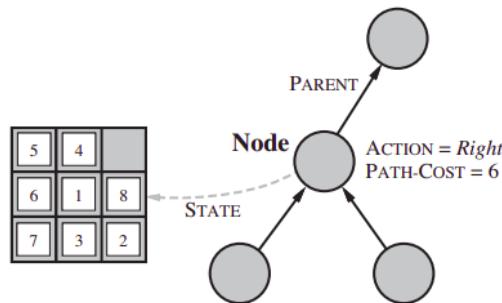
1. One kind of *goal-based agents* that use more advanced **factored** or **structured** representations [8]

CHAPTER 21

AI: SEARCHING SOLUTIONS

1. A solution is an action sequence, so search algorithms work by considering various possible action sequences. [8]
2. The possible action sequences starting at the initial state form a **search tree** with the initial state at the root; the branches are actions and the **nodes** correspond to states in the state space of the problem. [8]
3. We consider taking various actions by **expanding** the current state; that is, applying each legal action to the current state, thereby **generating** a new set of states. The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand. [8]
4. **leaf node**: a node with **no** children in the tree. [8]
5. **frontier/ open list**: set of all leaf nodes available for expansion at any given point [8]
6. Search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next - the so-called **search strategy**. [8]
7. Considering **loopy paths** means that the complete search tree is **infinite** because there is no limit to how often one can traverse a loop. loops can cause certain algorithms to fail, making otherwise solvable problems **unsolvable**. [8]
8. In some cases, **redundant paths** are *unavoidable*. This includes all problems where the actions are reversible, such as route-finding problems and sliding-block puzzles. Following redundant paths can cause a tractable problem to become **intractable**. This is true even for algorithms that know how to avoid infinite loops. [8]

21.1. Designing Search Node



Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent. [8]

For each NODE n of the tree, we have a structure that contains four components:

1. $n.\text{STATE}$: the state in the state space to which the node corresponds [8]
2. $n.\text{PARENT}$: the node in the search tree that generated this node [8]
3. $n.\text{ACTION}$: the action that was applied to the parent to generate the node [8]
4. $n.\text{PATH-COST}$: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers [8]

Note:

1. A node is a bookkeeping data structure used to represent the search tree. A state corresponds to a configuration of the world. Thus, nodes are on particular paths, as defined by PARENT pointers, whereas states are not. Furthermore, two different nodes can contain the same world state if that state is generated via two different search paths. [8]
2. The PARENT pointers string the nodes together into a tree structure. These pointers also allow the solution path to be extracted when a goal node is found. [8]
3. SOLUTION function is used to return the sequence of actions obtained by following parent pointers back to the root. [8]

```

1 class Node:
2     def __init__(self, state, parent, action, path_cost):
3         # the state in the state space to which the node corresponds
4         self.state = state
5
6         # the node in the search tree that generated this node
7         self.parent = parent
8
9         # the action that was applied to the parent to generate the node
10        self.action = action
11
12        """
13            the cost, traditionally denoted by g(n), of the path
14            from the initial state to the node, as indicated
15            by the parent pointers
16        """
17        self.path_cost = path_cost
18
19    def __lt__(self, __o):
20        return self.path_cost < __o.path_cost
21
22    def __str__(self):
23        return (
24            "<Node "
25            + f"state: {self.state} "
26            + f"parent: {self.parent} "
27            + f"action: {self.action} "
28            + f"path_cost: {self.path_cost} "
29            + ">"
30        )
31
32    def __repr__(self) -> str:
33        return str(self)

```

Python Snippet 21.1: Problem Solving Agent - Search Node

```

1 def solution(node: Node):
2     path = []
3
4     while node.parent is not None:
5         path.insert(0, node.action)
6         node = node.parent
7

```

```
8     return path
```

Python Snippet 21.2: Problem Solving Agent - solution

21.2. General Algorithms & Implementations

21.2.1. Child-node

Algorithm 21.1: The function CHILD-NODE takes a parent node and an action and returns the resulting child node [8]

```
1 function CHILD-NODE(problem) returns a NODE
2     return a node with
3         STATE = problem.RESULT(parent.STATE, action),
4         PARENT = parent,
5         ACTION = action,
6         PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

```
1 def child_node(problem: Problem, parent: Node, action):
2     new_state = problem.result(parent.state, action)
3
4     return Node(
5         state=new_state,
6         parent=parent,
7         action=action,
8         path_cost=(parent.path_cost
9                     + problem.step_cost(parent.state, action, new_state))
10    )
```

Python Snippet 21.3: Problem Solving Agents - child_node

21.2.2. Tree Search

Algorithm 21.2: An informal description of the general tree-search algorithm. [8]

```
1 function TREE-SEARCH(problem) returns a solution, or failure
2     initialize the frontier using the initial state of problem
3
4     while do
5         if the frontier is empty then
6             return failure
7
8         choose a leaf node and remove it from the frontier
9         if the node contains a goal state then
10            return the corresponding solution
11
12        expand the chosen node, adding the resulting nodes to the frontier
```

```
1 def tree_search(problem):
2     frontier = [Node(problem.initial_state, None, None, 0)]
3
4     while True:
5         if len(frontier) == 0:
6             return None
```

```

7     node: Node = frontier.pop()
8
9
10    if problem.goal_test(node.state):
11        return solution(node)
12
13    for action in problem.actions(node.state):
14        new_state = problem.result(node.state, action)
15        path_cost = (node.path_cost
16                    + problem.step_cost(node.state, action, new_state))
17        new_node = Node(new_state, node, action, path_cost)
18        frontier.append(new_node)

```

Python Snippet 21.4: Problem Solving Agents - tree_search

21.2.3. Graph search

Algorithm 21.3: An informal description of the general graph-search algorithm. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states. [8]

```

1 function GRAPH-SEARCH(problem) returns a solution, or failure
2     initialize the frontier using the initial state of problem
3     initialize the explored set to be empty
4
5     while do
6         if the frontier is empty then
7             return failure
8         choose a leaf node and remove it from the frontier
9         if the node contains a goal state then
10            return the corresponding solution
11         add the node to the explored set
12         if chosen node not in the frontier or explored set then
13             expand the chosen node, adding the resulting nodes to the frontier

```

```

1 def graph_search(problem: Problem):
2     frontier = [Node(problem.initial_state, None, None, 0)]
3     explored = set()
4
5     while True:
6         if len(frontier) == 0:
7             return None
8
9         node: Node = frontier.pop()
10
11        if problem.goal_test(node.state):
12            return solution(node)
13
14        explored.add(node)
15
16        for action in problem.actions(node.state):
17            new_state = problem.result(node.state, action)
18            path_cost = (node.path_cost
19                         + problem.step_cost(node.state, action, new_state))

```

```
20     new_node = Node(new_state, node, action, path_cost)
21
22     if new_node not in frontier and new_node not in explored:
23         frontier.append(new_node)
```

Python Snippet 21.5: Problem Solving Agents - graph_search

1. **explored set/ closed list:** remembers every expanded node [8]
2. Newly generated nodes that match previously generated nodes - ones in the explored set or the frontier - can be discarded instead of being added to the frontier. [8]
3. the search tree constructed by the GRAPH-SEARCH algorithm contains **at most one copy** of each state, so we can think of it as growing a tree directly on the state-space graph. [8]
4. The explored set can be implemented with a **hash table** to allow efficient checking for repeated states. [8]

21.3. Search Strategies/ Search Algorithms

21.3.1. Uninformed Search/ Blind Search

1. strategies have **no** additional information about states beyond that provided in the problem definition. [8]
2. All they can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the order in which nodes are expanded. [8]

SEE:

- (22.2) Breadth-first search (BFS) [8]
- (22.3) Uniform-cost search (UCS) [8]
- (22.4) Depth-first search (DFS) [8]
- (22.5) Backtracking Search [8]
- (22.6) Depth-limited search (DLS) [8]
- (22.7) Iterative Deepening Search (IDS) [8]
- (22.8) Iterative Lengthening Search (ILS) [8]
- (22.9) Bidirectional search [8]

21.3.2. Informed Search/ Heuristic Search

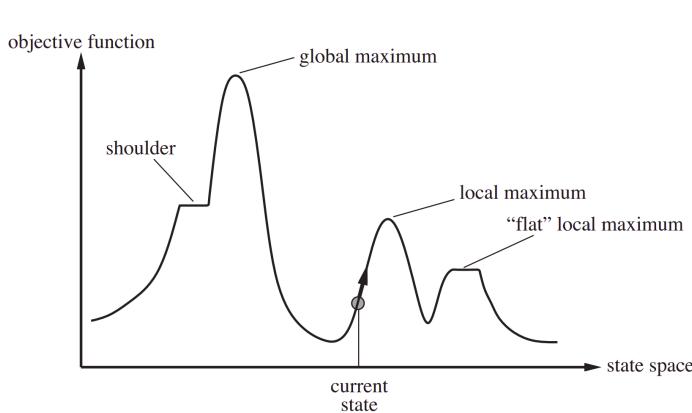
1. Strategies that know whether one non-goal state is “*more promising*” than another [8]
2. uses problem-specific knowledge beyond the definition of the problem itself - can find solutions more efficiently than can an uninformed strategy. [8]

SEE:

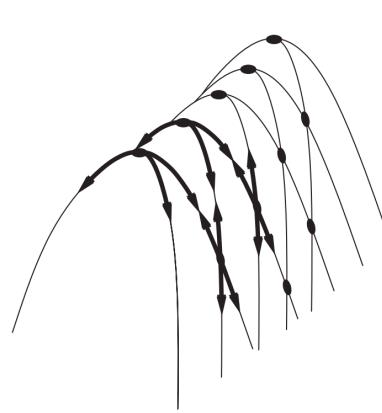
- (22.10) Best-first search (BestFS) [8]
- (22.11) Greedy best-first search (GBFS) [8]
- (22.12) A* Search [8]

- (22.13) Iterative-Deepening A* (IDA*) search [8]
- (22.14) Recursive best-first search (RBFS) [8]
- (22.15) Memory-Bounded A* (MA*) Search [8]
- (22.16) Simplified Memory-Bounded A* (SMA*) Search [8]

21.3.3. Local Search



A one-dimensional **state-space landscape** in which elevation corresponds to the objective function. The aim is to find the global maximum. [8]



The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available actions point downhill. [8]

State-space landscape:

1. A landscape has both “location” (defined by the state) and “elevation” (defined by the value of the heuristic cost function or objective function). [8]
2. if elevation corresponds to cost, then the aim is to find the lowest valley (**global minimum**) [8]
3. if elevation corresponds to an objective function, then the aim is to find the highest peak (**global maximum**) [8]
4. These can convert from one to the other just by inserting a minus sign. [8]
5. **Local maxima:** a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. [8]
6. **Ridges:** Ridges result in a sequence of local maxima [8]
7. **Plateaux:** a plateau is a flat area of the state-space landscape. It can be a **flat local maximum**, from which no uphill exit exists, or a **shoulder**, from which progress is possible. [8]

Local Search:

8. evaluates and modifies one or more current states rather than systematically exploring paths from an initial state. [8]
9. These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to reach it. these do not worry about paths at all. [8]

10. The family of local search algorithms includes methods inspired by statistical physics (simulated annealing) and evolutionary biology (genetic algorithms). [8]
11. Local search algorithms operate using a **single current node** (rather than multiple paths) and generally move only to neighbors of that node. Typically, the paths followed by the search are **not retained**. [8]
12. Local search algorithms are useful for solving **pure optimization problems**, in which the aim is to find the best state according to an **objective function**. [8]
13. Many optimization problems do not fit the “standard” search models. **For example:** nature provides an objective function—reproductive fitness—that Darwinian evolution could be seen as attempting to optimize, but there is no “goal test” and no “path cost” for this problem. [8]
14. Local search algorithms explore this state-space landscape. [8]
15. A complete local search algorithm always finds a goal if one exists. [8]
16. An optimal algorithm always finds a global minimum/ maximum. [8]
17. Local search algorithms typically use a **complete-state formulation**. [8]
18. **Advantages:**
 - (a) they use very little memory—usually a constant amount [8]
 - (b) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable [8]

21.3.4. Online Search

1. agent is faced with a state space that is initially unknown and must be explored. [8]

CHAPTER 22

AI: ALGORITHMS

22.1. Intro

V	set	set of vertices (nodes) of the graph
E	set	set of edges (links) of the graph
b	$\in \mathbb{R}$	branching factor or maximum number of successors of any node
d	$\in \mathbb{R}$	depth of the shallowest goal node (i.e., the number of steps along the path from the root)
m	$\in \mathbb{R}$	maximum length of any path in the state space
ϵ	$\in \mathbb{R}$	minimum step cost (small positive constant)
C	$\in \mathbb{R}$	cost of the solution
C^*	$\in \mathbb{R}$	cost of the optimal solution
ℓ	$\in \mathbb{R}$	predetermined depth limit
G_n	node	goal node closest to n
M	$\in \mathbb{R}$	memory bound
N	$\in \mathbb{R}$	total number of nodes generated
k	$\in \mathbb{R}$	1. number of restarts (for (22.20) Random-restart hill climbing search [8]) 2. number of beams (for (22.22) Local beam search [8])
p	$\in \mathbb{R}$	population size (for (22.24) Genetic algorithm (GA) [8])
f	$\in \mathbb{R}$	time to evaluate the fitness function (for (22.24) Genetic algorithm (GA) [8])
$g(n)$	$\in \mathbb{R}$	Path Cost (The actual cost from the start node to the current node n) (Eg: UCS)
$h(n)$	$\in \mathbb{R}$	Heuristic Estimate (The estimated cost from node n to the goal) (Eg: GBFS, A*)
$f(n)$	$\in \mathbb{R}$	Evaluation Function (The total estimated cost of the cheapest solution through n) (Eg: A*)
$h^*(n)$	$\in \mathbb{R}$	actual cost of getting from the root to the goal
Δ	$\in \mathbb{R}$	absolute error: $\Delta \equiv h^* - h$
ϵ OR Δ_r	$\in \mathbb{R}$	relative error: $\epsilon \equiv \Delta_r \equiv (h^* - h)/h^*$
b^{Δ_r} OR b^ϵ OR b^*	$\in \mathbb{R}$	effective branching factor

1. heuristic function

- (a) $h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state [8]
- (b) Heuristic functions are the most common form in which additional knowledge of the problem is

- imparted to the search algorithm. [8]
- (c) if n is a goal node, then $h(n) = 0$ [8]
- (d) it depends only on the **state** at that node. [8]
- (e) the values of heuristic (eg: h_{SLD}) **may not** be computed from the problem description itself. Moreover, it takes a certain amount of experience to know that h_{SLD} is correlated with actual road distances and is, therefore, a useful heuristic. [8]
- (f) With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic. [8]
- (g) **admissible heuristic:** An admissible heuristic is one that **never overestimates** the cost to reach the goal. Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is. [8]
- (h) **consistency/monotonicity:** A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a , the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n : [8]

$$h(n) \leq c(n, a, n') + h(n')$$
 [8]
every consistent heuristic is also admissible [8]
- (i) Consistency is a stricter requirement than admissibility, but one has to work quite hard to concoct heuristics that are admissible but not consistent. [8]
- (j) if $h(n)$ is consistent, then the values of $f(n)$ along any path are non-decreasing [8]
- (k) For almost all heuristics in practical use, the absolute error is at least proportional to the path cost h^* , so ϵ is constant or growing and the time complexity is exponential in d . [8]
- (l) When the state space has many goal states - particularly **near-optimal goal** states - the search process can be led astray from the optimal path and there is an extra cost proportional to the number of goals whose cost is within a factor of the optimal cost. [8]
- (m) experimental measurements of b^* on a small set of problems can provide a good guide to the heuristic's overall usefulness. A well-designed heuristic would have a value of b^* close to 1, allowing fairly large problems to be solved at reasonable computational cost. [8]
- (n) **Dominating heuristic:** if for any node n , $h_2(n) \geq h_1(n)$, h_2 **dominates** h_1 . Domination translates directly into efficiency: algorithm using h_2 will **never** expand more nodes than same algorithm using h_1 (except possibly for some nodes with $f(n) = C^*$). [8]
For A* search, every node with $f(n) < C^*$ will surely be expanded. This is the same as saying that every node with $h(n) < C^* - g(n)$ will surely be expanded. [8]
- (o) One problem with generating new heuristic functions is that one often fails to get a single "clearly best" heuristic. If a collection of admissible heuristics h_1, \dots, h_m is available for a problem and none of them dominates any of the others, we need not make a choice: [8]

$$h(n) = \max \{h_1(n), \dots, h_m(n)\}$$
 [8]
This **composite heuristic** uses whichever function is most accurate on the node in question. Because the component heuristics are admissible, h is admissible; it is also easy to prove that h is consistent. Furthermore, h dominates all of its component heuristics. [8]
- (p) Note that a perfect heuristic can be obtained simply by allowing h to run a full breadth-first search "on the sly". Thus, there is a tradeoff between accuracy and computation time for heuristic functions. [8]
- (q) Admissible heuristics can also be derived from the solution cost of a **subproblem** of a given problem. [8]
- (r) The idea behind **pattern databases** is to store these exact solution costs for every possible subproblem instance. We compute an admissible heuristic h_{DB} for each complete state encountered during a search simply by looking up the corresponding subproblem configuration in the database.

The database itself is constructed by searching back from the goal and recording the cost of each new pattern encountered; the expense of this search is amortized over many subsequent problem instances. Each database yields an admissible heuristic, and these heuristics can be combined by taking the maximum value. [8]

(s) [8]

2. **evaluation function:**

(a) The evaluation function f is construed/ interpreted as a cost estimate, so the node with the lowest evaluation is expanded first. [8]

(b) The choice of f determines the search strategy. [8]

(c) $f(n) = g(n)$ or $h(n)$ or $g(n) + h(n)$ or something else depending on the algorithm

(d) The fact that f -costs are *non-decreasing* along any path also means that we can draw **contours** in the state space, just like the contours in a topographic map. With more accurate heuristics, the bands will stretch toward the goal state and become more narrowly focused around the optimal path. [8]

(e) There can be exponentially many states with $f(n) < C^*$ even if the absolute error is bounded by a constant. [8]

3. memory limitations can make a problem intractable from the point of view of computation time [8]

4. The effective branching factor can vary across problem instances, but usually it is fairly constant for sufficiently hard problems. [8]

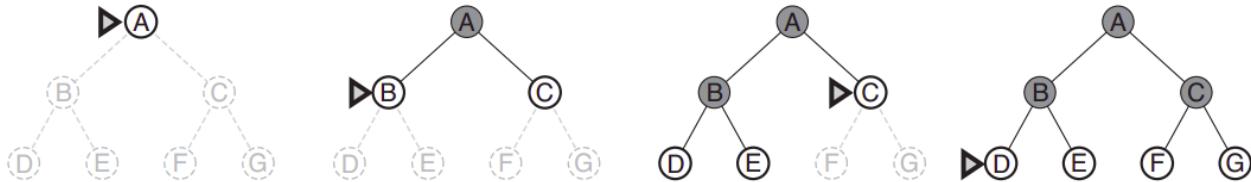
Classical Search Algorithms

1. **category of problems:** observable, deterministic, known environments where the solution is a sequence of actions [8]

2. These search algorithms are designed to explore search spaces **systematically**. This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path. When a goal is found, the path to that goal also constitutes a solution to the problem. In many problems, however, the path to the goal is irrelevant. [8]

Uninformed Search/ Blind Search

22.2. Breadth-first search (BFS) [8]



Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker. [8]

1. Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. [8]
2. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. [8]
3. Breadth-first search is an instance of the general graph-search algorithm in which the *shallowest unexpanded node* is chosen for expansion. [8]
4. new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first. [8]
5. There is one slight tweak on the general graph-search algorithm, which is that the goal test is applied to each node when it is generated rather than when it is selected for expansion. [8]
If the algorithm were to apply the goal test to nodes when selected for expansion, rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected and the time complexity would be $\mathcal{O}(b^{d+1})$. [8]
6. the algorithm, following the general template for graph search, discards any new path to a state already in the frontier or explored set; it is easy to see that any such path must be at least as deep as the one already found. [8]
7. breadth-first search **always** has the *shallowest path* to every node on the frontier. As soon as a goal node is generated, we know it is the shallowest goal node because all shallower nodes must have been generated already and failed the goal test. [8]
8. **performance:**
 - (a) **complete:** if the shallowest goal node is at some finite depth d , breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite). [8]
 - (b) the shallowest goal node is **not necessarily** the *optimal* one. breadth-first search is optimal if the path cost is a non-decreasing function of the depth of the node. The most common such scenario is that all actions have the same cost. the algorithm is optimal if step costs are all identical. [8]
 - (c) **Space Complexity:**
 - i. explored set: $\mathcal{O}(b^{d-1})$ [8]
 - ii. frontier: $\mathcal{O}(b^d)$ [8]
 - iii. overall: $\mathcal{O}(b^d)$ [8]
 - (d) **Time Complexity:** $\mathcal{O}(b^d)$ [8]
9. **Disadvantages:**

- (a) the memory requirements are a bigger problem for breadth-first search than is the execution time.
[8]

Implementation

1. frontier: FIFO queue

Algorithm 22.1: Breadth-first search on a graph. [8]

```

1 function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
2   node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
3   if problem.GOAL-TEST(node.STATE) then return SOLUTION(node);
4   frontier  $\leftarrow$  a FIFO queue with node as the only element
5   explored  $\leftarrow$  an empty set
6
7   while do
8     if EMPTY?(frontier) then return failure ;
      /* chooses the shallowest node in frontier
9     node  $\leftarrow$  POP(frontier)
10    add node.STATE to explored
11
12    foreach action in problem.ACTIONS(node.STATE) do
13      child  $\leftarrow$  CHILD-NODE(problem, node, action)
14      if child.STATE is not in explored or frontier then
15        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child) ;
16        frontier  $\leftarrow$  INSERT(child, frontier)

```

```

1 from queue import Queue
2
3 def breadth_first_search(problem: Problem):
4   node = Node(problem.initial_state, None, None, 0)
5
6   if problem.goal_test(node.state):
7     return solution(node)
8
9   frontier = Queue()
10  explored = set()
11
12  frontier.put(node)
13
14  while True:
15    if frontier.empty():
16      return None
17
18    node = frontier.get()
19    explored.add(node)
20
21    for action in problem.actions(node.state):
22      child = child_node(problem, node, action)
23
24      if (not any([n.state == child.state for n in frontier.queue]) and
25          not any([n.state == child.state for n in explored])):
26        if problem.goal_test(child.state):

```

```

27     return solution(child)
28
29     frontier.put(child)

```

Python Snippet 22.1: Problem Solving Agent - Breadth-first search on a graph

22.3. Uniform-cost search (UCS) [8]

- Instead of expanding the shallowest node (as in BFS), uniform-cost search expands the node n with the *lowest path cost* $g(n)$. [8]

2. Performance:

- (a) uniform-cost search is **optimal** in general. uniform-cost search expands nodes in order of their optimal path cost. [8]
- (b) **Completeness is guaranteed** provided the cost of every step exceeds some small positive constant ϵ and b is finite [8]
- (c) **Space Complexity:**
 - i. worst: $\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$ (can be much greater than b^d) [8]
 - ii. When all step costs are equal: $\mathcal{O}(b^{d+1})$ [8]
- (d) **Time Complexity:**
 - i. worst: $\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$ (can be much greater than b^d) [8]
 - ii. When all step costs are equal: $\mathcal{O}(b^{d+1})$ [8]

3. Disadvantages:

- (a) it will get stuck in an **infinite loop** if there is a path with an infinite sequence of *zero-cost actions* - for example, a sequence of *NoOp* actions. [8]
- (b) it suffers from the same difficulties with realvalued costs [8]
- 4. When all step costs are the same, uniform-cost search is similar to breadth-first search, except that the latter stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost; thus uniform-cost search does strictly more work by expanding nodes at depth d unnecessarily. [8]

Implementation

- This is done by storing the frontier as a **priority queue** ordered by $g(n)$. [8]
- In addition to the ordering of the queue by path cost, there are two other significant differences from breadth-first search. [8]
 - (a) goal test is applied to a node when it is *selected for expansion* rather than when it is first generated. The reason is that the first goal node that is *generated* may be on a suboptimal path. [8]
 - (b) a test is added in case a better path is found to a node currently on the frontier. [8]

Algorithm 22.2: UNIFORM-COST search on a graph. [8]

```

1 function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
2   node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
3   frontier  $\leftarrow$  a priority queue ordered by PATH-COST, with node as the only element
4   explored  $\leftarrow$  an empty set
5
6   while do
7     if EMPTY?(frontier) then return failure ;
8       /* chooses the lowest-cost node in frontier
9       node  $\leftarrow$  POP(frontier)
10      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node) ;
11      add node.STATE to explored
12
13      foreach action in problem.ACTIONS(node.STATE) do
14        child  $\leftarrow$  CHILD-NODE(problem, node, action)
15        if child.STATE is not in explored or frontier then
16          frontier  $\leftarrow$  INSERT(child, frontier)
17        else if child.STATE is in frontier with higher PATH-COST then
18          replace that frontier node with child
```

```

1 import heapq
2
3 def uniform_cost_search(problem: Problem):
4     node = Node(problem.initial_state, None, None, 0)
5
6     if problem.goal_test(node.state):
7         return solution(node)
8
9     frontier = [(node.path_cost, node)]
10    explored = set()
11
12    heapq.heapify(frontier)
13
14    while True:
15        if len(frontier) == 0:
16            return None
17
18        path_cost, node = frontier.pop(0)
19
20        if problem.goal_test(node.state):
21            return solution(node)
22
23        explored.add(node)
24        for action in problem.actions(node.state):
25            child = child_node(problem, node, action)
26
27            if (not any([n.state == child.state for (path_cost, n) in frontier])
28                and not any([n.state == child.state for n in explored])):
29                frontier.append((child.path_cost, child))
30                heapq.heapify(frontier)
31            else:
32                for idx, (path_cost, n) in enumerate(frontier):
```

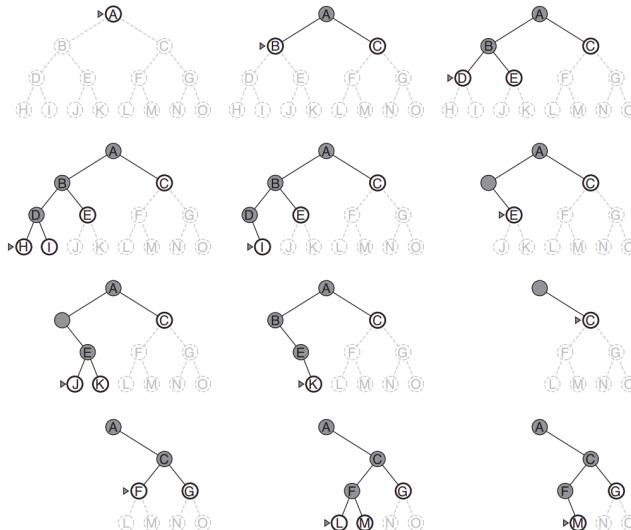
```

33         if n.state == child.state and path_cost > child.path_cost:
34             frontier[idx] = (child.path_cost, child)
35             heapq.heapify(frontier)

```

Python Snippet 22.2: Problem Solving Agent - Uniform cost search on a graph

22.4. Depth-first search (DFS) [8]



Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and *M* is the only goal node. [8]

1. Depth-first search always expands the **deepest node** in the current frontier of the search tree. [8]
2. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors. [8]

3. Performance:

(a) graph-search version:

- i. **complete** in finite state spaces because it will eventually expand every node [8]
- ii. **not optimal** [8]
- iii. **time complexity:** $\mathcal{O}(b^d)$ [8]
- iv. **space complexity:** $\mathcal{O}(b^d)$ [8]

(b) tree-search version:

- i. **not complete**, as it can fall in *infinite loop* [8]
- ii. **not optimal** [8]
- iii. **time complexity:** $\mathcal{O}(b^m)$ [8]
- iv. **space complexity:** $\mathcal{O}(bm)$ [8]

Implementation

1. The depth-first search algorithm is an instance of the graph-search algorithm that uses a LIFO queue. [8]

2. it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn. [8]

```

1 from queue import LifoQueue
2
3 def depth_first_search(problem: Problem):
4     node = Node(problem.initial_state, None, None, 0)
5
6     if problem.goal_test(node.state):
7         return solution(node)
8
9     frontier = LifoQueue()
10    explored = set()
11
12    frontier.put(node)
13
14    while True:
15        if frontier.empty():
16            return None
17
18        node = frontier.get()
19        explored.add(node)
20
21        for action in problem.actions(node.state):
22            child = child_node(problem, node, action)
23
24            if (not any([n.state == child.state for n in frontier.queue]) and
25                not any([n.state == child.state for n in explored])):
26                if problem.goal_test(child.state):
27                    return solution(child)
28
29            frontier.put(child)

```

Python Snippet 22.3: Problem Solving Agent - Depth first search on a graph

22.5. Backtracking Search [8]

1. A variant of depth-first search that uses still less memory. [8]
2. **only one successor** is generated at a time rather than all successors; each partially expanded node remembers which successor to generate next. [8]

3. Performance:

- (a) **Space Complexity:** $\mathcal{O}(m)$ [8]

4. Advantages:

- (a) Backtracking search facilitates memory-saving (and time-saving) trick: the idea of generating a successor by **modifying** the current state description directly rather than copying it first. For this to work, we must be able to **undo** each modification when we go back to generate the next successor. [8]

- (b) For problems with large state descriptions, such as robotic assembly, these techniques are critical to success. [8]

```

1 def backtrack(node: Node, explored: set):
2     if problem.goal_test(node.state):
3         return solution(node)
4
5     explored.add(node)
6
7     for action in problem.actions(node.state):
8         child = child_node(problem, node, action)
9         if not any([child.state == n.state for n in explored]):
10            result = backtrack(child, explored)
11            if result is not None:
12                return result
13
14    # Optional: allow revisiting for other paths (depends on problem)
15    explored.remove(node)
16    return None
17
18 def backtracking_search(problem: Problem):
19     root = Node(problem.initial_state, None, None, 0)
20     return backtrack(root, set())

```

Python Snippet 22.4: Problem Solving Agent - Backtracking using recursion [7]

22.6. Depth-limited search (DLS) [8]

1. The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit ℓ . That is, nodes at depth ℓ are treated as if they have no successors. [8]

2. Performance:

- (a) **Completeness:** NO if $\ell < d$ else YES [8]
 - (b) **Optimal:** NO if $\ell > d$ else YES [8]
 - (c) **time complexity:** $\mathcal{O}(b^\ell)$ [8]
 - (d) **space complexity:** $\mathcal{O}(b\ell)$ [8]
3. Depth-first search can be viewed as a special case of depth-limited search with $\ell = \infty$. [8]
 4. diameter of the state space can be a better limit for efficiency [8]

Algorithm 22.3: A recursive implementation of depth-limited tree search. [8]

```

1 function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
2   return RECURSIVE-DLS(
3     MAKE-NODE(problem.INITIAL-STATE),
4     problem,
5     limit,
6   )
7
8 function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
9   if problem.GOAL-TEST(node.STATE) then return SOLUTION(node);
10  else if limit = 0 then return cutoff ;
11  else
12    cutoff_occurred?  $\leftarrow$  false
13
14    foreach action in problem.ACTIONS(node.STATE) do
15      child  $\leftarrow$  CHILD-NODE(problem, node, action)
16      result  $\leftarrow$  RECURSIVE-DLS(child, problem, limit - 1)
17      if result = cutoff then cutoff_occurred?  $\leftarrow$  true ;
18      else if result  $\neq$  failure then return result ;
19
20    if cutoff_occurred? then return cutoff ;
21    else return failure ;

```

```

1 CUTOFF = "CUT-OFF"
2
3 def depth_limited_search(problem: Problem, limit: int):
4   return recursive_dls(
5     Node(problem.initial_state, None, None, 0),
6     problem,
7     limit,
8   )
9
10 def recursive_dls(node: Node, problem: Problem, limit: int):
11   if problem.goal_test(node.state):
12     return solution(node)
13
14   elif limit == 0:
15     return CUTOFF
16
17   else:
18     cutoff_occurred = False
19     for action in problem.actions(node.state):
20       child = child_node(problem, node, action)
21       result = recursive_dls(child, problem, limit-1)
22       if result == CUTOFF:
23         cutoff_occurred = True
24       elif result is not None:
25         return result
26     if cutoff_occurred:
27       return CUTOFF
28     else:

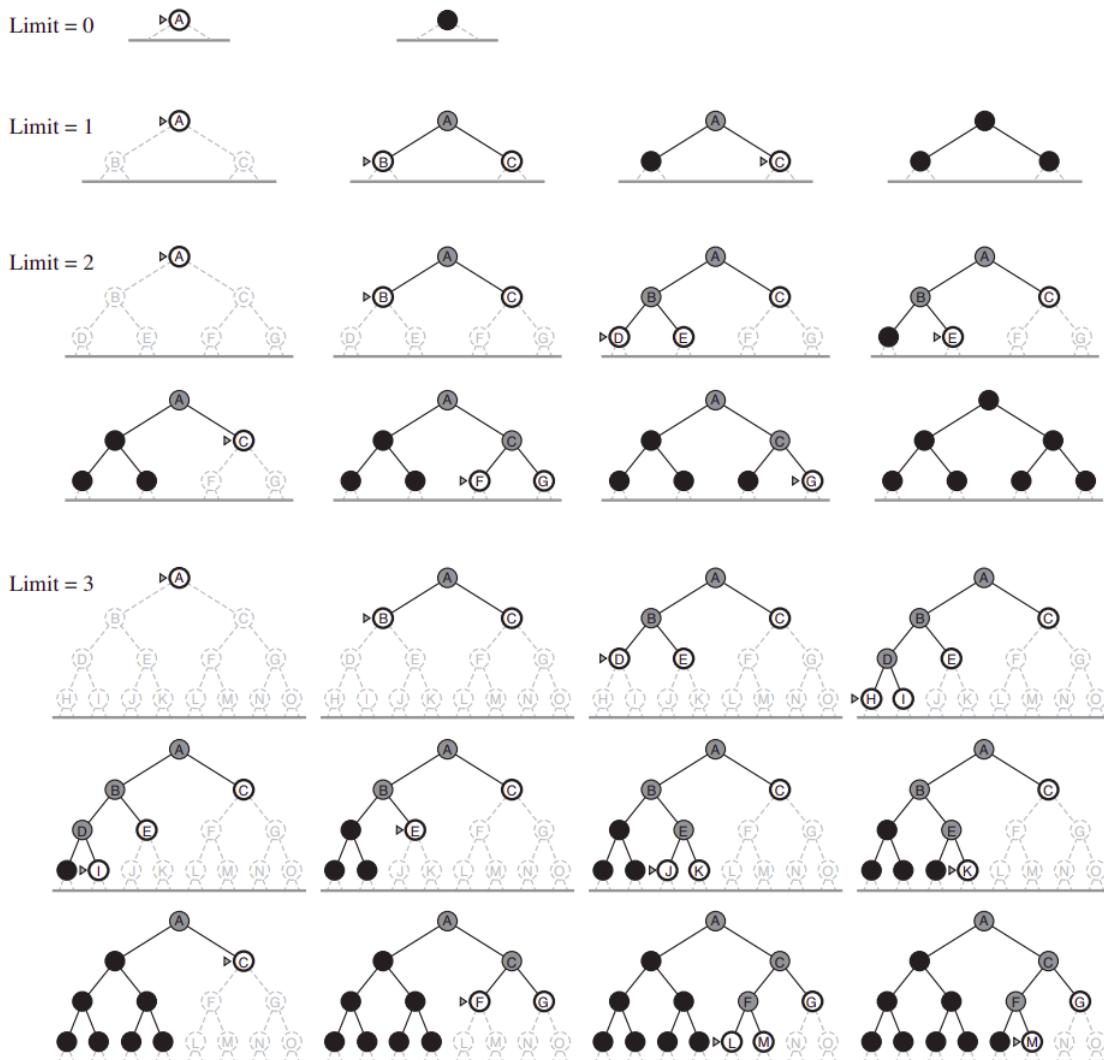
```

```
return
```

Python Snippet 22.5: Problem Solving Agent - Depth limited search (recursive)

1. *failure* value indicates no solution [8]
2. *cutoff* value indicates no solution within the depth limit [8]

22.7. Iterative Deepening Search (IDS) [8]



Four iterations of iterative deepening search on a binary tree [8]

1. **Iterative deepening search (or iterative deepening depth-first search)** is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit. [8]
2. It gradually increases the depth limit $\ell \in \{0, 1, 2, \dots, \infty\}$ till a goal is found, which occurs when $\ell = d$ (d is unknown in reality) [8]
3. Iterative deepening combines the benefits of depth-first and breadth-first search.
 - (a) Like depth-first search, its memory requirements are modest: $\mathcal{O}(bd)$ to be precise. [8]
 - (b) Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a non-decreasing function of the depth of the node. [8]

4. total number of nodes generated in the worst case is
 $N(\text{IDS}) = (d)b + (d-1)b^2 + \dots + (1)b^d$ [8]
5. you can use a hybrid approach that runs breadth-first search until almost all the available memory is consumed, and then runs iterative deepening from all the nodes in the frontier. [8]
6. iterative deepening is the *preferred uninformed search* method when the search space is large and the depth of the solution is not known. [8]

7. Performance:

- (a) **completeness**: YES if b is finite else NO [8]
- (b) **optimal**: YES if path cost is a non-decreasing function of depth of the node else NO [8]
- (c) **space complexity**: $\mathcal{O}(bd)$ [8]
- (d) **time complexity**: $\mathcal{O}(b^d)$ [8]

Algorithm 22.4: The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns failure, meaning that no solution exists. [8]

```

1 function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
2   for depth = 0 to  $\infty$  do
3     result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
4     if result  $\neq$  cutoff then return result ;

```

```

1 def iterative_deepening_search(problem: Problem, limit=1000000):
2   for i in range(limit):
3     result = depth_limited_search(problem, i)
4     if result != CUTOFF:
5       return result

```

Python Snippet 22.6: Problem Solving Agent - Iterative Deepening Search

22.8. Iterative Lengthening Search (ILS) [8]

1. an iterative analog to uniform-cost search, inheriting the Iterative deepening search algorithm's optimality guarantees while avoiding its memory requirements. The idea is to use increasing path-cost limits instead of increasing depth limits. [8]
2. **Disadvantage:** It turns out that iterative lengthening incurs substantial overhead compared to uniform-cost search. [8]

```

1 def iterative_lengthening_search(problem: Problem):
2   cost_limit = 0
3
4   while True:
5     result, new_cost_limit = uniform_cost_search_with_cost_limit(
6       problem,
7       cost_limit,
8     )
9
10    if result is not None:

```

```

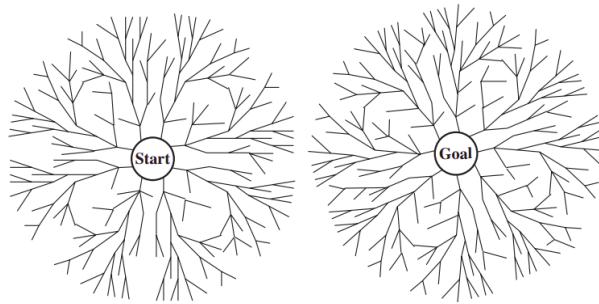
11         return result
12
13     if new_cost_limit == float('inf'):
14         return None
15
16     cost_limit = new_cost_limit
17
18
19 def uniform_cost_search_with_cost_limit(problem: Problem, cost_limit: int):
20     node = Node(problem.initial_state, None, None, 0)
21
22     if problem.goal_test(node.state):
23         return solution(node), cost_limit
24
25     frontier = [(node.path_cost, node)]
26     explored = set()
27     heapq.heapify(frontier)
28     next_cost_limit = float('inf')
29
30     while frontier:
31         path_cost, node = heapq.heappop(frontier)
32
33         if path_cost > cost_limit:
34             next_cost_limit = min(next_cost_limit, path_cost)
35             continue
36
37         if problem.goal_test(node.state):
38             return solution(node), cost_limit
39
40         explored.add(node)
41
42         for action in problem.actions(node.state):
43             child = child_node(problem, node, action)
44
45             if (not any(child.state == n.state for n in explored) and
46                 not any(n.state == child.state for _, n in frontier)):
47                 heapq.heappush(frontier, (child.path_cost, child))
48
49     return None, next_cost_limit

```

Python Snippet 22.7: Problem Solving Agent - Iterative Lengthening Search [7]

22.9. Bidirectional search [8]

1. The idea behind bidirectional search is to run two simultaneous searches - one forward from the initial state and the other backward from the goal - hoping that the two searches meet in the middle [8]
2. The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d [8]
3. Bidirectional search is implemented by replacing the goal test with a check to see whether the frontiers of the two searches intersect; if they do, a solution has been found. [8]
4. The check can be done when each node is generated or selected for expansion and, with a hash table, will take constant time. [8]



A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node. [8]

5. **predecessors** of a state x be all those states that have x as a successor. Bidirectional search requires a method for computing predecessors. When all the actions in the state space are reversible, the predecessors of x are just its successors. [8]

6. **Performance:**

(a) **optimality:**

- i. YES if additional search is used to make sure the path isn't another short-cut across the gap
else NO [8]
- ii. YES if step costs are all identical **and** both directions use breadth-first search **else** NO [8]

(b) **completeness:** YES if b is finite **and** both directions use breadth-first search **else** NO [8]

(c) **space complexity:**

- i. using breadth-first searches in both directions: $\mathcal{O}(b^{d/2})$ [8]

(d) **time complexity:**

- i. using breadth-first searches in both directions: $\mathcal{O}(b^{d/2})$ [8]

Informed Search/ Heuristic Search

22.10. Best-first search (BestFS) [8]

1. Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function**, $f(n)$. $f(n)$ can be any function that returns a value for a given node. [8]
2. The implementation of best-first graph search is identical to that for uniform-cost search, except for the use of f instead of g to order the priority queue. [8]
3. Most best-first algorithms include h as a component of f [8]

```

1 def best_first_search(problem: Problem, f):
2     node = Node(problem.initial_state, None, None, 0)
3
4     if problem.goal_test(node.state):
5         return solution(node)
6
7     frontier = [(f(node), node)]
8     heapq.heapify(frontier)
9     explored = set()
10
11    while frontier:
12        _, node = heapq.heappop(frontier)
13
14        if problem.goal_test(node.state):
15            return solution(node)
16
17        explored.add(node)
18
19        for action in problem.actions(node.state):
20            child = child_node(problem, node, action)
21            if (all(n.state != child.state for n in explored)
22                and all(n.state != child.state for _, n in frontier)):
23                heapq.heappush(frontier, (f(child), child))
24            else:
25                # Optional: Replace in frontier if better
26                for i, (old_f, old_node) in enumerate(frontier):
27                    if old_node.state == child.state and f(child) < old_f:
28                        frontier[i] = (f(child), child)
29                        heapq.heapify(frontier)
30                        break
31
32    return None

```

Python Snippet 22.8: Problem Solving Agent - (General) Best-first search (BestFS) on a graph

22.11. Greedy best-first search (GBFS) [8]

$$f(n) = h(n)$$

[8]

1. Greedy best-first search tries to expand the node that is **closest** to the goal, on the grounds that this is likely to lead to a solution quickly. [8]
 2. at each step it tries to get as close to the goal as it can [8]
3. **Performance:**
- (a) **Completeness:** YES if graph version is used **and** states are finite **else** NO [8]
 - (b) **time complexity**
 - i. worst case: $\mathcal{O}(b^m)$ [8]
 - (c) **space complexity**
 - i. worst case: $\mathcal{O}(b^m)$ [8]

```
1 import heapq
2
3 def greedy_best_first_search(problem: Problem):
4     node = Node(problem.initial_state, None, None, 0)
5
6     if problem.goal_test(node.state):
7         return solution(node)
8
9     # Priority queue ordered by heuristic cost
10    frontier = [(problem.heuristic(node.state), node)]
11    heapq.heapify(frontier)
12
13    explored = set()
14
15    while len(frontier) > 0:
16        _, node = heapq.heappop(frontier)
17
18        if problem.goal_test(node.state):
19            return solution(node)
20
21        explored.add(node)
22
23        for action in problem.actions(node.state):
24            child = child_node(problem, node, action)
25            h_cost = problem.heuristic(child.state)
26
27            # Add child only if it's not in frontier or explored
28            in_frontier = any(n.state == child.state for _, n in frontier)
29            in_explored = any(n.state == child.state for n in explored)
30
31            if not in_frontier and not in_explored:
32                heapq.heappush(frontier, (h_cost, child))
33            else:
34                # Optional: If it's in the frontier but has a
35                # better heuristic, replace it
36                for idx, (old_cost, old_node) in enumerate(frontier):
37                    if old_node.state == child.state and h_cost < old_cost:
38                        frontier[idx] = (h_cost, child)
```

```

39             heapq.heapify(frontier)
40             break
41
42     return None # No solution found
43
44 # Greedy Best-First Search using general best first search
45 greedy_best_first_search_alt = lambda problem: best_first_search(
46     problem,
47     f=lambda n: problem.heuristic(n.state)
48 )

```

Python Snippet 22.9: Problem Solving Agent - Greedy Best-first search (GBFS) on a graph

22.12. A* Search [8]

$$f(n) = g(n) + h(n) \quad [8]$$

1. It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal ($f(n)$ = estimated cost of the cheapest solution through n) [8]
2. A* selects a node n for expansion, the optimal path to that node has been found [8]
3. Because A* expands the frontier node of lowest f -cost, we can see that an A* search fans out from the start node, adding nodes in **concentric bands** of increasing f -cost. [8]
4. If C^* is the cost of the optimal solution path, then we can say the following:
 - (a) A* expands all nodes with $f(n) < C^*$ [8]
 - (b) A* might then expand some of the nodes right on the “goal contour” (where $f(n) = C^*$) before selecting a goal node. [8]
5. if $h(n)$ is admissible, the algorithm can safely **ignore/ prune** this sub-tree while still guaranteeing optimality [8]
6. among optimal algorithms of this type - algorithms that extend search paths from the root and use the same heuristic information - A* is **optimally efficient** for any given consistent heuristic. [8]
7. best results follow the 3 assumptions:
 - (a) Single goal state [8]
 - (b) Tree structure [8]
 - (c) Reversible actions [8]
8. **Performance:**
 - (a) the *tree-search* version of A* is **optimal** if $h(n)$ is admissible, while the *graph-search* version is **optimal** if $h(n)$ is consistent [8]
 - (b) **Completeness** requires that there be only finitely many nodes with cost less than or equal to C^* , a condition that is true if all step costs exceed some finite and if b is finite. [8]
 - (c) **time complexity:**
 - i. in maximum absolute error: $\mathcal{O}(b^\Delta)$ [8]
 - ii. constant step costs: $\mathcal{O}(b^{\Delta_r d})$ or $\mathcal{O}(b^{ed})$ [8]
9. **Disadvantages**

- (a) for most problems, the number of states within the goal contour search space is still **exponential** in the length of the solution [8]
- (b) Because it keeps all generated nodes in memory (as do all GRAPH-SEARCH algorithms), A* usually runs out of space long before it runs out of time. [8]
- (c) A* is not practical for many large-scale problems. [8]
- (d) The existence of an effective branching factor follows from the result that the number of nodes expanded by A* grows exponentially with solution depth. [8]

```

1 def a_star_search(problem: Problem):
2     start_node = Node(problem.initial_state, None, None, 0)
3
4     if problem.goal_test(start_node.state):
5         return solution(start_node)
6
7     # Priority queue ordered by f(n) = g(n) + h(n)
8     frontier = [(problem.heuristic(start_node.state), start_node)]
9     heapq.heapify(frontier)
10
11    explored = dict() # {state: path_cost}
12
13    while len(frontier) > 0:
14        f, current = heapq.heappop(frontier)
15
16        if problem.goal_test(current.state):
17            return solution(current)
18
19        # If this state has already been explored with a lower cost, skip
20        if any(n.state == current.state for n in explored) and explored[current] <= current.path_cost:
21            continue
22
23        explored[current] = current.path_cost
24
25        for action in problem.actions(current.state):
26            child = child_node(problem, current, action)
27            f_child = child.path_cost + problem.heuristic(child.state)
28
29            # Check if child.state was already explored with a lower path_cost
30            if (any(n.state == child.state for n in explored) or
31                explored[child] > child.path_cost):
32                heapq.heappush(frontier, (f_child, child))
33
34    return None # No solution found
35
36 # A* Search using general best first search
37 a_star_search_alt = lambda problem: best_first_search(
38     problem,
39     f=lambda n: n.path_cost + problem.heuristic(n.state)
40 )

```

Python Snippet 22.10: Problem Solving Agent - A* search [7]

22.13. Iterative-Deepening A* (IDA*) search [8]

1. adapt the idea of iterative deepening to the heuristic search context [8]
2. The main difference between IDA* and standard iterative deepening is that the cutoff used is the f -cost ($g + h$) rather than the depth [8]
3. at each iteration, the cutoff value is the smallest f -cost of any node that exceeded the cutoff on the previous iteration [8]
4. IDA* is practical for many problems with unit step costs and avoids the substantial overhead associated with keeping a sorted queue of nodes. [8]
5. **Disadvantages:**
 - (a) it suffers from the same difficulties with real-valued costs [8]
 - (b) suffer from using too little memory. [8]
 - (c) may end up re-expanding the same states many times over [8]
 - (d) suffer the potentially exponential increase in complexity associated with redundant paths in graphs [8]

```

1 def ida_star_search(problem: Problem):
2     """
3         Performs Iterative Deepening A* Search.
4         Returns the solution as a list of actions, or None if no solution is found.
5     """
6
7     start_node = Node(problem.initial_state, None, None, 0)
8     bound = problem.heuristic(start_node.state)
9
10    while True:
11        result = _ida_search(start_node, problem, bound)
12
13        if isinstance(result, list): # Found a solution
14            return result
15
16        if result == float('inf'):
17            return None # No solution
18
19        bound = result # Increase bound and try again
20
21
22 def _ida_search(node: Node, problem: Problem, bound: float):
23     """
24         Helper function for IDA* search.
25         Returns either:
26             - A solution path (list of actions), or
27             - The next bound (float) to use in the next iteration
28     """
29     f = node.path_cost + problem.heuristic(node.state)
30
31     if f > bound:
32         return f
33
34     if problem.goal_test(node.state):

```

```

35     return solution(node)
36
37 min_threshold = float('inf')
38 for action in problem.actions(node.state):
39     child = child_node(problem, node, action)
40     result = _ida_search(child, problem, bound)
41
42     if isinstance(result, list):
43         return result # Solution found
44
45     if result < min_threshold:
46         min_threshold = result
47
48 return min_threshold

```

Python Snippet 22.11: Problem Solving Agent - Iterative-Deepening A* (IDA*)

22.14. Recursive best-first search (RBFS) [8]

1. simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only **linear space** [8]
2. Its structure is similar to that of a recursive depth-first search, but rather than continuing indefinitely down the current path, it uses the *f_limit* variable to keep track of the *f*-value of the best **alternative** path available from any ancestor of the current node. If the current node exceeds this limit, the recursion unwinds back to the alternative path. [8]
3. As the recursion unwinds, RBFS replaces the f-value of each node along the path with a **backed-up value** - the best f-value of its children. In this way, RBFS remembers the *f*-value of the best leaf in the forgotten sub-tree and can therefore decide whether it's worth re-expanding the sub-tree at some later time. [8]
4. RBFS is somewhat more **efficient** than IDA* [8]
5. it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded. [8]
6. **Disadvantages:**
 - (a) suffers from excessive node regeneration [8]
 - (b) Each mind change corresponds to an iteration of IDA* and could require many re-expansions of forgotten nodes to recreate the best path and extend it one more node. [8]
 - (c) suffer from using too little memory. [8]
 - (d) suffer the potentially exponential increase in complexity associated with redundant paths in graphs [8]

Algorithm 22.5: The algorithm for recursive best-first search. [8]

```

1 function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
2   return RBFS(
3     problem,
4     MAKE-NODE(problem.INITIAL-STATE),
5      $\infty$ ,
6   )
7
8 function RBFS(problem) returns a solution, or failure and a new f-cost limit
9   if problem.GOAL-TEST(node.STATE) then return SOLUTION(node);
10  successors  $\leftarrow []$ 
11  foreach action in problem.ACTIONS(node.STATE) do
12    add CHILD-NODE(problem, node, action) into successors
13  if successors is empty then return failure,  $\infty$ ;
14  /* update f with value from previous search, if any */
15  foreach s in successors do s.f  $\leftarrow \max(s.g + s.h, node.f)$ ;
16
17  while do
18    best  $\leftarrow$  the lowest f-value node in successors
19    if best.f  $> f\_limit$  then return failure, best.f;
20    alternative  $\leftarrow$  the second-lowest f-value among successors
21    result, best.f  $\leftarrow$  RBFS(problem, best, min(f_limit, alternative))
      if result  $\neq$  failure then return result ;

```

22.15. Memory-Bounded A* (MA*) Search [8]

1.

[8]

22.16. Simplified Memory-Bounded A* (SMA*) Search [8]

1. SMA* proceeds just like A*, expanding the best leaf until memory is full. At this point, it cannot add a new node to the search tree without dropping an old one. SMA* always drops the **worst leaf node**—the one with the highest *f*-value. [8]
2. Like RBFS, SMA* then backs up the value of the forgotten node to its parent. In this way, the ancestor of a forgotten sub-tree knows the quality of the best path in that sub-tree. SMA* regenerates the sub-tree only when all other paths have been shown to look worse than the path it has forgotten. [8]
3. if all the descendants of a node *n* are forgotten, then we will not know which way to go from *n*, but we will still have an idea of how worthwhile it is to go anywhere from *n*. [8]
4. if all the leaf nodes have the same *f*-value, SMA* expands the **newest best** leaf and deletes the **oldest worst** leaf to avoid selecting the same node for deletion and expansion [8]
5. If the leaf is not a goal node, then even if it is on an optimal solution path, that solution is not reachable with the available memory. Therefore, the node can be discarded exactly as if it had no successors. [8]
6. In practical terms, SMA* is a fairly robust choice for finding optimal solutions, particularly when the state space is a graph, step costs are not uniform, and node generation is expensive compared to the overhead of maintaining the frontier and the explored set. [8]
7. **Disadvantages:**

- (a) On very hard problems, it will often be the case that SMA* is forced to switch back and forth continually among many candidate solution paths, only a small subset of which can fit in memory. [8]
- (b) the extra time required for repeated regeneration of the same nodes means that problems that would be practically solvable by A*, given unlimited memory, become intractable for SMA*. [8]

Local Search

22.17. (Greedy) Hill Climbing Search [8]

1. It is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a “peak” where no neighbor has a higher value. [8]
2. The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function. [8]
3. Hill climbing does not look ahead beyond the immediate neighbors of the current state. This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia. [8]
4. Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next. [8]

5. Disadvantages:

- (a) Hill-climbing algorithms that reach the vicinity of a **local maximum** will be drawn upward toward the peak but will then be stuck with nowhere else to go. [8]
- (b) very difficult for greedy algorithms to navigate **ridges**. [8]
- (c) A hill-climbing search might get lost on the **plateau** [8]

Algorithm 22.6: The hill-climbing search algorithm (**steepest-ascent** version), which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h . [8]

```

1 function HILL-CLIMBING-SEARCH(problem) returns a state that is a local maximum
2   current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
3
4   while do
5     neighbor  $\leftarrow$  a highest-valued successor of current
6     if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE ;
7     current  $\leftarrow$  neighbor

```

22.18. Stochastic hill climbing search [8]

1. Stochastic hill climbing chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. [8]
2. This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions. [8]

22.19. First-choice hill climbing search [8]

1. First-choice hill climbing implements stochastic hill climbing by generating successors **randomly** until one is generated that is better than the current state. [8]
2. This is a good strategy when a state has many (e.g., thousands) of successors. [8]

22.20. Random-restart hill climbing search [8]

1. Random-restart hill climbing adopts the well-known adage, “*If at first you don’t succeed, try, try again*”. [8]
2. It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found. [8]
3. It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state. [8]
4. If each hill-climbing search has a probability p of success, then the expected number of restarts required is $1/p$. [8]
5. The expected number of steps is the cost of one successful iteration plus $(1 - p)/p$ times the cost of failure. [8]

22.21. Simulated annealing search [8]

1. In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state. [8]
2. The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and then gradually reduce the intensity of the shaking (i.e., lower the temperature). [8]

Algorithm 22.7: The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time. [8]

```

/* schedule is a mapping from time to “temperature” */ 
1 function SIMULATED-ANNEALING(problem, schedule) returns a solution state
2   current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
3   for t = 1 to  $\infty$  do
4     T  $\leftarrow$  schedule(t)
5     if T = 0 then return current;
6     next  $\leftarrow$  a randomly selected successor of current
7      $\Delta E \leftarrow$  next.VALUE - current.VALUE
8     if  $\Delta E > 0$  then current  $\leftarrow$  next ;
9     else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$  ;

```

1. The innermost loop of the simulated-annealing algorithm is quite similar to hill climbing. Instead of picking the **best** move, however, it picks a **random** move. If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1. [8]
2. The probability decreases exponentially with the “badness” of the move—the amount ΔE by which the evaluation is worsened. [8]
3. The probability also decreases as the “temperature” T goes down: “bad” moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases. [8]
4. If the *schedule* lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1. [8]

22.22. Local beam search [8]

1. The local beam search algorithm keeps track of k states rather than just one. [8]
2. It begins with k randomly generated states. At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the k best successors from the complete list and repeats. The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made. [8]
3. In a local beam search, useful information is passed among the parallel search threads. [8]
4. **Disadvantages:**
 - (a) local beam search can suffer from a lack of diversity among the k states—they can quickly become concentrated in a small region of the state space, making the search little more than an expensive version of hill climbing. [8]

22.23. Stochastic beam search [8]

1. Instead of choosing the best k from the pool of candidate successors, stochastic beam search chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value. [8]
2. Stochastic beam search bears some resemblance to the process of natural selection, whereby the “successors” (offspring) of a “state” (organism) populate the next generation according to its “value” (fitness). [8]

22.24. Genetic algorithm (GA) [8]

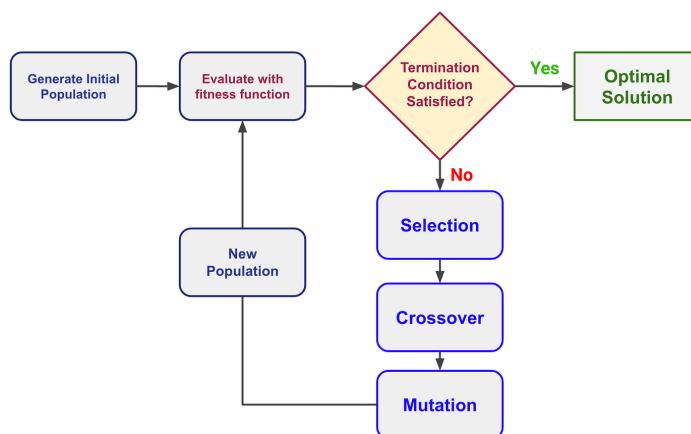


Fig. 22.1: Genetic Algorithm - flowchart

1. A genetic algorithm (GA) is a variant of stochastic beam search in which successor states are generated by combining *two* parent states rather than by modifying a single state. [8]
2. Like beam searches, GAs begin with a set of k randomly generated states, called the **population**. Each state, or **individual**, is represented as a string over a finite alphabet—most commonly, a string of 0s and 1s. [8]
3. **culling:** all individuals below a given threshold are discarded, can be shown to converge faster than the random version [8]
4. **steps:**

(a) population states [8]

(b) each state is rated by the objective function, or (in GA terminology) the **fitness function**. A fitness function should return higher values for better states [8]

(c) two pairs are selected at **random** for **reproduction**, in accordance with the probabilities in (b). For each pair to be mated, a **crossover point** is chosen randomly from the positions in the string. [8]

(d) the offspring themselves are created by crossing over the parent strings at the crossover point. When two parent states are quite different, the crossover operation can produce a state that is a long way from either parent state. It is often the case that the population is quite diverse early on in the process, so crossover (like simulated annealing) frequently takes large steps in the state space early in the search process and smaller steps later on when most individuals are quite similar [8]

(e) each location (gene) is subject to random **mutation** with a small independent probability. [8]

5. Like stochastic beam search, genetic algorithms combine an uphill tendency with random exploration and exchange of information among parallel search threads. [8]

6. The primary advantage, if any, of genetic algorithms comes from the crossover operation. Yet it can be shown mathematically that, if the positions of the genetic code are permuted initially in a random order, crossover conveys no advantage. [8]

7. **Schema:** Schema is a pattern or template where some positions are fixed (specified) and others are left open (unspecified). It's like a rule that only certain spots matter, and the rest can be anything. For example, in the pattern "246****", the first three positions (2, 4, 6) are fixed, and the last four can be any number or value. Genetic algorithms work best when schemata correspond to meaningful components of a solution. [7, 8]

8. **Instance:** An instance is a specific example that fits the schema. It follows the pattern by having the fixed parts the same as the schema but fills in the unspecified parts with actual values. For example, "24613578" is an instance of the schema "246****" because it has 2, 4, and 6 in the first three positions, and the rest are any values. [7, 8]

9. If the average fitness of the instances of a schema is above the mean, then the number of instances of the schema within the population will grow over time. This effect is unlikely to be significant if adjacent bits are totally unrelated to each other, because then there will be few contiguous blocks that provide a consistent benefit. [8]

Algorithm 22.8: A genetic algorithm. In this more popular version, each mating of two parents produces only one offspring, not two. [8]

```

1 function REPRODUCE(x,y) returns an individual
  Input: x,y: parent individuals
2   n  $\leftarrow$  LENGTH(x)
3   c  $\leftarrow$  random number from 1 to n
4   return APPEND(
5     SUBSTRING(x, 1, c),
6     SUBSTRING(y, c + 1, n)
7   )
8 function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  Input: population: a set of individuals,
          FITNESS-FN: a function that measures the fitness of an individual
9  repeat
10    new_population  $\leftarrow$   $\emptyset$ 
11    for i  $\leftarrow$  1 to SIZE(population) do
12      x  $\leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)
13      y  $\leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)
14      child  $\leftarrow$  Reproduce(x,y)
15      if (small random probability) then
16        child  $\leftarrow$  MUTATE(child)
17        add child to new_population
18    population  $\leftarrow$  new_population
19  until some individual is fit enough, or enough time has elapsed;
20  return the best individual in population, according to FITNESS-FN

```

22.25. Local Search in Continuous Space

1. One way to avoid continuous problems is simply to discretize the neighborhood of each state. [8]
2. Many methods attempt to use the gradient of the landscape to find a maximum. The gradient of the objective function is a vector ∇f that gives the magnitude and direction of the steepest slope. [8]
3. In some cases, we can find a maximum by solving the equation $\nabla f = 0$. [8]
4. In many cases, however, this equation cannot be solved in closed form. Given a locally correct expression for the gradient, we can perform **steepest-ascent hill climbing** by updating the current state according to the formula $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$, where α is a small constant often called the step size. [8]
 - (a) The basic problem is that, if α is too small, too many steps are needed; if α is too large, the search could overshoot the maximum. [8]
 - (b) The technique of **line search** tries to overcome this dilemma by extending the current gradient direction—usually by repeatedly doubling α —until f starts to decrease again. The point at which this occurs becomes the new current state. [8]
5. In other cases, the objective function might not be available in a differentiable form at all. In those cases, we can calculate a so-called **empirical gradient** by evaluating the response to small increments and decrements in each coordinate. Empirical gradient search is the same as **steepest-ascent hill climbing** in a **discretized** version of the state space. [8]
6. For many problems, the most effective algorithm is the venerable **Newton–Raphson method**. This is a general technique for finding roots of functions—that is, solving equations of the form $g(x) = 0$. It works by computing a new estimate for the root x according to Newton's formula $x \leftarrow x - \frac{g(x)}{g'(x)}$. To find a maximum or minimum of f , we need to find \mathbf{x} such that the gradient is zero (i.e., $\nabla f(\mathbf{x}) = \mathbf{0}$). Thus, $g(x)$ in Newton's formula becomes $\nabla f(\mathbf{x})$, and the update equation can be written in matrix–vector form as $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$, where $\mathbf{H}_f(\mathbf{x})$ is the **Hessian matrix** of second derivatives, whose elements $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. [8]

22.26. Searching with Non-deterministic Actions

1. When the environment is either partially observable or nondeterministic (or both), percepts become useful. [8]
 - (a) In a partially observable environment, every percept helps narrow down the set of possible states the agent might be in, thus making it easier for the agent to achieve its goals. [8]
 - (b) When the environment is nondeterministic, percepts tell the agent which of the possible outcomes of its actions has actually occurred. [8]

In both cases, the future percepts cannot be determined in advance and the agent's future actions will depend on those future percepts. So the solution to a problem is not a sequence but a **contingency plan** (also known as a **strategy**) that specifies what to do depending on what percepts are received. [8]

22.27. Summary

Algorithm	Complete?	Optimal?	Time Complexity		Space Complexity	
			Graph	Tree	Graph	Tree
UNINFORMED SEARCH/ BLIND SEARCH						
Breadth-first search (BFS)	YES ³	YES ^{1 2}	$\mathcal{O}(b^d)$		$\mathcal{O}(b^d)$	
Uniform-cost search (UCS)	YES ⁴	YES	$\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$		$\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$	
Depth-first search (DFS)	YES ^{3&5}	NO	$\mathcal{O}(b^d)$	$\mathcal{O}(b^m)$	$\mathcal{O}(bd)$	$\mathcal{O}(bm)$
Backtracking Search	YES ^{3&5}	NO	$\mathcal{O}(b^d)$	$\mathcal{O}(b^m)$	$\mathcal{O}(d)$	$\mathcal{O}(m)$
Depth-limited search (DLS)	NO ⁶	YES ⁶	$\mathcal{O}(b^\ell)$		$\mathcal{O}(b\ell)$	
Iterative Deepening Search (IDS)	YES ³	YES ^{1 2}	$\mathcal{O}(b^d)$		$\mathcal{O}(bd)$	
Iterative Lengthening Search (ILS)	YES ⁴	YES	$\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$		$\mathcal{O}(bC^*/\varepsilon)$	
Bidirectional search	YES ⁷	YES ^{2&8}	$\mathcal{O}(b^{d/2})$		$\mathcal{O}(b^{d/2})$	
INFORMED SEARCH/ HEURISTIC SEARCH						
Greedy best-first search (GBFS)	YES ^{3&5}	NO	$\mathcal{O}(b^m)$		$\mathcal{O}(b^m)$	
A* Search	YES ^{3&4}	YES ^{(9&10) (5&11)}	$\mathcal{O}(b^\Delta)^{12}$ or $\mathcal{O}(b^{\Delta_r d})^2$		$\mathcal{O}(b^d)$	
Iterative-Deepening A* (IDA*) search	YES ⁴	YES ¹⁰	$\mathcal{O}(b^d)$		$\mathcal{O}(d)$	
Recursive best-first search (RBFS)	YES ⁴	YES ¹⁰	$\mathcal{O}(b^d)$		$\mathcal{O}(bd)$	
Memory-Bounded A* (MA*) Search	YES ¹³	NO	$\mathcal{O}(b^d)$		$\mathcal{O}(M)$	
Simplified Memory-Bounded A* (SMA*) Search	YES ¹⁴	YES ¹⁵	$\mathcal{O}(b^d)$		$\mathcal{O}(M)$	
LOCAL SEARCH						
(Greedy) Hill Climbing Search	NO	NO	$\mathcal{O}(b^m)$		$\mathcal{O}(m)$	
Stochastic hill climbing search	NO	NO	$\mathcal{O}(b^m)$		$\mathcal{O}(m)$	
First-choice hill climbing search	NO	NO	$\mathcal{O}(b)$ per move		$\mathcal{O}(m)$	

Algorithm	Complete?	Optimal?	Time Complexity		Space Complexity	
			Graph	Tree	Graph	Tree
Random-restart hill climbing search	YES ¹⁶	YES ¹⁷	$\mathcal{O}(kb^m)$		$\mathcal{O}(m)$	
Simulated annealing search	YES ¹⁸	YES ¹⁸	$\mathcal{O}(b^m)$		$\mathcal{O}(m)$	
Local beam search	NO	NO	$\mathcal{O}(kb)$		$\mathcal{O}(k)$	
Stochastic beam search	NO	NO	$\mathcal{O}(kb)$		$\mathcal{O}(k)$	
Genetic algorithm (GA)	NO	NO	$\mathcal{O}(pbf)$		$\mathcal{O}(p)$	

Conditions:

- | | |
|--|---|
| 1. path cost is a non-decreasing function of the depth of the node
2. step costs are all identical
3. branching factor b is finite (number of states are finite)
4. every step exceeds some small positive constant $\epsilon > 0$
5. graph version is used
6. $\ell < d$
7. additional search is used to make sure the path isn't another short-cut across the gap
8. both directions use breadth-first search | 9. tree version is used
10. $h(n)$ is admissible
11. $h(n)$ is consistent
12. in the maximum absolute error
13. memory bound is sufficient
14. depth of the shallowest goal node, is less than the memory size (expressed in nodes)
15. optimal solution is reachable
16. given infinite restarts
17. run indefinitely
18. cooling schedule is slow enough |
|--|---|

Notation Interpretation:

- | | |
|---|--|
| 1. YES ^{a b} = YES if either a or b must satisfy else NO | 2. YES ^{a&b} = YES if either a and b must satisfy else NO |
|---|--|

CHAPTER 23

AI: EXAMPLES

23.1. Vacuum Cleaner (Toy problem) [8]

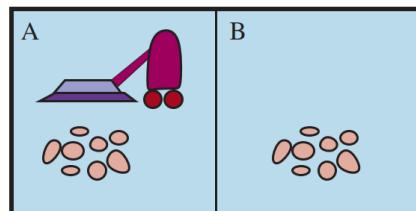


Fig. 23.1: A vacuum-cleaner world with just two locations

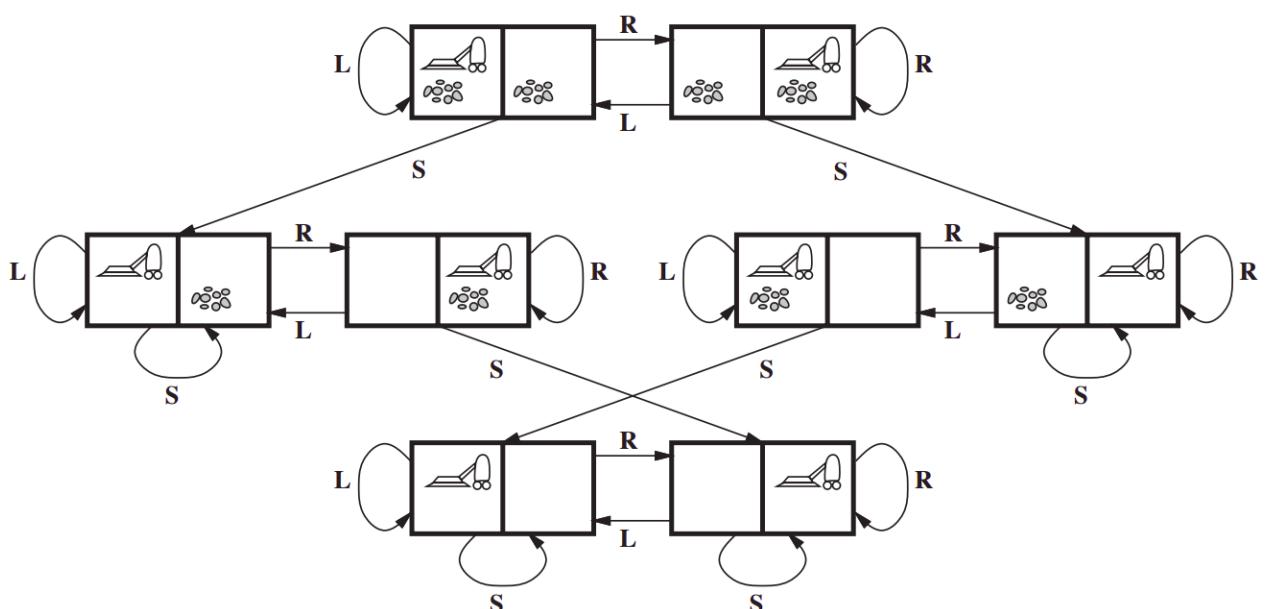


Fig. 23.2: The state space for the vacuum world. Links denote actions: L = Left, R = Right, S= Suck. [8]

1. it's a made-up world
2. This particular world has just two locations: squares A and B.
3. The vacuum agent perceives which square it is in and whether there is dirt in the square.
4. It can choose to **move left, move right, suck up the dirt, or do nothing**.
5. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.

23.1.1. As a table driven agent

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
:	:
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
:	:

23.1.2. As a simple reflex agent

Algorithm 23.1: The agent program for a simple reflex agent in the two-state vacuum environment.

[8]

```

1 function REFLEX-VACUUM-AGENT([location, status]) returns an action
2   if status = Dirty then return Suck
3   else if location = A then return Right
4   else if location = B then return Left

```

23.1.3. As a problem solving agent

1. **States:** The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
2. **Initial state:** Any state can be designated as the initial state.
3. **Actions:** In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
4. **Transition model:** The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.
5. **Goal test:** This checks whether all the squares are clean.
6. **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

Defining Problem & State

```

1 vacuumActions = ["L", "R", "S"] # L: Left, R: Right, S: Suck
2 vacuumSquareState = ["D", "C"] # D: Dirty, C: Clean
3
4 class VacuumState(State):
5   def __init__(self, squareA, squareB, currLoc):
6     self.squares = {}
7     self.squares["A"] = squareA
8     self.squares["B"] = squareB
9     self.currLoc = currLoc

```

```
10
11     def __str__(self):
12         return (
13             "<VacuumState " +
14             f"squares={self.squares} " +
15             f"currLoc={self.currLoc}>" +
16         )
17
18     def __repr__(self) -> str:
19         return str(self)
20
21     def __eq__(self, __o: object) -> bool:
22         return str(self) == str(__o)
23
24     def copy(self):
25         return VacuumState(
26             self.squares["A"],
27             self.squares["B"],
28             self.currLoc,
29         )
30
31 class VacuumProblem(Problem):
32     def __init__(self, initial_state: VacuumState):
33         self.initial_state = initial_state
34
35     def step_cost(self, state: VacuumState, action: str, new_state: VacuumState):
36         return (int(state.squares[state.currLoc] == "D" and action == "S") +
37                 int(state.currLoc == "A" and action == "R") +
38                 int(state.currLoc == "B" and action == "L"))
39
40     def result(self, state: VacuumState, action: str):
41         state = state.copy()
42
43         state.currLoc = {
44             "L": "A",
45             "R": "B",
46         }.get(action, state.currLoc)
47         state.squares[state.currLoc] = {
48             "S": "C"
49         }.get(action, state.squares[state.currLoc])
50
51         return state
52
53     def actions(self, state: VacuumState):
54         actions = []
55
56         if state.squares[state.currLoc] == "D":
57             actions.append("S")
58
59         if state.currLoc == "A":
60             actions.append("R")
61
62         if state.currLoc == "B":
63             actions.append("L")
```

```

63
64     return actions
65
66     def goal_test(self, state: VacuumState):
67         return all([v == "C" for v in state.squares.values()])
68
69 init_state = VacuumState("D", "D", "A")

```

Solution using breadth_first_search

```

1 problem = VacuumProblem(init_state)
2 path = breadth_first_search(problem)
3 print(path)
4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node

```

Solution using uniform_cost_search

```

1 problem = VacuumProblem(init_state)
2 path = uniform_cost_search(problem)
3 print(path)
4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node

```

Solution using depth_first_search

```

1 problem = VacuumProblem(init_state)
2 path = depth_first_search(problem)
3 print(path)
4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node

```

Solution using backtracking_search

```

1 problem = VacuumProblem(init_state)
2 path = backtracking_search(problem)
3 print(path)
4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node

```

Solution using depth_limited_search

```
1 problem = VacuumProblem(init_state)
2 path = depth_limited_search(problem, 5)
3 print(path)

4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node

11
12 print("\n\n")
13
14 problem = VacuumProblem(init_state)
15 path = depth_limited_search(problem, 2)
16 print(path)

17
18 if isinstance(path, list):
19     node = Node(init_state, None, None, 0)
20     print(node)
21     for a in path:
22         new_node = child_node(problem, node, a)
23         print(a, new_node.state, new_node.path_cost, new_node)
24         node = new_node
```

Solution using iterative_deepening_search

```
1 problem = VacuumProblem(init_state)
2 path = iterative_deepening_search(problem)
3 print(path)

4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node
```

Solution using iterative_lengthening_search

```
1 problem = VacuumProblem(init_state)
2 path = iterative_lengthening_search(problem)
3 print(path)

4
5 node = Node(init_state, None, None, 0)
6 print(node)
7 for a in path:
8     new_node = child_node(problem, node, a)
9     print(a, new_node.state, new_node.path_cost, new_node)
10    node = new_node
```

23.2. Automated Taxi Driver [8]

1. **Performance Measure:** Safe, fast, legal, comfortable trip, maximize profits [8]
2. **Environment:** Roads, other traffic, pedestrians, customers [8]
3. **Actuators:** Steering, accelerator, brake, signal, horn, display [8]
4. **Sensors:** Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard [8]
5. an automated taxi cannot see what other drivers are thinking. [8]
6. partially cooperative multiagent environment: avoiding collisions maximizes the performance measure of all agents [8]
7. **Task Environment:** partially observable, multiagent, stochastic, sequential, dynamic, continuous, and unknown [8]

23.2.1. As a table driven reflex agent

1. the visual input from a single camera comes in at the rate of roughly 27 megabytes per second (30 frames per second, 640×480 pixels with 24-bits of color information). [8]
2. This gives a lookup table with over $10^{250,000,000,000}$ entries for **an hour's** driving. [8]
3. **Not possible** to construct the table [8]

23.2.2. As model-based reflex agents

1. the taxi may be driving back home, and it may have a rule telling it to fill up with gas on the way home unless it has at least half a tank. [8]
2. Although “driving back home” may seem to be an aspect of the world state, the fact of the taxi’s destination is actually an aspect of the agent’s internal state. [8]
3. If you find this puzzling, consider that the taxi could be in exactly the same place at the same time, but intending to reach a different destination. [8]

23.2.3. As utility-based agents

1. **utility measures:** quicker, safer, more reliable, or cheaper [8]

23.2.4. As Learning Agent

1. The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions. [8]
2. after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by other drivers. [8]
3. From this experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installation of the new rule. [8]

23.3. Road trip in Romania [8]

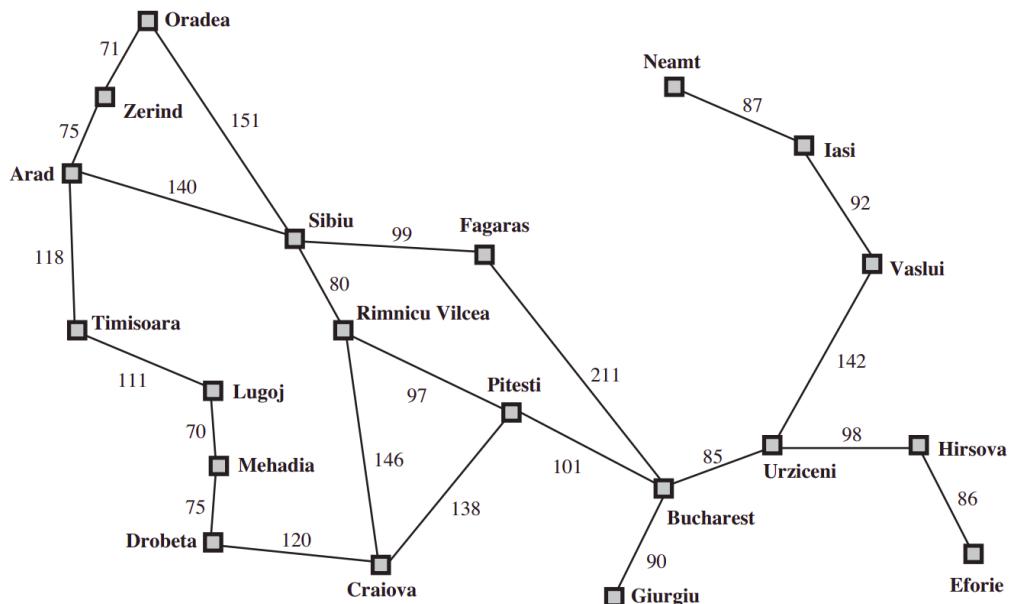


Fig. 23.3: A simplified road map of part of Romania. [8]

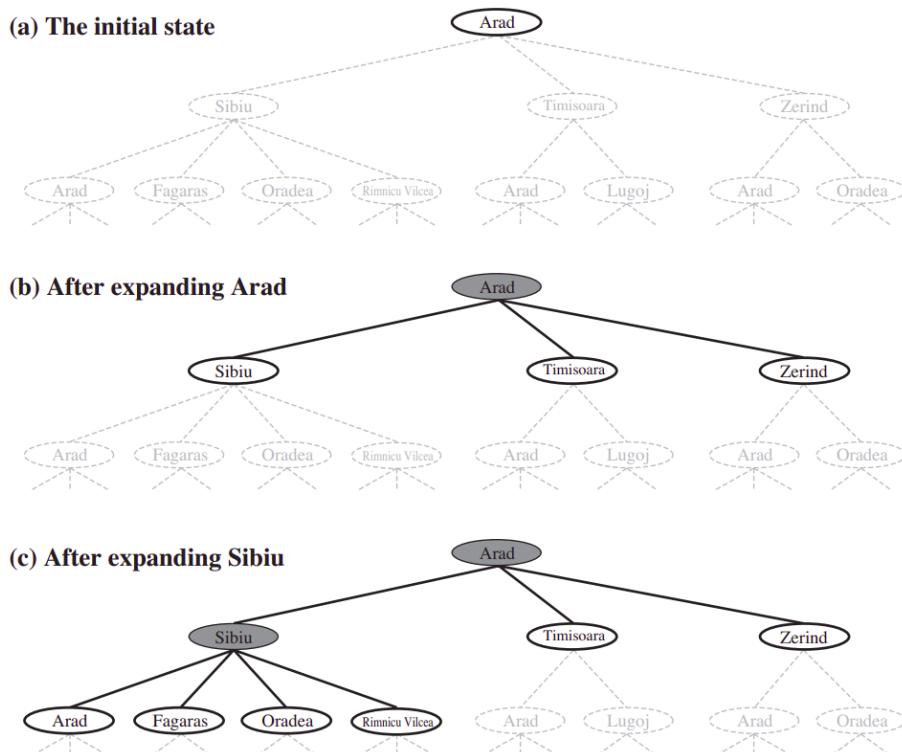


Fig. 23.4: Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines. [8]

1. we use the straight-line distance (SLD) heuristic

[8]

Defining Problem & State

```
1 # src: {tgt: path_cost}
```

```

2 romania_map_graph = {
3     "Arad": {"Timisoara": 118, "Sibiu": 140, "Zerind": 75},
4     "Bucharest": {"Pitesti": 101, "Fagaras": 211, "Giurgiu": 90, "Urziceni": 85},
5     "Craiova": {"Drobeta": 120, "Rimnicu Vilcea": 146, "Pitesti": 138},
6     "Drobeta": {"Mehadia": 75, "Craiova": 120},
7     "Eforie": {"Hirsova": 86},
8     "Fagaras": {"Sibiu": 99, "Bucharest": 211},
9     "Giurgiu": {"Bucharest": 90},
10    "Hirsova": {"Urziceni": 98, "Eforie": 86},
11    "Iasi": {"Vaslui": 92, "Neamt": 87},
12    "Lugoj": {"Mehadia": 70, "Timisoara": 111},
13    "Mehadia": {"Lugoj": 70, "Drobeta": 75},
14    "Neamt": {"Iasi": 87},
15    "Oradea": {"Zerind": 71, "Sibiu": 151},
16    "Pitesti": {"Rimnicu Vilcea": 97, "Craiova": 138, "Bucharest": 101},
17    "Rimnicu Vilcea": {"Sibiu": 80, "Pitesti": 97, "Craiova": 146},
18    "Sibiu": {"Fagaras": 99, "Rimnicu Vilcea": 80, "Arad": 140, "Oradea": 151},
19    "Timisoara": {"Arad": 118, "Lugoj": 111},
20    "Urziceni": {"Bucharest": 85, "Vaslui": 142, "Hirsova": 98},
21    "Vaslui": {"Urziceni": 142, "Iasi": 92},
22    "Zerind": {"Arad": 75, "Oradea": 71},
23 }
24
25 heuristic_Bucharest = {
26     "Arad": 366,
27     "Bucharest": 0,
28     "Craiova": 160,
29     "Drobeta": 242,
30     "Eforie": 161,
31     "Fagaras": 176,
32     "Giurgiu": 77,
33     "Hirsova": 151,
34     "Iasi": 226,
35     "Lugoj": 244,
36     "Mehadia": 241,
37     "Neamt": 234,
38     "Oradea": 380,
39     "Pitesti": 100,
40     "Rimnicu Vilcea": 193,
41     "Sibiu": 253,
42     "Timisoara": 329,
43     "Urziceni": 80,
44     "Vaslui": 199,
45     "Zerind": 374,
46 }
47
48
49 class RomaniaState(State):
50     def __init__(self, city: str) -> None:
51         self.city = city
52
53     def copy(self):
54         return RomaniaState(self.city)

```

```

55
56     def __str__(self):
57         return f"<RomaniaState city={self.city}>"
58
59     def __repr__(self):
60         return str(self)
61
62     def __eq__(self, __o):
63         return self.city == __o.city
64
65 class RomaniaProblem(Problem):
66     def __init__(self, initial_state: RomaniaState):
67         super().__init__(initial_state)
68
69     def goal_test(self, state: RomaniaState):
70         return state.city == "Bucharest"
71
72     def step_cost(self, state: RomaniaState, action: str, new_state: RomaniaState):
73         return romania_map_graph[state.city][new_state.city]
74
75     def heuristic(self, state: RomaniaState):
76         return heuristic_Bucharest[state.city]
77
78     def actions(self, state: RomaniaState):
79         return sorted(romania_map_graph[state.city].keys())
80
81     def result(self, state: RomaniaState, action: str):
82         return RomaniaState(action)
83
84
85 initial_state = RomaniaState("Arad")
86 problem = RomaniaProblem(initial_state=initial_state)

```

Solution using breadth_first_search

```

1 path = breadth_first_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13    node = new_node

```

Solution using uniform_cost_search

```

1 path = uniform_cost_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:

```

```

6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13    node = new_node

```

Solution using depth_first_search

```

1 path = depth_first_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13    node = new_node

```

Solution using backtracking_search

```

1 path = backtracking_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13    node = new_node

```

Solution using depth_limited_search

```

1 path = depth_limited_search(problem, 4)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13    node = new_node
14

```

```
15 print("\n")
16
17 path = depth_limited_search(problem, 2)
18 print(path)
19
20 if isinstance(path, list):
21     node = Node(initial_state, None, None, 0)
22     for e in path:
23         new_node = child_node(problem, node, e)
24         print(
25             node.state.city.ljust(20),
26             new_node.state.city.ljust(20),
27             str(new_node.path_cost - node.path_cost).rjust(5),
28             str(new_node.path_cost).rjust(5)
29         )
30         node = new_node
```

Solution using iterative_deepening_search

```
1 path = iterative_deepening_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13     node = new_node
```

Solution using iterative_lengthening_search

```
1 path = iterative_lengthening_search(problem)
2 print(path)
3
4 node = Node(initial_state, None, None, 0)
5 for e in path:
6     new_node = child_node(problem, node, e)
7     print(
8         node.state.city.ljust(20),
9         new_node.state.city.ljust(20),
10        str(new_node.path_cost - node.path_cost).rjust(5),
11        str(new_node.path_cost).rjust(5)
12    )
13     node = new_node
```

Solution using A* search

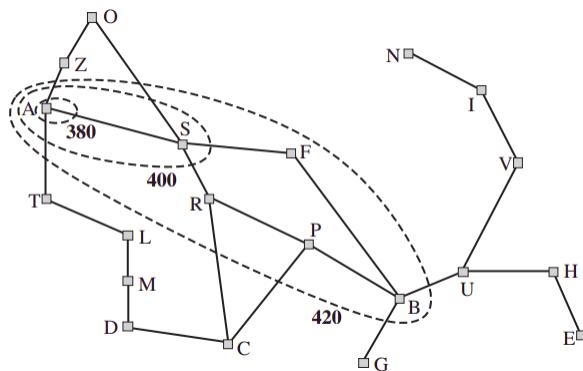
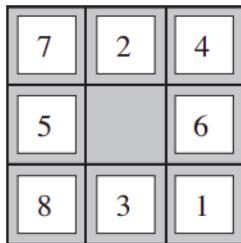
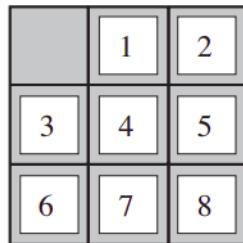


Fig. 23.5: Nodes inside a given contour have f-costs less than or equal to the contour value. [8]

23.4. 8-Puzzle [8]

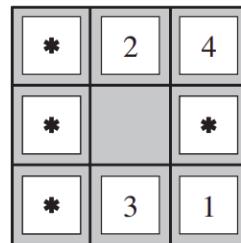


Start State

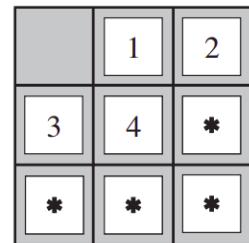


Goal State

Fig. 23.6: A typical instance of the 8-puzzle. The solution is 26 steps long. [8]



Start State



Goal State

Fig. 23.7: A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles. [8]

1. the objective of the puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration [8]
2. The average solution cost for a randomly generated 8-puzzle instance is about 22 steps. The branching factor is about 3. (When the empty tile is in the middle, four moves are possible; when it is in a corner, two; and when it is along an edge, three.) This means that an exhaustive tree search to depth 22 would look at about $3^{22} \approx 3.1 \times 10^{10}$ states. A *graph search* would cut this down by a factor of about 170,000 because only $9!/2 = 181,440$ distinct states are reachable. [8]
3. if the 8-puzzle actions are described as:
 - (a) A tile can move from square A to square B **if** A is horizontally or vertically *adjacent* to B **and** B is *blank* [8]
 - (b) we can generate three relaxed problems by removing one or both of the conditions:
 - i. A tile can move from square A to square B if A is adjacent to B. [8]
 - ii. A tile can move from square A to square B if B is blank. [8]
 - iii. A tile can move from square A to square B. [8]

23.4.1. Comparing heuristic functions

two commonly used candidates:

1. h_1 : the number of misplaced tiles. It is an **admissible** heuristic because it is clear that any tile that is out of place must be moved at least once. [8]
All of the eight tiles are out of position, so the start state would have $h_1 = 8$. [8]
2. h_2 = the sum of the distances of the tiles from their goal positions. (city block distance or Manhattan distance) h_2 is **admissible** because all any move can do is move one tile one step closer to the goal. [8]
 $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$ [8]

23.4.1.1. A* [8]

1. $N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$ [8]
2. if A* finds a solution at depth 5 using 52 nodes, then the effective branching factor is 1.92 [8]

23.4.1.2. Comparison [8]

1. h_2 is always better than h_1 . for any node n , $h_2(n) \geq h_1(n)$. [8]

d	Search Cost (nodes generated)			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Table 23.2: Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A^* algorithms with h_1, h_2 .

Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d . [8]

2. h_1 and h_2 are **estimates** of the remaining path length for the 8-puzzle, but they are also **perfectly accurate** path lengths for **simplified versions** of the puzzle. [8]

23.4.1.3. Generating admissible heuristics from subproblems: Pattern databases

Relaxed Sub-problem method

1. The choice of 1-2-3-4 is fairly arbitrary; we could also construct databases for 5-6-7-8, for 2-4-6-8, and so on. [8]
2. the heuristics obtained from the 1-2-3-4 database and the 5-6-7-8 **cannot** be added because the solutions of the 1-2-3-4 subproblem and the 5-6-7-8 subproblem for a given state will almost certainly share some moves—it is unlikely that 1-2-3-4 can be moved into place without touching 5-6-7-8, and vice versa. [8]

Experience method

3. “Experience” here means solving lots of 8-puzzles. Each optimal solution to an 8-puzzle problem provides examples from which $h(n)$ can be learned. Each example consists of a state from the solution path and the actual cost of the solution from that point. [8]
4. From these examples, a learning algorithm can be used to construct a function $h(n)$ that can (with luck) predict solution costs for other states that arise during search. Techniques for doing this use neural nets, decision trees, and other methods or reinforcement learning. [8]

23.5. 8-Queens problem

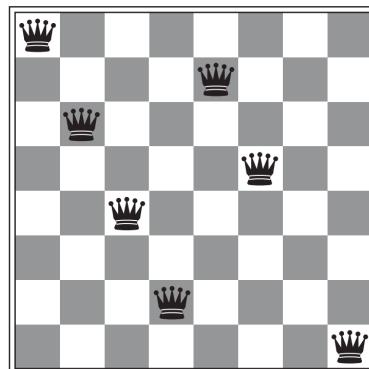


Fig. 23.8: Almost a solution to the 8-queens problem. the queen in the rightmost column is attacked by the queen at the top left.

1. The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) [8]
2. [8]

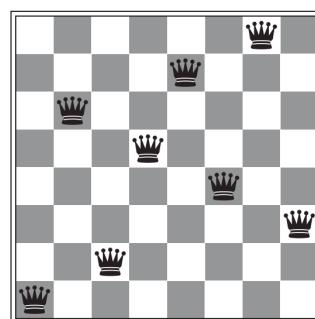
23.5.1. As a Problem Solving Agent

1. **States:** Any arrangement of 0 to 8 queens on the board is a state. [8]
2. **Initial state:** No queens on the board. [8]
3. **Actions:** Add a queen to any empty square. [8]
4. **Transition model:** Returns the board with a queen added to the specified square. [8]
5. **Goal test:** 8 queens are on the board, none attacked. [8]

23.5.2. As a Hill Climbing Agent

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	13	16	13	16
17	14	17	15	15	14	16	16
18	17	16	18	15	15	15	17
18	14	14	15	15	14	14	16
14	14	13	17	12	14	12	18

(a)



(b)

Fig. 23.9: (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. [8]
 (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost. [8]

1. each state has 8 queens on the board, one per column [8]

2. The successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has $8 \times 7 = 56$ successors). [8]
3. The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly. [8]
4. The global minimum of this function is zero, which occurs only at perfect solutions. [8]

23.5.3. Using Genetic Algorithm

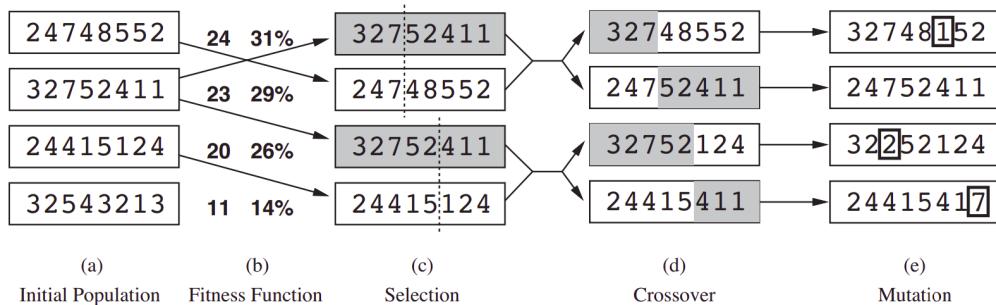


Fig. 23.10: The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e). [8]

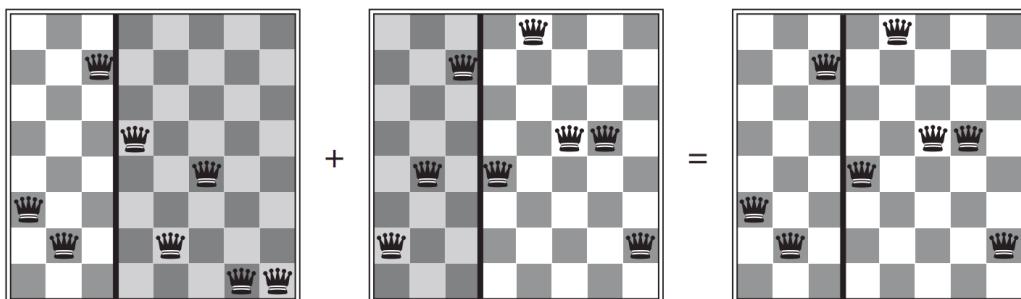


Fig. 23.11: The 8-queens states corresponding to the first two parents (c) and the first offspring (d). The shaded columns are lost in the crossover step and the unshaded columns are retained. [8]

1. an 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so requires $8 \times \log_2 8 = 24$ bits. Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8. [8]

VI

MACHINE LEARNING (ML) & DEEP LEARNING (DL)

CHAPTER 24

★ML PROJECT: CORE COMPONENTS [7]

24.1. Problem Definition

1. **Goal:** What are you trying to solve? (e.g., predict housing prices, detect spam, classify images)
2. **Type:** Supervised, unsupervised, reinforcement learning, etc.

24.2. Data

1. **Raw Data:** Collected from sensors, web scraping, databases, APIs, etc.
2. **Preprocessing:** Cleaning, normalization, handling missing values, feature extraction.
3. **Splits:**
 - (a) Training Set
 - (b) Validation Set
 - (c) Test Set
4. **Data Loaders:** Handle batching, shuffling, and transformations (especially in deep learning frameworks like PyTorch, TensorFlow).

24.3. Model Architecture

1. **Definition:** The structure of your model (e.g., decision tree, CNN, transformer).
2. **Layers:** Input, hidden, and output layers (e.g., linear, convolutional, recurrent, etc.)
3. **Hyperparameters:** Number of layers, units per layer, activation functions, etc.

24.4. Loss Function (Criterion)

1. **Purpose:** Quantifies the error between predicted and actual outputs.
2. **Examples:**
 - (a) **Classification:** Cross-Entropy Loss
 - (b) **Regression:** Mean Squared Error (MSE)
 - (c) **Others:** Hinge Loss, Huber Loss, etc.

24.5. Optimizer

1. **Purpose:** Updates model parameters to minimize the loss function.
2. **Examples:**

- (a) SGD (Stochastic Gradient Descent)
- (b) Adam
- (c) RMSProp
- (d) Adagrad

3. **Hyperparameters:** Learning rate, momentum, weight decay, etc.

24.6. Training Loop

1. **Steps:**
 - (a) Forward Pass
 - (b) Compute Loss
 - (c) Backward Pass (Gradient Computation)
 - (d) Optimizer Step (Parameter Update)
2. **Epochs:** Number of times the entire training data is passed through the model.
3. **Batch Size:** Number of samples processed before the model is updated.

24.7. Evaluation Metrics

1. Used to assess model performance
2. **Classification:** Accuracy, Precision, Recall, F1-Score, ROC-AUC
3. **Regression:** RMSE, MAE, R^2
4. **Custom Metrics:** Domain-specific metrics

24.8. Validation & Testing

1. **Validation:** Tune hyperparameters, early stopping.
2. **Test:** Final evaluation to report performance.

24.9. Model Saving & Loading

1. **Serialization:** Save model weights and architecture.
2. **Formats:** .pt, .pth (PyTorch), .h5, .pb (TensorFlow)

24.10. Inference / Deployment

1. **Inference:** Making predictions on new data.
2. **Deployment:** Integrating the model into production (e.g., via REST API, mobile app, web app, etc.)

24.11. Experiment Tracking

1. **Tools:** TensorBoard, Weights & Biases, MLflow, etc.
2. **Track:** Parameters, metrics, model versions, logs.

24.12. Reproducibility & Documentation

1. Code versioning (Git)
2. Environment management (Docker, Conda)
3. Documentation (README, Jupyter notebooks, etc.)

Optional (but valuable) components

1. **Data Augmentation:** Especially for image/audio/text data
2. **Transfer Learning:** Pretrained models
3. **Hyperparameter Tuning:** Grid Search, Random Search, Bayesian Optimization
4. **Ensembling:** Combining predictions of multiple models
5. Fairness & Bias Evaluation
6. **Model Explainability:** SHAP, LIME, Grad-CAM, etc.

CHAPTER 25

★ MACHINE LEARNING PARADIGMS [7]

- | | |
|--|---|
| <ul style="list-style-type: none">1. Supervised Learning<ul style="list-style-type: none">(a) Fully Supervised Learning(b) Weakly Supervised Learning<ul style="list-style-type: none">i. Incomplete Supervision (e.g., partial labels)ii. Inexact Supervision (e.g., coarse labels)iii. Inaccurate Supervision (e.g., noisy labels)2. Unsupervised Learning<ul style="list-style-type: none">(a) Clustering(b) Dimensionality Reduction(c) Density Estimation(d) Representation Learning3. Semi-Supervised Learning<ul style="list-style-type: none">(a) Transductive (train/test on same data pool)(b) Inductive (learn general function from few labels)4. Self-Supervised Learning<ul style="list-style-type: none">(a) Contrastive Learning(b) Predictive Tasks (e.g., next token)(c) Masked Modeling(d) Pretext Tasks (general)5. Reinforcement Learning<ul style="list-style-type: none">(a) Model-Free(b) Model-Based<ul style="list-style-type: none">(<i>Supervision from reward signals</i>) | <ul style="list-style-type: none">6. Active Learning<ul style="list-style-type: none">(a) Pool-based Sampling(b) Stream-based Sampling(c) Query Synthesis7. Online Learning<ul style="list-style-type: none">(a) Full Information(b) Bandit Feedback (partial feedback)8. Few-shot / One-shot / Zero-shot Learning<ul style="list-style-type: none">(a) Meta-Learning(b) Prompt-based / Embedding-based methods9. Multi-Task Learning<ul style="list-style-type: none">(a) Hard Parameter Sharing(b) Soft Parameter Sharing10. Multi-Instance Learning<ul style="list-style-type: none">(<i>Labels associated with bags of instances</i>)11. Continual / Lifelong Learning<ul style="list-style-type: none">(a) Task-Incremental(b) Domain-Incremental(c) Class-Incremental12. Transfer Learning<ul style="list-style-type: none">(a) Inductive(b) Transductive(c) Unsupervised Transfer13. Curriculum Learning<ul style="list-style-type: none">(a) Data is presented in increasing difficulty |
|--|---|

Supervised Learning

Learn from labeled data (input → known output)

Classic ML

1. Linear Regression
2. Logistic Regression
3. SVM
4. Decision Trees / Random Forest
5. k-NN
6. Naïve Bayes

Deep Learning

1. Feedforward Neural Networks (MLP)
2. Convolutional Neural Networks (CNNs)
3. Recurrent Neural Networks (RNNs)
4. LSTMs / GRUs
5. Transformers (for vision, text, etc.)

Unsupervised Learning

Find structure/ patterns in unlabeled data

Classic ML

1. k-Means
2. DBSCAN
3. Hierarchical Clustering
4. PCA, ICA
5. GMM, KDE

Deep Learning

1. Autoencoders
2. Variational Autoencoders (VAE)
3. Generative Adversarial Networks (GANs)
4. Self-Organizing Maps (SOMs, sometimes hybrid)
5. Deep Embedding Clustering

Semi-Supervised Learning

Mix of labeled and unlabeled data

Deep Learning

Classic ML

1. Label Propagation
2. Self-Training

1. Pseudo-Labeling with CNNs / Transformers
2. Consistency Regularization
3. FixMatch

Self-Supervised Learning

Labels generated from the data itself (no manual labels)

Deep Learning

1. Contrastive Learning (SimCLR, MoCo)
2. Masked Modeling (BERT, MAE)
3. Autoencoding pretraining

Reinforcement Learning

Learn via trial and error with rewards

Classic ML

1. Q-Learning
2. SARSA
3. Policy Iteration / Value Iteration

Deep Learning

1. Deep Q-Networks (DQN)
2. Actor-Critic, A3C, PPO, DDPG
3. AlphaGo-style systems (Deep + MCTS)

Online / Incremental Learning

Classic ML

1. Online Perceptron
2. SGD updates

Deep Learning

1. Continual Learning
2. Elastic Weight Consolidation (EWC)

Ensemble Learning

Classic ML

1. Bagging
2. Boosting
3. Stacking

Deep Learning

1. Model Ensembling
2. Snapshot Ensembles

Evolutionary / Bio-Inspired Learning

Classic ML

1. Genetic Algorithms, PSO, ACO

Deep Learning

1. Neuroevolution (e.g., evolving network architectures — NEAT, Genetic CNNs)

CHAPTER 26

★OPTIMIZER

★LOSS FUNCTION (CRITERION)

Regression related Loss

27.1. Sum of Errors (SE)

$$L = \sum_{i=1}^n (\hat{y}_i - y_i) \quad [39]$$

27.2. Sum of Absolute Errors (SAE)

$$L = \sum_{i=1}^n (|\hat{y}_i - y_i|) \quad [39]$$

27.3. Sum of Squared Errors (SSE)

$$L = \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad [39]$$

27.4. Mean Absolute Error (MAE) / L1 Loss

$$L = \frac{1}{n} \sum_{i=1}^n (|\hat{y}_i - y_i|) \quad [39]$$

27.5. Mean Square Error (MSE) / L2 Loss

$$L = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad [39]$$

27.6. Root Mean Square Error (RMSE) Loss

$$L = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2} \quad [39]$$

27.7. Huber Loss / Smooth Mean Absolute Error

27.8. Mean Bias Error (MBE)

Classification related Loss

27.9. Binary Cross-Entropy Loss / Log Loss

$$L = -y \log(\hat{y}) - (1-y) \log(1-\hat{y}) \quad [40]$$

27.10. Categorical Cross-Entropy Loss

27.11. Hinge Loss

1.

27.12. Log Loss

Regularization

1. Regularization is an important technique in machine learning that helps to improve model accuracy by preventing overfitting which happens when a model learns the training data too well including noise and outliers and perform poor on new data. [45]
2. By adding a penalty for complexity it helps simpler models to perform better on new data. [45]

27.13. Early Stopping

1. Stops training when validation performance deteriorates, preventing overfitting by halting before the model memorizes training data. [48]

27.14. Dropout

1. Dropout technique repeatedly ignores random subsets of neurons during training, which simulates the training of multiple neural network architectures at once to improve generalization. [48]

27.15. Lasso/ L1 Regression

1. Regularization Term = $\sum |w_i|$ [45]
2. **Regularized loss function or Objective function:** $L'(\mathbf{w}) = L(\mathbf{w}) + \lambda \sum |w_i|$
 λ : weight decay coefficient (hyperparameter)
3. A regression model which uses the L1 Regularization technique is called LASSO (Least Absolute Shrinkage and Selection Operator) regression. Techniques such as this are known in the statistics literature as **shrinkage** methods because they reduce the value of the coefficients. [45]
4. It adds the absolute value of magnitude of the coefficient as a penalty term to the loss function(L). [45]
5. This penalty can shrink some coefficients to zero which helps in selecting only the important features and ignoring the less important ones. [45]

27.16. Ridge/ L2 Regression

1. Regularization Term = $\sum w_i^2 = \mathbf{x}^\top \mathbf{x}$ [45]
2. **Regularized loss function or Objective function:** $L'(\mathbf{w}) = L(\mathbf{w}) + \lambda \mathbf{x}^\top \mathbf{x}$
 λ : weight decay coefficient (hyperparameter)
3. A regression model that uses the L2 regularization technique is called Ridge regression. [45]
4. It adds the squared magnitude of the coefficient as a penalty term to the loss function(L). [45]
5. It handles multicollinearity by shrinking the coefficients of correlated features instead of eliminating them. [45]

27.17. Elastic Net/ L1-L2 Regression

1. Regularization Term = $(1 - \alpha) \sum |w_i| + \alpha \sum w_i^2$
 $0 \leq \alpha \leq 1$: Mixing parameter, $\alpha = 0$: Lasso, $\alpha = 1$: Ridge [45]

2. Elastic Net Regression is a combination of both L1 as well as L2 regularization. [45]
3. That shows that we add the absolute norm of the weights as well as the squared measure of the weights. [45]
4. With the help of an extra hyperparameter that controls the ratio of the L1 and L2 regularization. [45]

CHAPTER 28

MACHINE LEARNING: INTRODUCTION

N	number of training records
$\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$	input records
$\{\mathbf{y}_1, \dots, \mathbf{y}_N\}$	(ground truth/ reference) outputs
$\mathbf{y}(\cdot)$	model/ function
$\hat{\mathbf{y}}_i = \mathbf{y}(\mathbf{x}_i)$	(hypothesis) output of i -th input

Notations

28.1. Intro

1. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalization is a central goal in pattern recognition. [41]

28.2. Pre-processing: Feature Extraction

1. For most practical applications, the original input variables are typically preprocessed to transform them into some new space of variables where, it is hoped, the pattern recognition problem will be easier to solve. [41]
2. Pre-processing might also be performed in order to speed up computation. [41]
3. Care must be taken during pre-processing because often information is discarded, and if this information is important to the solution of the problem then the overall accuracy of the system can suffer. [41]

28.3. Training/ Learning Phase

1. **Training set:** large set of N records $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ that can be used to tune the parameters of an adaptive model. [41]
2. The result of running the machine learning algorithm can be expressed as a function $\mathbf{y}(\mathbf{x}_i)$ which takes a new input record \mathbf{x} as input and that generates an output vector \mathbf{y}_i , encoded in the same way as the target vectors. The precise form of the function $\mathbf{y}(\cdot)$ is determined during the training phase, also known as the learning phase, on the basis of the **training data**. [41]

28.4. Testing/ Evaluation Phase

1. Once the model is trained it can then determine the identity of new digit images, which are said to comprise a **test set**. [41]

2. The ability to categorize correctly new examples that differ from those used for training is known as **generalization**. [41]

28.5. Supervised learning

1. Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as **supervised learning** problems. [41]

28.5.1. Regression

1. Cases in which the desired output consists of one or more continuous variables, then the task is called **regression**. [41]

28.5.2. Classification

1. Cases in which the aim is to assign each input vector to one of a finite number of discrete categories, are called **classification** problems. [41]

28.6. Unsupervised learning

1. the training data consists of a set of input vectors x without any corresponding target values [41]

28.6.1. Clustering

1. Cases to discover groups of similar examples within the data [41]

28.6.2. Density Estimation

1. Cases to determine the distribution of data within the input space [41]

28.6.3. Visualization

1. Cases to project the data from a high-dimensional space down to two or three dimensions [41]

28.7. Reinforcement Learning

1. It is concerned with the problem of finding suitable actions to take in a given situation in order to maximize a reward. [41]

2. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. [41]

3. Typically there is a sequence of states and actions in which the learning algorithm is interacting with its environment. [41]

4. In many cases, the current action not only affects the immediate reward but also has an impact on the reward at all subsequent time steps. [41]

5. **Credit assignment problem:** [41]

- (a) A major challenge is that a case can involve dozens of moves, and yet it is only at the end of the case that the reward, in the form of victory, is achieved. [41]

- (b) The reward must then be attributed appropriately to all of the moves that led to it, even though some moves will have been good ones and others less so. [41]
- 6. A general feature of reinforcement learning is the trade-off between **exploration**, in which the system tries out new kinds of actions to see how effective they are, and **exploitation**, in which the system makes use of actions that are known to yield a high reward. [41]
- 7. Too strong a focus on either exploration or exploitation will yield poor results. [41]

28.8. Model Comparison OR Model Selection

- 1. A bad model can cause *overfitting* or *underfitting* and sometimes even lead to increased computational costs. [44]
- 2. Different models have different ways of processing data and choosing the right one ensures that the system works efficiently.
 - (a) A simple model cannot capture details and has poor accuracy, while a model too complex might overfit that is doing very well on training data but fails on new data. [44]
 - (b) The goal is to find a model that learns patterns effectively without being too simple or too complex. [44]
- 3. Proper model selection involves experimenting with different models and comparing their performance using evaluation metrics such as accuracy, precision, recall or mean squared error. These metrics help in determining which model is best suited for a given task. [44]
- 4. Apart from performance metrics, other factors such as training time, dataset size and available computing power also play a crucial role in choosing the right model. [44]
- 5. Selecting an appropriate model not only improves prediction accuracy but also enhances efficiency, making the system faster and more reliable. [44]

28.8.1. Grid Search

- 1. One of the simplest and most commonly used model selection techniques [44]
- 2. In this approach, systematically different combinations of hyper-parameters are tried and that gives the best performance chosen. [44]
- 3. It can be effective, but the main drawback will be computationally intensive, especially for complex models and many parameters. [44]

28.8.2. Random Search

- 1. Similar to grid search, random search doesn't check all possible combinations. Instead, it randomly chooses a subset of the hyperparameter combinations. [44]
- 2. The random search method often runs much faster than the grid search method and yet achieves equally good results. [44]

28.8.3. Bayesian Optimization

- 1. Bayesian optimization is a smarter approach to model selection. [44]
- 2. Instead of just randomly searching for the best hyper-parameters, it uses probability models to predict which parameters are likely to perform best and focuses on evaluating those. [44]
- 3. This method is efficient and often finds better results than grid or random search. [44]

28.8.4. Cross-Validation Based Selection

1. This method involves using cross-validation to evaluate multiple models and selecting the one with the best average performance. [44]
2. Instead of relying on a single train-test split, cross-validation divides the dataset into multiple parts and trains the model on different subsets. This helps to ensure that the model's performance is not just due to a specific split of data. [44]
3. By averaging the results from different splits, we get how well the model will perform on new, unseen data. [44]
4. This approach reduces the risk of overfitting and helps in selecting a good model. [44]

28.8.4.1. Holdout Validation

1. In Holdout Validation we perform training on the 50% of the given dataset and rest 50% is used for the testing purpose. [43]
2. It's a simple and quick way to evaluate a model. [43]
3. The major drawback of this method is that we perform training on the 50% of the dataset, it may be possible that the remaining 50% of the data contains some important information which we are leaving while training our model that can lead to higher bias. [43]

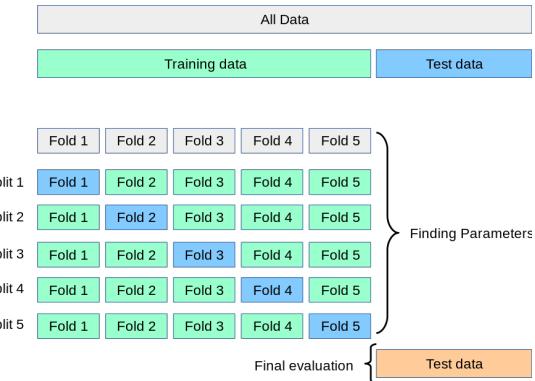
28.8.4.2. LOOCV (Leave One Out Cross Validation)

1. In this method we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. [43]
2. In LOOCV the model is trained on $n - 1$ samples and tested on the one omitted sample repeating this process for each data point in the dataset. [43]
3. An advantage of using this method is that we make use of all data points and hence it is low bias. [43]
4. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. [43]
5. Another drawback is it takes a lot of execution time as it iterates over the number of data points we have. [43]

28.8.4.3. Stratified Cross-Validation

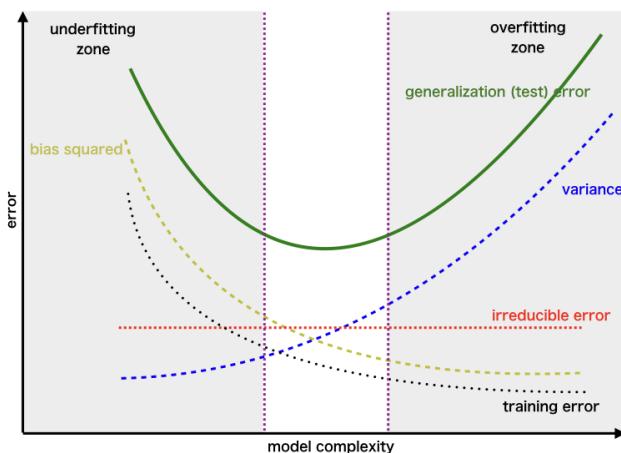
1. It is a technique used in machine learning to ensure that each fold of the cross-validation process maintains the same class distribution as the entire dataset. [43]
2. This is particularly important when dealing with **imbalanced datasets** where certain classes may be under represented. [43]
3. In this method:
 - (a) The dataset is divided into k folds while maintaining the proportion of classes in each fold. [43]
 - (b) During each iteration, one-fold is used for testing and the remaining folds are used for training. [43]
 - (c) The process is repeated k times with each fold serving as the test set exactly once. [43]
4. Stratified Cross-Validation is essential when dealing with classification problems where maintaining the balance of class distribution is crucial for the model to generalize well to unseen data. [43]

28.8.4.4. K-Fold Cross Validation



1. In K-Fold Cross Validation we split the dataset into k number of subsets known as folds then we perform training on the all the subsets but leave one ($k - 1$) subset for the evaluation of the trained model. [43]
2. In this method, we iterate k times with a different subset reserved for testing purpose each time. [43]

28.9. Bias-Variance Trade Off



28.9.0.1. Bias

1. The bias is known as the difference between the prediction of the values by the Machine Learning model and the correct value. [42]
2. It recommended that an algorithm should always be low-biased to avoid the problem of underfitting. [42]
3. Being high in biasing gives a large error in training as well as testing data. Such fitting is known as the Underfitting of Data. [42]

28.9.0.2. Variance

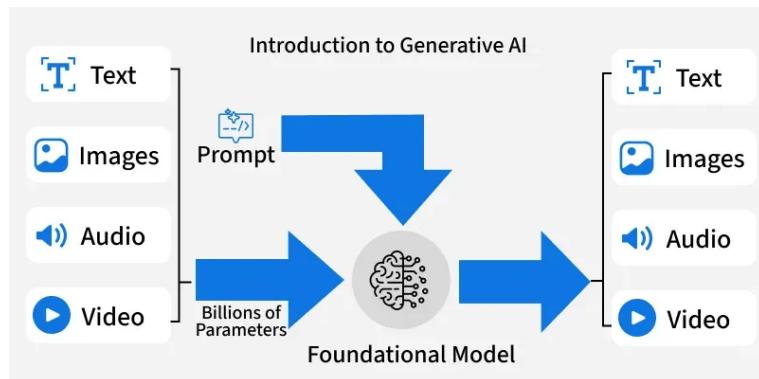
1. The variability of model prediction for a given data point which tells us the spread of our data is called the variance of the model. [42]
2. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before. As a result, such models perform very well on training data but have high error rates on test data. When a model is high on variance, it is then said to as Overfitting of Data. [42]

3. While training a data model variance should be kept low.

[42]

CHAPTER 29

GENERATIVE AI (GENAI)



Generative AI [46]

1. Generative AI is a type of artificial intelligence designed to create new content such as text, images, music or even code by learning patterns from existing data. [46]
2. These models generate original outputs that are often indistinguishable from human-created content. [46]
3. These models use techniques like deep learning and neural networks to generate output. [46]

29.1. How Generative AI Works?

29.1.1. Core Mechanism (Training & Inference)

1. Generative AI is trained on **large datasets** like text, images, audio or video using deep learning networks. [46]
2. During **training**, the model learns parameters (millions or billions of them) that help them predict or generate content. [46]
3. Here models generate output based on **learned patterns** and **prompts** provided. [46]

29.1.2. By Media Type

1. **Text:** Uses large language models (LLMs) to predict the next token in a sequence, enabling coherent paragraph or essay generation. [46]
2. **Images:** Diffusion models like DALL·E or Stable Diffusion start with noise and iteratively denoise to create realistic visuals [46]
3. **Speech:** Text-to-speech models synthesize human-like voice by modeling acoustic features based on prompt. [46]
4. **Video:** Multimodal systems like Sora by OpenAI or Runway generate short, temporally coherent video clips from text or other prompts [46]

29.1.3. Agents in Generative AI

1. Modern systems often uses agents which are **autonomous components** that interact with the environment, **obtain information** and **execute chains of tasks**. [46]
2. These agents uses LLMs to reason, plan and act enabling workflows like querying databases, performing retrieval or controlling external APIs. [46]

29.1.4. Training and Fine-Tuning

1. LLMs are trained on massive general corpora (e.g., web text) using self-supervised methods. [46]
2. These models become pre-trained models which can be further trained on domain-specific labeled data to **adapt** to specialized tasks or stylistic needs. This technique is called fine tuning and it can be done using:

(a) LoRA [46]	(c) Peft [46]
(b) QLoRA [46]	(d) LLM Distilation [46]
(e) Reinforcement Learning from Human Feedback (RLHF) [46]	

29.1.5. Retrieval-Augmented Generation (RAG)

1. Modern systems also uses RAG which enhances outputs by retrieving relevant documents at query time to ground the generation in accurate, up-to-date information, reducing hallucinations and improving factuality. [46]
2. The process typically involves:
 - (a) **Indexing** documents into embeddings stored in vector databases [46]
 - (b) **Retrieval** of relevant passages [46]
 - (c) **Augmentation** of the prompt with retrieved content [46]
 - (d) **Generation** of grounded, informed responses [46]

29.2. Types of Generative AI Models

29.2.1. Transformers or Autoregressive Models

1. **Auto-regressive** Transformers Models generate sequences by predicting the next token based on all previous ones moving step by step through the text. [46]
2. The architecture relies on the transformer's self attention mechanism to capture context from the entire input so far making it highly effective for natural language and code generation. [46]
3. Popular examples include GPT models which can produce coherent, context aware paragraphs, solve coding tasks or answer complex queries. [46]
4. The autoregressive approach gives fine grained control over each output step but can be slower for long generations since tokens are generated one at a time. [46]

29.2.2. Diffusion Models

1. Diffusion models generate data such as images or audio by starting with pure random noise and gradually refining it into a coherent output through a series of denoising steps. [46]
2. Each step reverses a simulated diffusion process that added noise to real data during training. [46]
3. This iterative approach can produce highly detailed and realistic results specially in image synthesis where models like Stable Diffusion and DALL-E 3 have set benchmarks. [46]
4. Diffusion models are also versatile they can be adapted for inpainting, style transfer and conditional generation from text prompts. [46]

29.2.3. Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs)

1. VAEs and GANs were among the first deep learning architectures for generative tasks. [46]
2. A VAE encodes data into a compressed latent space and then decodes it back with a probabilistic twist that encourages smooth, continuous representations. This makes them good for controllable generation and interpolation between styles. [46]
3. GANs in contrast use two networks against each other a generator that tries to produce realistic outputs and a discriminator that tries to detect fakes. This adversarial setup leads to sharp, lifelike images though training can be unstable and prone to mode collapse. [46]

29.2.4. Encoder Decoder Models

1. Encoder decoder architectures consist of two stages: the encoder processes the input into a dense representation and the decoder generates the desired output from that representation. [46]
2. They are widely used for sequence to sequence tasks like language translation, summarization and image captioning. [46]
3. The encoder captures the full context of the input before the decoder starts producing tokens, allowing for strong performance on tasks that require global understanding rather than token by token prediction. [46]
4. Modern encoder decoder models often use transformers for both stages as in T5, BART and many multimodal system. [46]

29.3. Evaluation of Generative AI

Evaluating generative AI involves multiple dimensions because outputs can vary in accuracy, style and usefulness depending on the task. Key aspects include: [46]

1. **Fact Accuracy and Hallucination Avoidance:** Benchmarks like BEIR, Natural Questions assess factual correctness. Techniques like RAG and fine-tuning reduce hallucinations and ground responses in reliable data. [46]
2. **Quality Metrics:** Outputs are judged on fluency, coherence, logical consistency and contextual relevance. Commonly used metrics are BLEU, ROUGE, METEOR, FID and IS. [46]
3. **Efficiency and Accuracy Trade-Offs:** LoRA and QLoRA helps in balancing performance with computational cost making models faster and lighter without losing quality. [46]
4. **Resilience to Retrieval Noise:** Advanced approaches like “Finetune-RAG” improve model accuracy by training the model to handle imperfect retrieval inputs hence increasing factual reliability. [46]

5. **Creativity and Diversity:** Models should generate varied and original outputs rather than repetitive or biased ones. [46]
6. **Bias and Fairness:** Evaluation includes checking whether outputs reflect harmful stereotypes or unfair treatment of groups. Tools like Bias Benchmark for QA (BBQ) and StereoSet measure bias levels. [46]
7. **User Experience and Usefulness:** Beyond technical metrics, effectiveness is judged by how well the system supports users in real scenarios like chatbots providing relevant, actionable responses. [46]

29.4. Applications of Generative AI

1. **Text:** Powers chatbots, virtual assistants, content creation, document summarization and even code generation tools like GitHub Copilot. [46]
2. **Images:** Used in digital art, product design, fashion, advertising and medical imaging to create visuals that are close to real-world examples. [46]
3. **Audio & Speech:** Enables natural-sounding voice assistants, multilingual dubbing, music composition, personalized voice cloning and accessibility tools. [46]
4. **Video:** Supports animation, movie special effects, gaming, marketing videos and realistic training simulations. [46]
5. **Business Use Cases:** Enhances customer support with AI agents, boosts knowledge discovery using RAG systems, accelerates drug discovery, assists in financial forecasting and improves data-driven decision-making. [46]

29.5. Advantages

1. **Accelerates Research and Development:** In fields like science and technology, Generative AI reduces the time needed for research by generating multiple outcomes and predictions such as molecular structures in drug development. This speeds up innovation and helps solve complex problems efficiently. [46]
2. **Improves Personalization:** Generative AI creates tailored content based on user preferences. From personalized product designs to customized marketing campaigns it enhances user engagement and satisfaction by delivering exactly what users need or want. [46]
3. **Empowers Non Experts:** Even users without expertise can create high quality content using Generative AI. This helps individuals learn new skills access creative tools and open doors to personal and professional growth. [46]
4. **Drives Economic Growth:** Generative AI introduces new roles and opportunities by fostering innovation, automating tasks and enhancing productivity. This leads to economic expansion and the creation of jobs in emerging fields. [46]

29.6. Disadvantages

1. **Data Dependence:** The accuracy and quality of Generative AI outputs depend entirely on the data it is trained on. If the training data is biased, incomplete or inaccurate the generated content will reflect these flaws. [46]
2. **Limited Control Over Outputs:** It can produce unexpected or irrelevant results making it challenging to control the content and ensure it aligns with specific user requirements. [46]

3. **High Computational Requirements:** Training and running Generative AI models demand significant computing power which can be costly and resource intensive. This limits accessibility for smaller organizations or individuals. [46]
4. **Ethical and Legal Concerns:** It can be misused to create harmful content like deepfakes or fake news which can spread misinformation or violate privacy. These ethical and legal challenges require careful regulation and oversight to prevent abuse. [46]

CHAPTER 30

AGENTIC AI

1. Agentic AI is an artificial intelligence system that can accomplish a specific goal with **limited supervision**. [47]
2. It consists of **AI agents**—machine learning models that mimic human decision-making to solve problems in real time. [47]
3. In a **multiagent system**, each agent performs a specific subtask required to reach the goal and their efforts are coordinated through AI orchestration. [47]
4. Unlike traditional AI models, which operate within predefined constraints and require human intervention, agentic AI exhibits autonomy, goal-driven behavior and adaptability. [47]
5. The term “agentic” refers to these models’ agency, or, their capacity to act independently and purposefully. [47]
6. Agentic AI builds on **generative AI** (gen AI) techniques by using large language models (LLMs) to function in dynamic environments. While generative models focus on creating content based on learned patterns, agentic AI extends this capability by applying generative outputs toward specific goals. [47]
7. **Example:** A generative AI model like OpenAI’s ChatGPT might produce text, images or code, but an agentic AI system can use that generated content to complete complex tasks autonomously by calling external tools. Agents can, for example, not only tell you the best time to climb Mt. Everest given your work schedule, it can also book you a flight and a hotel. [47]

30.1. How agentic AI works

1. **Perception:** Agentic AI begins by collecting data from its environment through sensors, APIs, databases or user interactions. This step ensures that the system has up-to-date information to analyze and act upon. [47]
2. **Reasoning:** Once the data is collected, the AI processes it to extract meaningful insights. Using natural language processing (NLP), computer vision or other AI capabilities, it interprets user queries, detects patterns and understands the broader context. This ability helps the AI determine what actions to take based on the situation. [47]
3. **Goal setting:** The AI sets objectives based on predefined goals or user inputs. It then develops a strategy to achieve these goals, often by using decision trees, reinforcement learning or other planning algorithms. [47]
4. **Decision-making:** AI evaluates multiple possible actions and chooses the optimal one based on factors such as efficiency, accuracy and predicted outcomes. It might use probabilistic models, utility functions or machine learning-based reasoning to determine the best course of action. [47]
5. **Execution:** After selecting an action, the AI executes it, either by interacting with external systems (APIs, data, robots) or providing responses to users. [47]
6. **Learning and adaptation:** After executing an action, the AI evaluates the outcome, gathering feedback to improve future decisions. Through reinforcement learning or self-supervised learning, the AI refines its strategies over time, making it more effective in handling similar tasks in the future. [47]
7. **Orchestration:** AI orchestration is the coordination and management of systems and agents. Orchestration platforms automate AI workflows, track progress toward task completion, manage resource usage,

monitor data flow and memory and handle failure events. With the right architecture, dozens, hundreds or even thousands of agents could theoretically work together in harmonious productivity. [47]

30.2. Advantages

Autonomous

1. The most important advancement of agentic systems is that they allow for autonomy to perform tasks without constant human oversight. [47]
2. Agentic systems can maintain long-term goals, manage multistep problem-solving tasks and track progress over time. [47]

Proactive

1. Agentic systems provide the flexibility of LLMs, which can generate responses or actions based on nuanced, context-dependent understanding, with the structured, deterministic and reliable features of traditional programming. [47]
2. This approach allows agents to “think” and “do” in a more human-like fashion. [47]
3. LLMs by themselves can’t directly interact with external tools or databases or set up systems to monitor and collect data in real time, but agents can. [47]
4. Agents can search the web, call application programming interfaces (APIs) and query databases, then use this information to make decisions and take actions. [47]

Specialized

1. Agents can specialize in specific tasks. Some agents are simple, performing a single repetitive task reliably. [47]
2. Others can use perception and draw on memory to solve more complex problems. [47]
3. An agentic architecture might consist of a “conductor” model powered by an LLM that oversees tasks and decisions and supervises other, simpler agents. Such architectures are ideal for sequential workflows but are vulnerable to bottlenecks. [47]
4. Other architectures are more horizontal, with agents working in harmony as equals in a decentralized fashion, but this architecture can be slower than a vertical hierarchy. [47]

Adaptable

1. Agents can learn from their experiences, take in feedback and adjust their behavior. [47]
2. With the right guardrails, agentic systems can improve continuously. [47]
3. Multiagent systems possess the scalability to eventually handle broadly scoped initiatives. [47]

Intuitive

1. Because agentic systems are powered by LLMs, users can engage with them with natural language prompts. This means that entire software interfaces—think of the many tabs, dropdowns, charts, sliders, pop-ups and other UI elements involved in the SaaS platform of one’s choice—can be replaced by simple language or voice commands. [47]

2. Theoretically, any software user experience can now be reduced to “talking” with an agent, who can fetch the information one needs and take action based on that information. This productivity benefit can barely be overstated, when one considers the time it takes for workers to learn and master new interfaces and tools. [47]

30.3. Challenges

1. The autonomous nature can bring serious consequences if agentic systems go “off the rails.” The usual AI risks apply, but can be magnified in agentic systems. [47]
2. Many agentic AI systems use reinforcement learning, which involves maximizing a reward function. If the reward system is poorly designed, the AI might exploit loopholes to achieve “high scores” in unintended ways. [47]
3. Some agentic AI systems can become self-reinforcing, escalating behaviors in an unintended direction. This issue happens when the AI optimizes too aggressively for a particular metric without safeguards. [47]
4. Because agentic systems are often composed of multiple autonomous agents working together, there are opportunities for failure. [47]
5. Traffic jams, bottlenecks, resource conflicts—all of these errors have the potential to cascade. [47]

CHAPTER 31

ENCODER-DECODER ARCHITECTURE

Domain	Task	Example	Encoder	Decoder	seq2seq?
Computer Vision	Autoencoders	VAE, Denoising AE	CNN	CNN/Deconv	
	Image Segmentation	U-Net	CNN	CNN (upsampling)	
	Image Generation	Diffusion models, VAEs, GANs	CNN Transformer	/ CNN Transformer	/
Representation Learning	Learn embeddings for downstream tasks	SimCLR, BYOL	CNN	Projection MLP	
Speech	Speech → Text (ASR)	Whisper	CNN/Transformer	Transformer	YES
Music	Melody → Accompaniment	Music Transformer	Transformer	Transformer	YES
Text (NLP)	Autoencoding masked tokens	BERT	Transformer encoder	Transformer decoder (only for MLM head)	
Multimodal Models	Image → Caption	CLIP, Flamingo, BLIP	Vision encoder	Text decoder	Partly
Generative Biology	Protein sequence → structure	AlphaFold	Transformer encoder	Structure generator	
Anomaly Detection	Reconstruct normal pattern	Autoencoder	MLP/CNN	MLP/CNN	

Table 31.1: Encoder–Decoder Usage by category [7]

VII

SUPERVISED LEARNING

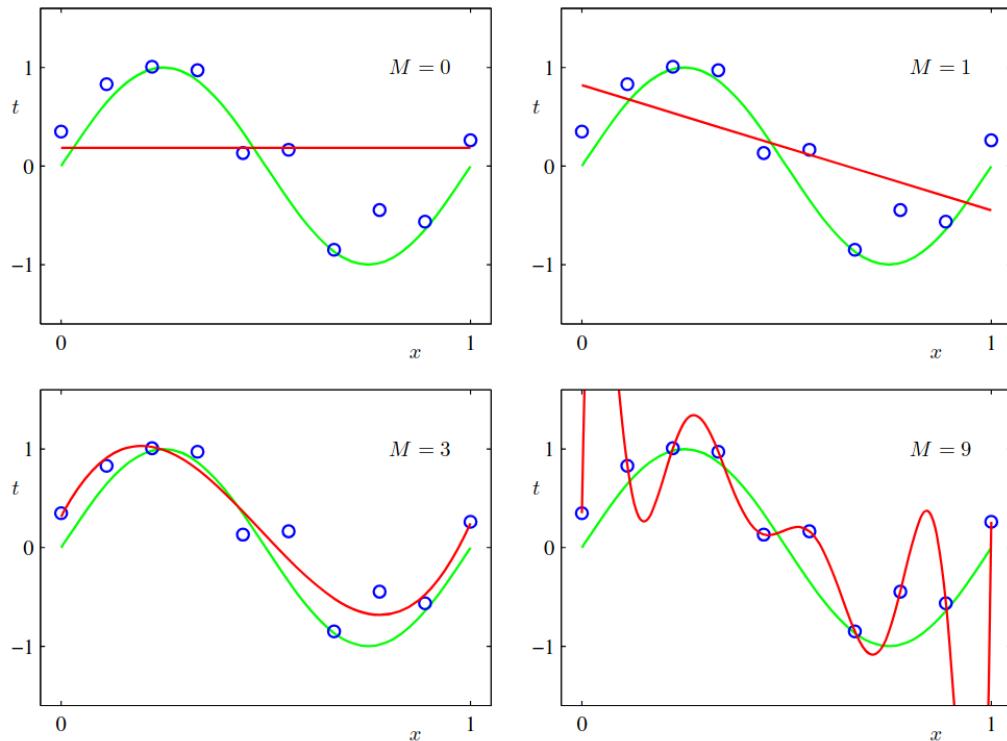
CHAPTER 32

REGRESSION

32.1. Polynomial Curve Fitting

N	number of observations (in training set)	known
M	order of polynomial	hyper-param
$\mathbf{x} \equiv (x_1, \dots, x_N)^\top$	set of observations	known
$\mathbf{t} \equiv (t_1, \dots, t_N)^\top$	set of values for the observations	known
$\mathbf{w} \equiv (w_0, \dots, w_M)^\top$	weights (polynomial coefficients)	unknown (param)
\hat{x}	new input data	
\hat{t}	predicted value	

Notations



Plots of polynomials having various orders M fitted to the data set [41]
 green curves: original $\sin(2\pi x)$ curve | blue dots: data points | red curves: prediction polynomial

1. Polynomial function: $\hat{t}_i = y(x_i, \mathbf{w}) = w_0 + w_1 x_i + w_2 x_i^2 + \dots + w_M x_i^M = \sum_{j=0}^M w_j x_i^j$ [41]

2. Error: $E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (\hat{t}_n - t_n)^2$ $\hat{t}_n = y(x_n, \mathbf{w})$ [41]

- (a) $\frac{1}{2}$ is used instead of $\frac{1}{N}$ for mathematical convenience (while differentiation)
3. We can solve the curve fitting problem by choosing the value of \mathbf{w} for which $E(\mathbf{w})$ is as small as possible. Because the error function is a quadratic function of the coefficients \mathbf{w} , its derivatives with respect to the coefficients will be linear in the elements of \mathbf{w} , and so the minimization of the error function has a unique solution, denoted by \mathbf{w}^* , which can be found in **closed form**. The resulting polynomial is given by the function $y(x, \mathbf{w}^*)$. [41]

VIII

RECURRENT NEURAL NETWORK (RNN)

CHAPTER 33

RECURRENT NEURAL NETWORKS (RNN) (1982)

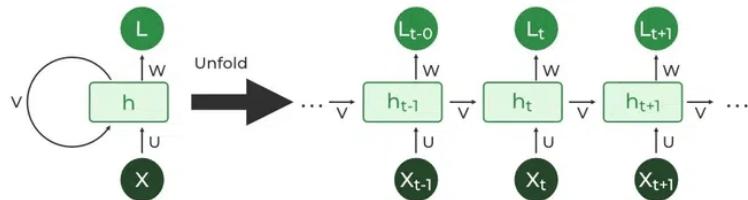


Fig. 33.1: Recurrent Neural Networks: Unfolding [55]

1. Recurrent Neural Networks (RNNs) differ from regular neural networks in how they process information. While standard neural networks pass information in one direction i.e. from input to output, RNNs feed information back into the network at each step. [55]
2. RNNs work by “remembering” past information and passing the output from one step as input to the next i.e it considers all the earlier words to choose the most likely next word. This memory of previous steps helps the network understand context and make better predictions. [55]
3. Recurrent neural networks (RNNs) are a promising architecture for general-purpose sequence transduction. [6]
4. The combination of a high-dimensional multivariate internal state and nonlinear state-to-state dynamics offers more expressive power than conventional sequential algorithms such as hidden Markov models. [6]
5. In particular, RNNs are better at storing and accessing information over long periods of time. [6]
6. RNNs are usually restricted to problems where the alignment between the input and output sequence is known in advance. For example, RNNs may be used to classify every frame in a speech signal, or every amino acid in a protein chain. If the network outputs are probabilistic this leads to a distribution over output sequences of the same length as the input sequence. [6]
7. Unlike traditional feedforward neural networks which process inputs as fixed-length vectors, RNNs can manage **variable-length sequences** by maintaining a hidden state that stores information from previous steps in the sequence. [13]

33.1. Key Components of RNNs

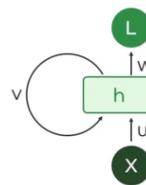


Fig. 33.2: RNN: Neuron (Recurrent Unit) [55]

1. **Recurrent Neurons (Recurrent Unit):** They hold a hidden state that maintains information about previous inputs in a sequence. Recurrent units can "remember" information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time. [55]

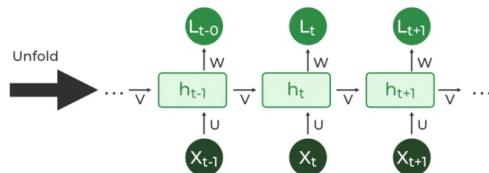


Fig. 33.3: RNN: Neuron (Recurrent Unit) [55]

2. **RNN Unfolding:** RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step. This unrolling enables backpropagation through time (BPTT) a learning process where errors are propagated across time steps to adjust the network's weights enhancing the RNN's ability to learn dependencies within sequential data. [55]

33.2. How does RNN work?

Forward Pass

1. **State Update:** $h_t = f(h_{t-1}, x_t)$ [55]
 - (a) h_t is the current state [55]
 - (b) h_{t-1} is the previous state [55]
 - (c) x_t is the input at the current time step [55]
2. **Activation Function Application:** $h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$ [55]
Here, W_{hh} is the weight matrix for the recurrent neuron and W_{xh} is the weight matrix for the input neuron. [55]
3. **Output Calculation:** $y_t = W_{hy} \cdot h_t$ [55]
where y_t is the output and W_{hy} is the weight at the output layer. [55]
4. These parameters are updated using backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as backpropagation through time. [55]

Backward Pass/ Backpropagation

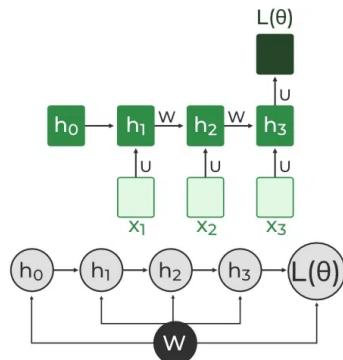


Fig. 33.4: Backpropagation Through Time (BPTT) In RNN

1. The loss function $L(\theta)$ depends on the final hidden state h_n and each hidden state relies on preceding ones forming a sequential dependency chain: h_n depends on h_{n-1} , h_{n-1} depends on h_{n-2} , \dots , h_1 depends on h_0 . [55]

2. **Simplified Gradient Calculation:**
$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_n} \cdot \frac{\partial h_n}{\partial W}$$
 [55]

3. **Handling Dependencies in Layers:** Each hidden state is updated based on its dependencies: $h_n = \sigma(W \cdot h_{n-1} + b)$ [55]

The gradient is then calculated for each state, considering dependencies from previous hidden states. [55]

4. **Gradient Calculation with Explicit and Implicit Parts:** The gradient is broken down into explicit and implicit parts summing up the indirect paths from each hidden state to the weights. [55]

$$\frac{\partial h_n}{\partial W} = \frac{\partial h_n^+}{\partial W} + \frac{\partial h_n^-}{\partial h_{n-1}} \cdot \frac{\partial h_{n-1}^+}{\partial W}$$
 [55]

5. **Final Gradient Expression:** The final derivative of the loss function with respect to the weight matrix W is computed: [55]

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_n} + \sum_{k=1}^n \frac{\partial h_n}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$
 [55]

This iterative process is the essence of backpropagation through time. [55]

```

1 class VanillaRNN(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         super().__init__()
4         self.input_size = input_size
5         self.hidden_size = hidden_size
6         # input -> hidden
7         self.Wx = nn.Parameter(torch.randn(input_size, hidden_size) * 0.1)
8         # hidden -> hidden
9         self.Wh = nn.Parameter(torch.randn(hidden_size, hidden_size) * 0.1)
10        # bias for hidden
11        self.bh = nn.Parameter(torch.zeros(hidden_size))
12        # hidden -> output (many-to-one)
13        self.Why = nn.Parameter(torch.randn(hidden_size, output_size) * 0.1)
14        self.by = nn.Parameter(torch.zeros(output_size))
15
16    def forward(self, x, h0=None):

```

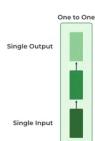
```

17     """
18     x: (batch, seq_len, input_size)
19     h0: (batch, hidden_size) or None
20     returns logits (batch, output_size) using final hidden state
21     """
22
23     batch, seq_len, _ = x.shape
24     if h0 is None:
25         h = x.new_zeros(batch, self.hidden_size)
26     else:
27         h = h0
28
29     for t in range(seq_len):
30         xt = x[:, t, :]                      # (batch, input_size)
31         preact = xt @ self.Wx + h @ self.Wh + self.bh
32         h = torch.tanh(preact)                # (batch, hidden_size)
33
34     logits = h @ self.Why + self.by          # (batch, output_size)
35
36     return logits, h

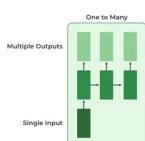
```

Python Snippet 33.1: Vanilla RNN from scratch (PyTorch) [7]

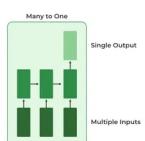
33.3. Types Of Recurrent Neural Networks



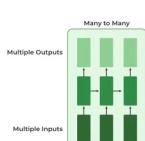
One-to-One RNN: This is the simplest type of neural network architecture where there is a single input and a single output. It is used for straightforward classification tasks such as binary classification where no sequential data is involved. [55]



One-to-Many RNN: In a One-to-Many RNN the network processes a single input to produce multiple outputs over time. This is useful in tasks where one input triggers a sequence of predictions (outputs). For example in image captioning a single image can be used as input to generate a sequence of words as a caption. [55]



Many-to-One RNN: The Many-to-One RNN receives a sequence of inputs and generates a single output. This type is useful when the overall context of the input sequence is needed to make one prediction. In sentiment analysis the model receives a sequence of words (like a sentence) and produces a single output like positive, negative or neutral. [55]



Many-to-Many RNN: The Many-to-Many RNN type processes a sequence of inputs and generates a sequence of outputs. In language translation task a sequence of words in one language is given as input and a corresponding sequence in another language is generated as output. [55]

33.4. Variants of Recurrent Neural Networks (RNNs)

1. **Vanilla RNN:** This **simplest** form of RNN consists of a single hidden layer where weights are shared across time steps. Vanilla RNNs are suitable for learning short-term dependencies but are limited by the vanishing gradient problem, which hampers long-sequence learning. [55]
2. **Bidirectional RNNs:** Bidirectional RNNs process inputs in both forward and backward directions, capturing both **past and future context** for each time step. This architecture is ideal for tasks where the entire sequence is available, such as named entity recognition and question answering. [55]
3. **Long Short-Term Memory Networks (LSTMs):** Long Short-Term Memory Networks (LSTMs) introduce a memory mechanism to overcome the vanishing gradient problem. Each LSTM cell has three gates: [55]
 - (a) **Input Gate:** Controls how much new information should be added to the cell state. [55]
 - (b) **Forget Gate:** Decides what past information should be discarded. [55]
 - (c) **Output Gate:** Regulates what information should be output at the current step. This selective memory enables LSTMs to handle long-term dependencies, making them ideal for tasks where earlier context is critical. [55]
4. **Gated Recurrent Units (GRUs):** Gated Recurrent Units (GRUs) **simplify LSTMs** by combining the input and forget gates into a single update gate and streamlining the output mechanism. This design is computationally efficient, often performing similarly to LSTMs and is useful in tasks where simplicity and faster training are beneficial. [55]

33.5. Advantages

1. Simple and **computationally cheap.** [7]
2. Works well for **short sequences or simple dependencies.** [7]
3. Forms the **foundation** for all later architectures (LSTM, GRU). [7]

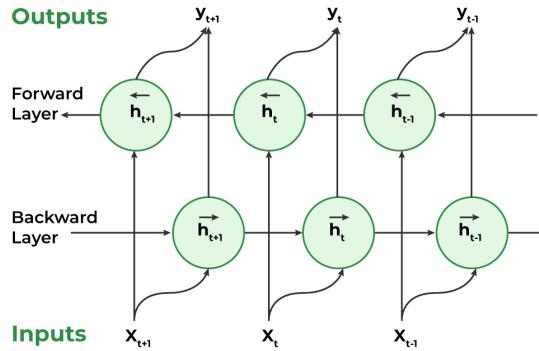
33.6. Disadvantages

1. Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t , as a function of the previous hidden state h_{t-1} and the input for position t . This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. [3]
2. RNNs traditionally require a pre-defined alignment between the input and output sequences to perform transduction. This is a severe limitation since finding the alignment is the most difficult aspect of many sequence transduction problems. [6]
3. traditional RNNs face challenges such as the **vanishing gradient** problem where gradients become too small during backpropagation making training difficult. [13]
4. **Vanishing/ exploding gradients:** during backpropagation through time (BPTT), gradients become too small or large → training unstable. [7]
5. **Poor long-term memory:** can't remember information from many steps ago. [7]
6. **One-directional:** can only use **past context**, not future context. [7]

33.7. What fixes it

1. LSTM and GRU add **gating mechanisms** to handle long-term dependencies. [7]
2. BiRNN adds reverse processing to capture **future context**. [7]

BI-DIRECTIONAL RNN (BRNN/ BIRNN) (1997)



1. A Bidirectional Recurrent Neural Network (BRNN) is an extension of the traditional RNN that processes sequential data in both forward and backward directions. This allows the network to utilize both past and future context when making predictions providing a more comprehensive understanding of the sequence. [13]

34.1. Working of Bidirectional Recurrent Neural Networks

1. **Inputting a Sequence:** A sequence of data points each represented as a vector with the same dimensionality is fed into the BRNN. The sequence may have varying lengths. [13]
2. **Dual Processing:** BRNNs process data in two directions: [13]
 - (a) **Forward direction:** The hidden state at each time step is determined by the current input and the previous hidden state. [13]
 - (b) **Backward direction:** The hidden state at each time step is influenced by the current input and the next hidden state. [13]
3. **Computing the Hidden State:** A non-linear activation function is applied to the weighted sum of the input and the previous hidden state creating a memory mechanism that allows the network to retain information from earlier steps. [13]
4. **Determining the Output:** A non-linear activation function is applied to the weighted sum of the hidden state and output weights to compute the output at each step. This output can either be: [13]
 - (a) The final output of the network. [13]
 - (b) An input to another layer for further processing. [13]

```

1 class BidirectionalVanillaRNN(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         super().__init__()
4         self.hidden_size = hidden_size
5         final concatenated hidden will be hidden_size * 2
6
7         self.super().__init__()
8         self.input_size = input_size

```

```

9      self.hidden_size = hidden_size
10     # FORWARD direction params
11     self.Wx_f = nn.Parameter(torch.randn(input_size, hidden_size) * 0.1)
12     self.Wh_f = nn.Parameter(torch.randn(hidden_size, hidden_size) * 0.1)
13     self.bh_f = nn.Parameter(torch.zeros(hidden_size))
14     # BACKWARD direction params
15     self.Wx_b = nn.Parameter(torch.randn(input_size, hidden_size) * 0.1)
16     self.Wh_b = nn.Parameter(torch.randn(hidden_size, hidden_size) * 0.1)
17     self.bh_b = nn.Parameter(torch.zeros(hidden_size))
18
19     # final classifier from concatenated hidden (h_f_last || h_b_last)
20     self.Why = nn.Parameter(torch.randn(hidden_size * 2, output_size) * 0.1)
21     self.by = nn.Parameter(torch.zeros(output_size))
22
23 def forward(self, x, h0_f=None, h0_b=None):
24     """
25         x: (batch, seq_len, input_size)
26         returns logits (batch, output_size),
27         and tuple of last forward/backward hidden states
28     """
29     batch, seq_len, _ = x.shape
30     # init hidden states
31     if h0_f is None:
32         h_f = x.new_zeros(batch, self.hidden_size)
33     else:
34         h_f = h0_f
35     if h0_b is None:
36         h_b = x.new_zeros(batch, self.hidden_size)
37     else:
38         h_b = h0_b
39
40     # forward pass (t = 0 .. seq_len-1)
41     for t in range(seq_len):
42         xt = x[:, t, :]                      # (batch, input_size)
43         pre_f = xt @ self.Wx_f + h_f @ self.Wh_f + self.bh_f
44         h_f = torch.tanh(pre_f)              # (batch, hidden_size)
45
46     # backward pass (t = seq_len-1 .. 0)
47     for t in range(seq_len - 1, -1, -1):
48         xt = x[:, t, :]
49         pre_b = xt @ self.Wx_b + h_b @ self.Wh_b + self.bh_b
50         h_b = torch.tanh(pre_b)
51
52     # concatenate final forward and backward hidden states
53     h_cat = torch.cat([h_f, h_b], dim=1)    # (batch, hidden_size*2)
54     logits = h_cat @ self.Why + self.by      # (batch, output_size)
55     return logits, (h_f, h_b)

```

Python Snippet 34.1: BiRNN from scratch (PyTorch) [7]

34.2. Advantages

- 1. Enhanced Context Understanding:** Considers both past and future data for improved predictions.

[7, 13]

2. **Improved Accuracy:** Particularly effective for NLP and speech processing tasks. [13]
3. **Better Handling of Variable-Length Sequences:** More flexible than traditional RNNs making it suitable for varying sequence lengths. [13]
4. **Increased Robustness:** Forward and backward processing help filter out noise and irrelevant information, improving robustness. [13]
5. Very effective for NLP tasks like POS tagging, NER, and speech recognition. [7]

34.3. Disadvantages

1. **High Computational Cost:** Requires twice the processing time compared to unidirectional RNNs. [13]
2. **Longer Training Time:** More parameters to optimize result in slower convergence. [13]
3. **Limited Real-Time Applicability:** Since predictions depend on the entire sequence hence they are not ideal for real-time applications like live speech recognition. [13]
4. **Less Interpretability:** The bidirectional nature of BRNNs makes it more difficult to interpret predictions compared to standard RNNs. [13]
5. Still suffers from **vanishing gradients** if it's a Vanilla BiRNN. [7]

34.4. What fixes it

1. Use BiLSTM or BiGRU → combine bidirectional context with gating mechanisms. [7]

LONG SHORT-TERM MEMORY (LSTM) (1997)

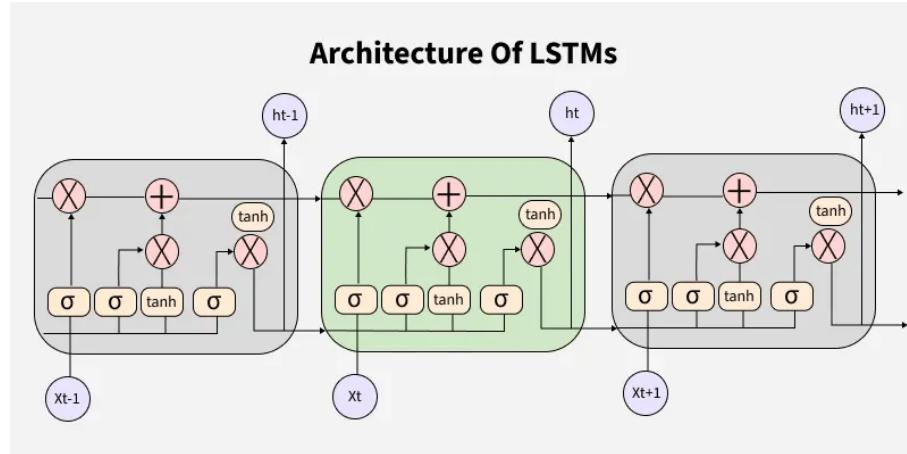


Fig. 35.1: LSTM: unrolling [38]

1. Long Short-Term Memory (LSTM) is an **enhanced version** of the Recurrent Neural Network (RNN) designed by Hochreiter and Schmidhuber. [38]
2. LSTMs can capture long-term dependencies in sequential data making them ideal for tasks like language translation, speech recognition and time series forecasting. [38]
3. Unlike traditional RNNs which use a single hidden state passed through time LSTMs introduce a **memory cell** that holds information over extended periods addressing the challenge of learning long-term dependencies. [38]
4. **Architecture:** LSTM architectures involves the memory cell which is controlled by three gates: [38]
 - (a) **Input gate:** Controls what information is added to the memory cell. [38]
 - (b) **Forget gate:** Determines what information is removed from the memory cell. [38]
 - (c) **Output gate:** Controls what information is output from the memory cell. [38]

This allows LSTM networks to selectively retain or discard information as it flows through the network which allows them to learn long-term dependencies. The network has a hidden state which is like its **short-term memory**. This memory is updated using the current input, the previous hidden state and the current state of the memory cell. [38]

35.1. Working of LSTM

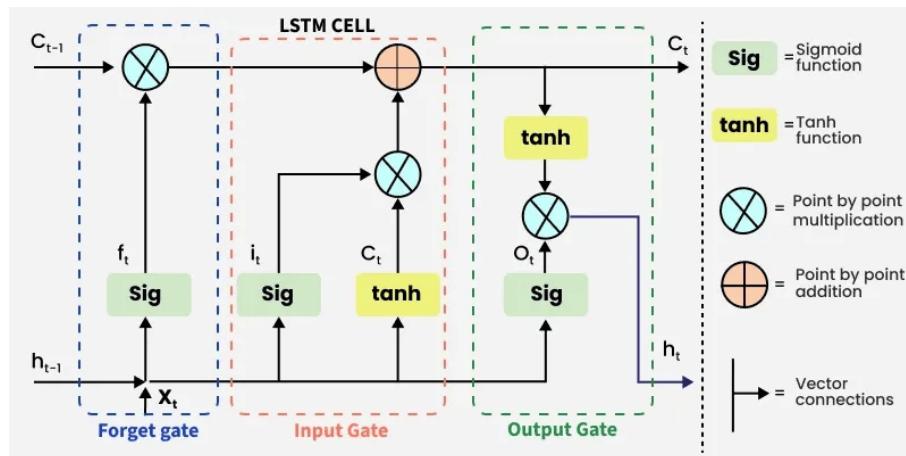


Fig. 35.2: LSTM Model [38]

```

1 class ManualLSTM(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         """
4             Single-layer LSTM implemented without using nn.LSTM / nn.LSTMCell.
5             Uses concatenated matrices for input-to-gates and hidden-to-gates:
6                 gates = x @ Wx + h @ Wh + b
7             where gates are concatenation [i | f | g | o] (input, forget, cell, output)
8         """
9         super().__init__()
10        self.input_size = input_size
11        self.hidden_size = hidden_size
12
13        # input -> gates (4 * hidden)
14        self.Wx = nn.Parameter(torch.randn(input_size, 4 * hidden_size) * 0.1)
15        # hidden -> gates (4 * hidden)
16        self.Wh = nn.Parameter(torch.randn(hidden_size, 4 * hidden_size) * 0.1)
17        # bias (4 * hidden)
18        self.b = nn.Parameter(torch.zeros(4 * hidden_size))
19
20        # final classifier from last hidden state
21        self.Why = nn.Parameter(torch.randn(hidden_size, output_size) * 0.1)
22        self.by = nn.Parameter(torch.zeros(output_size))
23
24    def forward(self, x, h0=None, c0=None):
25        """
26            x: (batch, seq_len, input_size)
27            returns logits (batch, output_size) and (h_last, c_last)
28        """
29        batch, seq_len, _ = x.shape
30        H = self.hidden_size
31
32        if h0 is None:
33            h = x.new_zeros(batch, H)
34        else:
35            h = h0
36        if c0 is None:
37            c = x.new_zeros(batch, H)
38        else:

```

```

39     c = c0
40
41     for t in range(seq_len):
42         xt = x[:, t, :] # (batch, input_size)
43         gates = xt @ self.Wx + h @ self.Wh + self.b # (batch, 4*H)
44         # split gates into i, f, g, o
45         i, f, g, o = gates.chunk(4, dim=1)
46         i = torch.sigmoid(i)
47         f = torch.sigmoid(f)
48         g = torch.tanh(g)           # candidate cell
49         o = torch.sigmoid(o)
50
51         c = f * c + i * g
52         h = o * torch.tanh(c)
53
54     logits = h @ self.Why + self.by # (batch, output_size)
55     return logits, (h, c)

```

Python Snippet 35.1: LSTM from scratch (PyTorch) [7]

35.1.1. Forget Gate

1. The information that is no longer useful in the cell state is removed with the forget gate. [38]
2. Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through **sigmoid activation function** which gives output in range of $[0, 1]$. [38]
3. If for a particular cell state the output is 0 or near to 0, the piece of information is **forgotten** and for output of 1 or near to 1, the information is **retained** for future use. [38]
4.
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
 [38]
 - (a) W_f represents the weight matrix associated with the forget gate. [38]
 - (b) $[h_{t-1}, x_t]$ denotes the concatenation of the current input and the previous hidden state. [38]
 - (c) b_f bias with the forget gate [38]
 - (d) σ sigmoid activation function [38]

35.1.2. Input gate

1. The addition of useful information to the cell state is done by the input gate. [38]
2. First the information is regulated using the **sigmoid function** and filter the values to be remembered similar to the forget gate using inputs h_{t-1} and x_t . [38]

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
 [38]
3. Then, a vector is created using **tanh function** that gives an output from -1 to $+1$ which contains all the possible values from h_{t-1} and x_t . [38]

$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
 [38]
4. At last the values of the vector and the regulated values are multiplied to obtain the useful information. [38]
5. We multiply the previous state by f_t effectively filtering out the information we had decided to ignore earlier. Then we add $i_t \odot \hat{C}_t$ which represents the new **candidate** values scaled by how much we

decided to update each state value. [38]

.

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t \quad [38]$$

6. \odot denotes element-wise multiplication [38]

7. \tanh is activation function [38]

35.1.3. Output gate

1. The output gate is responsible for deciding what part of the current cell state should be sent as the hidden state (output) for this time step. [38]

2. First, the gate uses a **sigmoid function** to determine which information from the current cell state will be output. This is done using the previous hidden state h_{t-1} and the current input x_t : [38]

.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad [38]$$

3. Next, the current cell state C_t is passed through a \tanh activation to scale its values between -1 and $+1$. Finally, this transformed cell state is multiplied element-wise with o_t to produce the hidden state h_t : [38]

.

$$h_t = o_t \odot \tanh(C_t) \quad [38]$$

This hidden state h_t is then passed to the next time step and can also be used for generating the output of the network. [38]

(a) o_t is the output gate activation. [38]

(b) C_t is the current cell state. [38]

(c) \odot represents element-wise multiplication. [38]

(d) σ is the sigmoid activation function. [38]

35.2. Advantages

1. Solves **vanishing gradient** problem using constant error flow through the cell state. [7]

2. Can capture **long-range dependencies**. [7]

3. Very effective for **time series** and **language modeling**. [7]

35.3. Disadvantages

1. **Complex** (more parameters \rightarrow slower training). [7]

2. **Overkill** for simple tasks. [7]

3. **Harder to tune** and requires more memory. [7]

35.4. What fixes it

1. GRU simplifies LSTM while maintaining similar performance.

CHAPTER 36

GATED RECURRENT UNITS (GRUs) (2014)

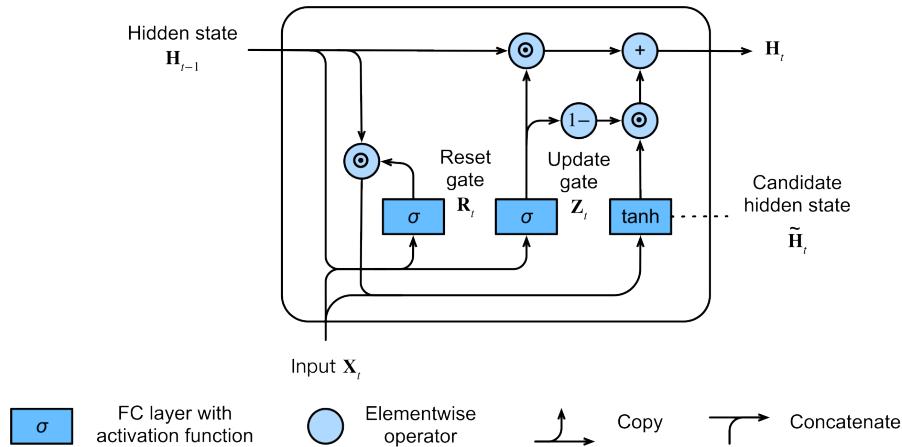


Fig. 36.1: GRU Architecture

1. Gated Recurrent Unit (GRU) was introduced which uses LSTM architecture by merging its gating mechanisms offering a more efficient solution for many sequential tasks without sacrificing performance. [37]
2. The core idea behind GRUs is to use gating mechanisms to selectively update the hidden state at each time step allowing them to remember important information while discarding irrelevant details. [37]
3. The GRU consists of two main gates:
 - (a) **Update Gate** (z_t): This gate decides how much information from previous hidden state should be retained for the next time step. [37]
 - (b) **Reset Gate** (r_t): This gate determines how much of the past hidden state should be forgotten. [37]

36.1. Working of GRU

1. **Reset gate:** $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$ [37]
The reset gate determines how much of the previous hidden state h_{t-1} should be forgotten. [37]
2. **Update gate:** $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$ [37]
The update gate controls how much of the new information x_t should be used to update the hidden state. [37]
3. **Candidate hidden state:** $h'_t = \tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t])$ [37]
This is the potential new hidden state calculated based on the current input and the previous hidden state. [37]
4. **Hidden state:** $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h'_t$ [37]
The final hidden state is a weighted average of the previous hidden state h_{t-1} and the candidate hidden state h'_t based on the update gate z_t . [37]

```

1 class ManualGRU(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         """
4             Single-layer GRU implemented without using nn.GRU / nn.GRUCell.
5             Uses separate parameter chunks for gates:
6                 z_t = sigmoid(x W_z + h U_z + b_z)
7                 r_t = sigmoid(x W_r + h U_r + b_r)
8                 n_t = tanh( x W_n + (r * h) U_n + b_n )
9                 h_t = (1 - z_t) * n_t + z_t * h
10            """
11
12     super().__init__()
13     self.input_size = input_size
14     self.hidden_size = hidden_size
15     H = hidden_size
16
17     # Input -> gates (z, r, n) : shapes (input_size, H)
18     self.Wx_z = nn.Parameter(torch.randn(input_size, H) * 0.1)
19     self.Wx_r = nn.Parameter(torch.randn(input_size, H) * 0.1)
20     self.Wx_n = nn.Parameter(torch.randn(input_size, H) * 0.1)
21
22     # Hidden -> gates (z, r, n) : shapes (H, H)
23     self.Wh_z = nn.Parameter(torch.randn(H, H) * 0.1)
24     self.Wh_r = nn.Parameter(torch.randn(H, H) * 0.1)
25     self.Wh_n = nn.Parameter(torch.randn(H, H) * 0.1)
26
27     # biases for each gate
28     self.b_z = nn.Parameter(torch.zeros(H))
29     self.b_r = nn.Parameter(torch.zeros(H))
30     self.b_n = nn.Parameter(torch.zeros(H))
31
32     # final classifier from last hidden state
33     self.Why = nn.Parameter(torch.randn(H, output_size) * 0.1)
34     self.by = nn.Parameter(torch.zeros(output_size))
35
36     def forward(self, x, h0=None):
37         """
38             x: (batch, seq_len, input_size)
39             returns logits (batch, output_size) and last hidden (batch, hidden_size)
40         """
41
42         batch, seq_len, _ = x.shape
43         H = self.hidden_size
44
45         if h0 is None:
46             h = x.new_zeros(batch, H)
47         else:
48             h = h0
49
50         for t in range(seq_len):
51             xt = x[:, t, :] # (batch, input_size)
52
53             # update gate z
54             pre_z = xt @ self.Wx_z + h @ self.Wh_z + self.b_z
55             z = torch.sigmoid(pre_z)

```

```

54
55     # reset gate r
56     pre_r = xt @ self.Wx_r + h @ self.Wh_r + self.b_r
57     r = torch.sigmoid(pre_r)
58
59     # candidate n (uses r * h for hidden contribution)
60     pre_n = xt @ self.Wx_n + (r * h) @ self.Wh_n + self.b_n
61     n = torch.tanh(pre_n)
62
63     # new hidden
64     h = (1 - z) * n + z * h
65     # equivalently: h = z * h + (1 - z) * n
66
67     logits = h @ self.Why + self.by  # (batch, output_size)
68     return logits, h

```

Python Snippet 36.1: GRU from scratch (PyTorch) [7]

36.2. Advantages

1. **Handles Vanishing Gradient:** GRUs help mitigate this issue by using gates that regulate the flow of gradients during training ensuring that important information is preserved and that gradients do not shrink excessively over time. By using these gates, GRUs maintain a balance between remembering important past information and learning new, relevant data. [37]
2. **Fewer parameters** → faster training than LSTM. [7]
3. **Performs comparably** to LSTM on many tasks. [7]
4. Easier to implement and **less prone to overfitting.** [7]

36.3. Disadvantages

1. May perform **slightly worse** on very long sequences. [7]
2. No explicit cell state → **sometimes less interpretable.** [7]

36.4. What fixes it

1. For tasks needing very long dependencies or finer control, use LSTM. [7]

IX

TIMELINED ML-DL TECHNOLOGIES

CHAPTER 37

CONNECTIONIST TEMPORAL CLASSIFICATION (CTC) (2006)

ADVANCED ML TECH: PAPER: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks

AUTHOR(S): Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber

YEAR: 2006

URL: https://www.cs.toronto.edu/~graves/icml_2006.pdf

1. Connectionist Temporal Classification (CTC) is an RNN output layer that defines a distribution over all alignments with all output sequences not longer than the input sequence. [6]
2. However, as well as precluding tasks, such as text-to-speech, where the output sequence is longer than the input sequence, CTC does not model the interdependencies between the outputs. [6]

CHAPTER 38

TRANSDUCER (2012)

ADVANCED ML TECH: PAPER: Sequence Transduction with Recurrent Neural Networks
AUTHOR(S): Alex Graves
YEAR: 2012
URL: <https://arxiv.org/abs/1211.3711>

1. Many machine learning tasks can be expressed as the transformation—or transduction—of input sequences into output sequences. Example: speech recognition, machine translation, protein secondary structure prediction and text-to-speech, etc [6]
2. One of the key challenges in sequence transduction is learning to represent both the input and output sequences in a way that is **invariant to sequential distortions** such as shrinking, stretching and translating. [6]
3. For a general-purpose sequence transducer, where the output length is **unknown** in advance, we would prefer a distribution over sequences of **all** lengths. Furthermore, since we do not know how the inputs and outputs should be aligned, this distribution would ideally cover all possible alignments. [6]
4. The transducer extends CTC by defining a distribution over output sequences of all lengths, and by jointly modelling both input-output and output-output dependencies. [6]
5. As a discriminative sequential model the transducer has similarities with ‘chain-graph’ conditional random fields (CRFs). However the transducer’s construction from RNNs, with their ability to extract features from raw data and their potentially unbounded range of dependency, is in marked contrast with the pairwise output potentials and hand-crafted input features typically used for CRFs. [6]
6. Transducer is closer in spirit is the **Graph Transformer Network** paradigm, in which differentiable modules (often neural networks) can be globally trained to perform consecutive graph transformations such as detection, segmentation and recognition. [6]

38.1. RNN Transducer

1. Let $\mathbf{x} = (x_1, x_2, \dots, x_T)$ be a length T input sequence of arbitrary length belonging to the set X^* of all sequences over some input space X . Let $\mathbf{y} = (y_1, y_2, \dots, y_U)$ be a length U output sequence belonging to the set Y^* of all sequences over some output space Y . We assume that the output space is discrete; however the method can be readily extended to continuous output spaces, provided a tractable, differentiable model can be found for Y . [6]
2. Both the inputs vectors x_t and the output vectors y_u are represented by fixed-length real-valued vectors; [6]
3. Let the extended output space $\bar{Y} = Y \cup \emptyset$, where \emptyset denotes the **null output**. The intuitive meaning of \emptyset is ‘output nothing’; the sequence $(y_1, \emptyset, \emptyset, y_2, \emptyset, y_3) \in \bar{Y}^*$ is therefore equivalent to $(y_1, y_2, y_3) \in Y^*$. [6]
4. We refer to the elements $a \in \bar{Y}^*$ as alignments, since the location of the null symbols determines an alignment between the input and output sequences. [6]
5. Given x , the RNN transducer defines a conditional distribution $P(a \in \bar{Y}^* | x)$. This distribution is then collapsed onto the following distribution over Y^* : [6]

$$P(y \in Y^* | x) = \sum_{a \in B^{-1}(y)} P(a | x) \quad [6]$$

- where $B : \bar{Y}^* \mapsto Y^*$ is a function that removes the null symbols from the alignments in \bar{Y}^* . [6]
6. Two recurrent neural networks are used to determine $P(a \in \bar{Y}^* | x)$. [6]
 - (a) **transcription network F** : scans the input sequence x and outputs the sequence $f = (f_1, \dots, f_T)$ of transcription vectors. [6]
 - (b) The other network, referred to as the **prediction network G** , scans the output sequence y and outputs the prediction vector sequence $g = (g_0, g_1, \dots, g_U)$. [6]
 7. For a task with K output labels, the output layer of the transcription network is size $K + 1$, just like the prediction network, and hence the transcription vectors f_t are also size $K + 1$. [6]

38.1.1. Prediction Network (G)

1. The prediction network G is a recurrent neural network consisting of an input layer, an output layer and a single hidden layer. The length $U + 1$ input sequence $\hat{y} = (\emptyset, y_1, \dots, y_U)$ to G output sequence y with \emptyset prepended. [6]
2. The inputs are encoded as one-hot vectors; that is, if Y consists of K labels and $y_u = k$, then \hat{y}_u is a length K vector whose elements are all zero except the k -th, which is one. \emptyset is encoded as a length K vector of zeros. The input layer is therefore size K . [6]
3. The output layer is size $K + 1$ (one unit for each element of \bar{Y}) and hence the prediction vectors g_u are also size $K + 1$. [6]
4. Given \hat{y} , G computes the hidden vector sequence (h_0, \dots, h_U) and the prediction sequence (g_0, \dots, g_U) by iterating the following equations from $u = 0$ to U :

$$(a) h_u = H(W_{ih}\hat{y}_u + W_{hh}h_{u-1} + b_h) \quad [6] \quad (b) g_u = W_{ho}h_u + b_o \quad [6]$$

where W_{ih} is the input-hidden weight matrix, W_{hh} is the hidden-hidden weight matrix, W_{ho} is the hidden-output weight matrix, b_h and b_o are bias terms, and H is the hidden layer function. [6]

5. In traditional RNNs H is an elementwise application of the tanh or logistic sigmoid $\sigma(x) = \frac{1}{1 + \exp(-x)}$ functions. [6]
6. The prediction network attempts to model each element of y given the previous ones; it is therefore similar to a standard next-step-prediction RNN, only with the added option of making ‘null’ predictions. [6]

38.1.2. Transcription Network (F)

1. The transcription network F is a bidirectional RNN that scans the input sequence x forwards and backwards with two separate hidden layers, both of which feed forward to a single output layer. [6]
2. Bidirectional RNNs are preferred because each output vector depends on the whole input sequence (rather than on the previous inputs only, as is the case with normal RNNs). [6]
3. Given a length T input sequence $(x_1 \dots x_T)$, a bidirectional RNN computes the forward hidden sequence $(\overrightarrow{h}_1, \dots, \overrightarrow{h}_T)$, the backward hidden sequence $(\overleftarrow{h}_1, \dots, \overleftarrow{h}_T)$, and the transcription sequence (f_1, \dots, f_T) by first iterating the backward layer from $t = T$ to 1:

$$\overleftarrow{h}_t = H\left(W_{i\overleftarrow{h}}i_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \quad [6]$$

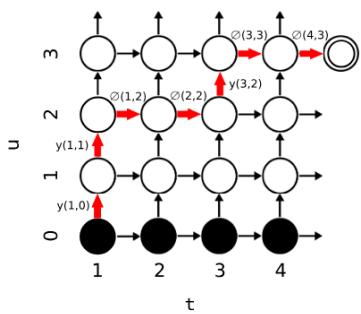
4. iterating the forward and output layers from $t = 1$ to T :

$$\overrightarrow{h}_t = H\left(W_{i\overrightarrow{h}}i_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t+1} + b_{\overrightarrow{h}}\right) \quad [6]$$

5. $o_t = W_{\overrightarrow{h}o}\overrightarrow{h}_t + W_{\overleftarrow{h}o}\overleftarrow{h}_t + b_o$ [6]

6. The transcription network is similar to a Connectionist Temporal Classification RNN, which also uses a null output to define a distribution over input-output alignments. [6]

38.1.3. Output Distribution



1. Output probability lattice defined by $P(k|t, u)$. [6]
2. The **node** at (t, u) represents the probability of having output the first u elements of the output sequence by point t in the transcription sequence. [6]
3. The **horizontal arrow** leaving node (t, u) represents the probability $\emptyset(t, u)$ of outputting nothing at (t, u) ; the **vertical arrow** represents the probability $y(t, u)$ of outputting the element $u + 1$ of \mathbf{y} . [6]
4. The **black nodes** at the bottom represent the null state before any outputs have been emitted. [6]
5. The **paths** starting at the bottom left and reaching the terminal node in the top right (one of which is shown in red) correspond to the possible alignments between the input and output sequences. [6]
6. Each alignment starts with probability 1, and its final probability is the product of the transition probabilities of the arrows they pass through (shown for the red path). [6]

1. Given the transcription vector f_t , where $1 \leq t \leq T$, the prediction vector g_u , where $0 \leq u \leq U$, and label $k \in \bar{Y}$, define the **output density function**:
$$h(k, t, u) = \exp(f_t^k + g_u^k)$$
 where superscript k denotes the k -th element of the vectors. [6]

2. The density can be normalised to yield the conditional output distribution:
$$P(k \in \bar{Y} | t, u) = \frac{h(k, t, u)}{\sum_{k' \in \bar{Y}} h(k', t, u)}$$
 [6]
3. simply: $y(t, u) \equiv P(y_{u+1} | t, u)$ $\emptyset(t, u) \equiv P(\emptyset | t, u)$ [6]
4. $P(k | t, u)$ is used to determine the transition probabilities in the lattice. [6]
5. The set of possible paths from the start node to the terminal node corresponds to the complete set of alignments between \mathbf{x} and \mathbf{y} , i.e. to the set $\bar{Y}^* \cap B^{-1}(\mathbf{y})$. Therefore all possible input-output alignments are assigned a probability, the sum of which is the total probability $P(\mathbf{y} | \mathbf{x})$ of the output sequence given the input sequence. [6]
6. Since a similar lattice could be drawn for any finite $\mathbf{y} \in Y^*$, $P(k | t, u)$ defines a distribution over all possible output sequences, given a single input sequence. [6]

38.2. LSTM Transducer

38.2.1. Prediction Network (G)

1. we have found that the Long Short-Term Memory (LSTM) architecture is better at finding and exploiting long range contextual information. [6]
2. hidden layer function H is implemented by the following composite function:

$$\begin{array}{lll} \text{(a)} & \alpha_n = \sigma(W_{i\alpha}i_n + W_{h\alpha}h_{n-1} + W_{s\alpha}s_{n-1}) & [6] \\ \text{(b)} & \beta_n = \sigma(W_{i\beta}i_n + W_{h\beta}h_{n-1} + W_{s\beta}s_{n-1}) & [6] \\ \text{(c)} & s_n = \beta_n s_{n-1} + \alpha_n \tanh(W_{is}i_n + W_{hs}) & [6] \end{array} \quad \begin{array}{ll} \text{(d)} & \gamma_n = \sigma(W_{i\gamma}i_n + W_{h\gamma}h_{n-1} + W_{s\gamma}s_n) & [6] \\ \text{(e)} & h_n = \gamma_n \tanh(s_n) & [6] \end{array}$$

where α, β, γ and s are respectively the **input gate**, **forget gate**, **output gate** and **state vectors**, all of which are the same size as the hidden vector h .

3. $W_{h\alpha}$ is the hidden-input gate matrix, $Wi\gamma$ is the input-output gate matrix etc. [6]
4. The weight matrices from the state to gate vectors are diagonal, so element m in each gate vector only receives input from element m of the state vector. The bias terms (which are added to α, β, s and γ) have been omitted for clarity. [6]
- 5.

38.3. Forward-Backward Algorithm

1. Define the forward variable $\alpha(t, u)$ as the probability of outputting $\mathbf{y}_{[1:u]}$ during $\mathbf{f}_{[1:t]}$. [6]
2. The forward variables for all $1 \leq t \leq T$ and $0 \leq u \leq U$ can be calculated recursively using:

$$\alpha(t, u) = \alpha(t-1, u) \otimes(t-1, u) + \alpha(t, u-1) y(t, u-1)$$
 [6]
 with initial condition $\alpha(1, 0) = 1$. [6]
3. The total output sequence probability is equal to the forward variable at the terminal node:

$$P(\mathbf{y}|\mathbf{x}) = \alpha(T, U) \otimes(T, U)$$
 [6]
4. Define the backward variable $\beta(t, u)$ as the probability of outputting $\mathbf{y}_{[u+1:U]}$ during $\mathbf{f}_{[t:T]}$. [6]
5. $\beta(t, u) = \beta(t+1, u) \otimes(t, u) + \beta(t, u+1) y(t, u)$
 with initial condition $\beta(T, U) = \otimes(T, U)$. [6]
6. From the definition of the forward and backward variables it follows that their product $\alpha(t, u)\beta(t, u)$ at any point (t, u) in the output lattice is equal to the probability of emitting the complete output sequence if y_u is emitted during transcription step t . [6]

38.4. Training

1. Given an input sequence \mathbf{x} and a target sequence \mathbf{y}^* , the natural way to train the model is to minimise the log-loss $L = -\ln(P(\mathbf{y}^*|\mathbf{x}))$ of the target sequence. [6]
2. $\forall n : 1 \leq n \leq U + T, P(\mathbf{y}^*|\mathbf{x}) = \sum_{(t,u):t+u=n} \alpha(t, u) \beta(t, u)$ [6]
3.
$$\frac{\partial L}{\partial P(k|t, u)} = \frac{\alpha(t, u)}{P(\mathbf{y}^*|\mathbf{x})} = \begin{cases} \beta(t, u+1) & \text{if } k = y_{u+1} \\ \beta(t+1, u) & \text{if } k = \emptyset \\ 0 & \text{otherwise} \end{cases}$$
 [6]
- (a)
$$\frac{\partial L}{\partial f_t^k} = \sum_{u=0}^U \sum_{k' \in \bar{Y}} \frac{\partial L}{\partial Pr(k'|t, u)} \frac{\partial Pr(k'|t, u)}{\partial f_t^k}$$
 [6] (b)
$$\frac{\partial L}{\partial g_u^k} = \sum_{t=1}^T \sum_{k' \in \bar{Y}} \frac{\partial L}{\partial Pr(k'|t, u)} \frac{\partial Pr(k'|t, u)}{\partial g_u^k}$$
 [6]
- where,
$$\frac{\partial Pr(k'|t, u)}{\partial f_t^k} = \frac{\partial Pr(k'|t, u)}{\partial g_u^k} = P(k'|t, u)[\delta_{kk'} - P(k|t, u)]$$
 [6]
4. A separate softmax could be calculated for every $P(k|t, u)$ required by the forward-backward algorithm. [6]

- (a) this is computationally expensive due to the high cost of the exponential function. [6]
- (b) we can instead pre-compute all the $\exp(f(t, x))$ and $\exp(g(y_{[1:u]}))$ terms and use their products to determine $P(k|t, u)$. [6]
- (c) This reduces the number of exponential evaluations from $\mathcal{O}(TU)$ to $\mathcal{O}(T + U)$ for each length T transcription sequence and length U target sequence used for training. [6]

38.5. Testing/ Inference

1. When the transducer is evaluated on test data, we seek the mode of the output sequence distribution induced by the input sequence. [6]
2. Unfortunately, finding the mode is much harder than determining the probability of a single sequence. The complication is that the prediction function $g(y_{[1:u]})$ (and hence the output distribution $P(k|t, u)$) may depend on all previous outputs emitted by the model. [6]
3. The method employed is a fixed-width beam search through the tree of output sequences. The advantage of beam search is that it scales to arbitrarily long sequences, and allows computational cost to be traded off against search accuracy. [6]
4. Let $P(\mathbf{y})$ be the approximate probability of emitting some output sequence \mathbf{y} found by the search so far. Let $P(k|\mathbf{y}, t)$ be the probability of extending \mathbf{y} by $k \in \bar{Y}$ during transcription step t . [6]
5. Let $\text{pref}(\mathbf{y})$ be the set of proper prefixes of \mathbf{y} (including the null sequence \emptyset), and for some $\hat{\mathbf{y}} \in \text{pref}(\mathbf{y})$, let $P(\mathbf{y}|\hat{\mathbf{y}}, t) = \prod_{u=|\hat{\mathbf{y}}|+1}^{|y|} P(y_u | \mathbf{y}_{[0:u-1]}, t)$. [6]

Algorithm 38.1: Output Sequence Beam Search [6]

```

1 Initialise:  $B = \{\emptyset\}; P(\emptyset) = 1$ 
2 for  $t \leftarrow 1$  to  $T$  do
3    $A = B$ 
4    $B = \{\}$ 
5   for  $\mathbf{y}$  in  $A$  do
6      $P(\mathbf{y}) += \sum_{\hat{\mathbf{y}} \in \text{pref}(\mathbf{y}) \cap A} P(\hat{\mathbf{y}}) P(\mathbf{y}|\hat{\mathbf{y}}, t)$ 
7   while  $B$  contains less than  $W$  elements more probable than the most probable in  $A$  do
8      $\mathbf{y}^* = \text{most probable in } A$ 
9     Remove  $\mathbf{y}^*$  from  $A$ 
10     $P(\mathbf{y}^*) = P(\mathbf{y}^*) P(\emptyset|\mathbf{y}, t)$ 
11    Add  $\mathbf{y}^*$  to  $B$ 
12    for  $k \in Y$  do
13       $P(\mathbf{y}^* + k) = P(\mathbf{y}^*) P(k|\mathbf{y}^*, t)$ 
14      Add  $\mathbf{y}^* + k$  to  $A$ 
15    Remove all but the  $W$  most probable from  $B$ 
16 return  $\mathbf{y}$  with highest  $\frac{\log P(\mathbf{y})}{|\mathbf{y}|}$  in  $B$ 

```

6. Pseudocode for a width W beam search for the output sequence with highest length-normalised probability given some length T transcription sequence. [6]
7. The algorithm can be trivially extended to an N best search ($N \leq W$) by returning a sorted list of the N best elements in B instead of the single best element. [6]
8. The length normalisation in the final line appears to be important for good performance, as otherwise shorter output sequences are excessively favoured over longer ones. [6]

-
- 9. The prediction network outputs are **independent** of previous hidden vectors given the current one, we can iteratively compute the prediction vectors for each output sequence $\mathbf{y} + k$ considered during the beam search by storing the hidden vectors for all \mathbf{y} , and running $h_u = H(W_{ih}\hat{\mathbf{y}}_u + W_{hh}h_{u-1} + b_h)$ for one step with k as input. [6]
 - 10. The prediction vectors can then be combined with the transcription vectors to compute the probabilities. This procedure greatly accelerates the beam search, at the cost of increased memory use. Note that for LSTM networks both the hidden vectors h and the state vectors s should be stored. [6]

ATTENTION MECHANISM (2014)

ADVANCED ML TECH: PAPER: Neural Machine Translation by Jointly Learning to Align and Translate

AUTHOR(S): Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

YEAR: 2014

URL: <https://arxiv.org/abs/1409.0473>

39.1. Intro: Encoder-decoder & Seq2seq

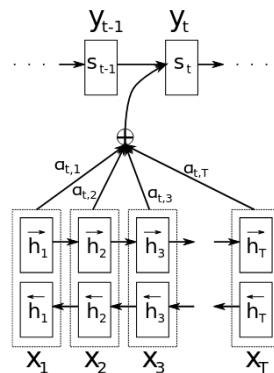
1. **Neural sequence modeling** is a framework in which a single neural network is trained end-to-end to map an input sequence to an output sequence. [5, 7]
2. **Traditional models** often rely on **manually designed** intermediate representations or **fixed-length embeddings** that attempt to compress all input information into one vector. This can create a bottleneck, since a single fixed-size representation may not capture all the necessary details of complex inputs. [5, 7]
3. To overcome this limitation, we can allow the model to **dynamically focus on specific parts** of the input that are most relevant to generating each part of the output. Instead of relying on a single static encoding, the model learns to compute **soft alignments**—that is, differentiable attention weights—over the input elements for each output step. This enables the system to retrieve context **adaptively** and improve performance across diverse tasks. [5, 7]
4. Empirical results show that models with such attention mechanisms often outperform fixed-representation architectures, and the learned alignments correspond intuitively to which input regions influence particular outputs. [5, 7]
5. Many modern neural models for sequence or structured data processing follow an encoder–decoder architecture. The encoder processes the input data (e.g., a sequence, image, or signal) and compresses it into a single fixed-length representation. The decoder then uses this representation to generate or predict an output sequence. [5, 7]
 - (a) This encoder–decoder system is trained jointly so that the generated outputs are as accurate as possible given the inputs. [5, 7]
 - (b) A potential **limitation** of this setup is that it requires the entire input to be compressed into a single fixed-size vector. This compression can lead to **information loss**, particularly for **long or complex inputs**. As a result, performance tends to degrade when the input size or complexity increases, because the fixed-length representation cannot retain all relevant details. [5, 7]
6. To address the limitations of fixed-length representations, we extend the encoder–decoder framework by allowing the model to **jointly learn to focus and predict**. At each output step, the model dynamically determines which parts of the input are most relevant for generating the current output element. Instead of compressing the entire input into a single vector, the model performs a soft search over all input positions to identify where important information is concentrated. [5, 7]
7. Using these relevance scores (attention weights), the model computes a **context vector** as a weighted combination of the input representations. This context vector summarizes the most pertinent information for the current prediction and is used—along with information from previous outputs—to generate the next element in the output sequence. [5, 7]

8. The key advantage of this mechanism is that it **eliminates the fixed-size bottleneck** of traditional encoder–decoder models. By representing the input as a sequence (or set) of feature vectors and selecting among them adaptively, the model can handle inputs of arbitrary length and complexity without being forced to compress all information into one vector. This flexibility allows it to retain richer details and cope more effectively with long or information-dense inputs. [5, 7]
9. Empirical studies show that models trained to learn attention jointly with prediction achieve **significant performance improvements** compared to basic encoder–decoder models. Moreover, the learned attention weights often provide meaningful insight into how the model associates different parts of the input with particular outputs—demonstrating that the learned “alignments” reflect human-interpretable relevance patterns. [5, 7]

39.2. Probabilistic Approach

1. From a probabilistic perspective, many learning tasks can be viewed as finding an output sequence y that maximizes the **conditional probability** $P(y|x)$, where x represents the input data. In practice, we train a parameterized model to approximate this conditional distribution using paired input–output data. Once the conditional model is learned, predictions can be generated by selecting the output sequence that maximizes $P(y|x)$. [5, 7]
2. Recent advances have introduced **neural network–based approaches** to directly model this conditional distribution. These approaches typically follow an **encoder–decoder design**: the encoder transforms the input x into a learned internal representation, and the decoder generates the output y conditioned on this representation. Recurrent neural networks (RNNs) or other sequential architectures (such as transformers or temporal CNNs) can be employed to handle variable-length inputs and outputs, encoding an input of arbitrary length into a fixed-length vector and then decoding it into a corresponding output sequence. [5, 7]
3. Although relatively new when first introduced, these neural encoder–decoder models demonstrated strong empirical performance on structured prediction tasks. For instance, recurrent architectures with memory mechanisms such as LSTMs were shown to achieve or surpass previous state-of-the-art results by capturing long-range dependencies between elements of the input and output. Further improvements were obtained by integrating neural sequence models with existing systems or by re-ranking candidate outputs using learned probabilistic scores. [5, 7]

39.2.1. RNN Encoder-Decoder



The graphical illustration of the proposed model trying to generate the t -th target token y_t given a source sentence (x_1, x_2, \dots, x_T) . [5, 7]

Encoder

1. the encoder processes the input — represented as a sequence of vectors: $\mathbf{x} = (x_1, \dots, x_{T_x})$ — and

transforms it into an internal representation c . [5, 7]

2. A common implementation uses a recurrent neural network (RNN), where each hidden state h_t is updated based on the current input and the previous hidden state: $h_t = f(x_t, h_{t-1})$ [5, 7]
3. The sequence of hidden states $\{h_1, \dots, h_{T_x}\}$ captures information about the entire input sequence. The final representation (or context vector) c is then computed as a function of these hidden states: $c = q(\{h_1, \dots, h_{T_x}\})$ where $h_t \in \mathbb{R}^n$ is a hidden state at time t , and c is a vector generated from the sequence of the hidden states. f and q are some nonlinear functions. [5, 7]
 - (a) LSTM can be used as f and $q(\{h_1, \dots, h_T\}) = h_T$ [Sutskever et al. (2014)] [5]

Decoder

1. The decoder is often trained to predict the next token/token $y_{t'}$ given the context vector c and all the previously predicted tokens $\{y_1, \dots, y_{t'-1}\}$. [5]
2. In other tokens, the decoder defines a probability over the translation \mathbf{y} by decomposing the joint probability into the ordered conditionals: [5]

$$P(\mathbf{y}) = \prod_{t=1}^T P(y_t | \{y_1, \dots, y_{t-1}\}, c) \quad [5]$$

where $\mathbf{y} = (y_1, \dots, y_{T_y})$. [5]

3. With an RNN, each conditional probability is modeled as: [5]

$$P(y_t | \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c) \quad [5]$$

where g is a nonlinear, potentially multi-layered, function that outputs the probability of y_t , and s_t is the hidden state of the RNN. [5]
4. Other architectures such as a hybrid of an RNN and a de-convolutional neural network can be used. [5]

39.3. Learning (RNN Encoder-Decoder)

Encoder: Bidirectional RNN For Annotating Sequences

1. The usual RNN reads an input sequence \mathbf{x} in order starting from the first symbol x_1 to the last one x_{T_x} . However, we would like the annotation of each token to summarize not only the preceding tokens, but also the following tokens. [5]
2. A BiRNN consists of forward and backward RNN's. The **forward RNN** \vec{f} reads the input sequence as it is ordered (from x_1 to x_{T_x}) and calculates a sequence of forward hidden states $(\vec{h}_1, \dots, \vec{h}_{T_x})$. The **backward RNN** \overleftarrow{f} reads the sequence in the reverse order (from x_{T_x} to x_1), resulting in a sequence of backward hidden states $(\overleftarrow{h}_1, \dots, \overleftarrow{h}_{T_x})$. [5]
3. We obtain an annotation for each word x_j by concatenating the forward hidden state \vec{h}_j and the backward one \overleftarrow{h}_j , i.e., $h_j = [\vec{h}_j; \overleftarrow{h}_j]^\top$. In this way, the annotation h_j contains the summaries of both the preceding words and the following words. Due to the tendency of RNNs to better represent recent inputs, the annotation h_j will be focused on the words around x_j . This sequence of annotations is used by the decoder and the alignment model later to compute the context vector c_i . [5]

Decoder

1. we define each conditional probability as: $P(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$ [5]

where s_i is an RNN hidden state for time i , computed by $s_i = f(s_{i-1}, y_{i-1}, c_i)$. [5]

2. unlike the existing encoder–decoder approach ($P(\mathbf{y}) = \prod_{t=1}^T P(y_t | \{y_1, \dots, y_{t-1}\}, c)$), here the probability is conditioned on a distinct context vector c_i for each target token y_i . [5]
3. The context vector c_i depends on a sequence of annotations (h_1, \dots, h_{T_x}) to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i -th token of the input sequence. [5]
4. The context vector c_i is, then, computed as a weighted sum of these annotations h_i : [5]

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \quad [5]$$

5. The weight α_{ij} of each annotation h_j is computed by: [5]

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \quad [5]$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i , $P(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$) and the j -th annotation h_j of the input sentence. [5]

6. We parametrize the alignment model a as a feedforward neural network which is jointly trained with all the other components. The alignment is not considered to be a latent variable Instead, the alignment model directly computes a **soft alignment**, which allows the gradient of the cost function to be backpropagated through. This gradient can be used to train the alignment model as well as the whole translation model jointly. [5]
7. We can understand the approach of taking a weighted sum of all the annotations as computing an **expected annotation**, where the expectation is over possible alignments. Let α_{ij} be a probability that the target token y_i is aligned to, or translated from, a source token x_j . Then, the i -th context vector c_i is the expected annotation over all the annotations with probabilities α_{ij} . [5]
8. The probability α_{ij} , or its associated energy e_{ij} , reflects the importance of the annotation h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i . Intuitively, this implements a mechanism of attention in the decoder. [5]
9. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly. [5]
10. An attention function can be described as **mapping a query** and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. [3]
11. The output is computed as a **weighted sum** of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. [3]

39.4. How Attention Mechanism Works?

1. **Input Encoding:** Input data is transformed into a format that the model can process and creating representations of the data. [2]
2. **Query Generation:** A query vector is generated based on the current state or context of the model. This query tells the model what it is looking for in the input data. [2]

3. **Key-Value Pair Creation:** Input is splitted into key-value pairs: [2]

(a) **Keys** represents the important information required to measure the relevant data. [2]

(b) **Values** hold actual data. [2]

4. **Similarity Computation:** Model calculates similarity between the query vector and each key. This helps find how relevant each part of the input is. Various methods can be used to calculate this similarity such as dot products or cosine similarity. [2]

$$\text{Score}(s, i) = \begin{cases} h_s^{(1)} \cdot y_i & \text{Dot Product} \\ (h_s^{(2)})^\top W y_i & \text{General} \\ v^\top \tanh \left(W \begin{bmatrix} h_s \\ y_i \end{bmatrix} \right) & \text{Concat} \end{cases} \quad [2]$$

(a) h_s : Encoder Source hidden state at position s [2]

(b) y_i : Encoder Target hidden state at the position i [2]

(c) W : Weight Matrix [2]

(d) v : Weight vector [2]

5. **Attention Weights Calculation:** Similarity scores are passed through a softmax function to find attention weights. These weights indicate the importance of each key-value pair. [2]

$$\cdot \quad \text{Attention Weight}(\alpha(s, i)) = \text{softmax}(\text{Similarity Scores}(s, i)) \quad [2]$$

6. **Weighted Sum:** Attention weights are applied to the corresponding values which helps in generating a weighted sum. This step adds the relevant information from the input based on their importance calculated by the attention mechanism. [2]

$$\cdot \quad c_t = \sum_{i=1}^{T_s} \alpha(s, i) h_j^{(1)} \quad [2]$$

where T_s : Total number of key-value pairs (source hidden states) in the encoder. [2]

7. **Context Vector:** Weighted sum act as a context vector which represents attended information from the input. It captures the relevant context for the current task. [2]

8. **Integration with the Model:** Context vector is combined with the model's current state or hidden representation which provides additional information for other steps of the model. [2]

9. **Repeat:** Steps 2 to 8 are repeated for each step of the model which allows the attention mechanism to focus on different parts of the input sequence or data. [2]

39.5. Self-Attention

1. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. [3]

2. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representation. [3]

3. the self-attention mechanism allows every position to attend directly to every other position in a single step. [7]

4. That means any two tokens can interact in one operation, independent of how far apart they are in the sequence. [7]

5. the number of operations needed to connect two arbitrary positions is constant ($\mathcal{O}(1)$), not dependent on distance. [7]

6. Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. [3]

39.6. Scaled Dot-Product Attention (Attention(Q, K, V))

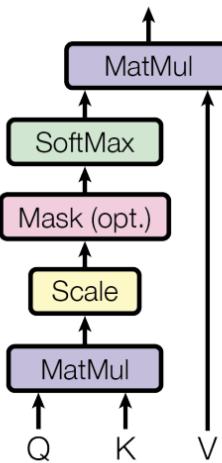


Fig. 39.1: Scaled Dot-Product Attention [3]

1. The input consists of queries and keys of dimension d_k , and values of dimension d_v . We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a **softmax function** to obtain the weights on the values. [3]
2. We compute the attention function on a set of queries simultaneously, packed together into a matrix Q . The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as: [3]

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V \quad [3]$$

3. The two most commonly used attention functions are additive attention, and dot-product (multiplicative) attention. [3]

(a) Dot-product attention is identical, except for the scaling factor $\frac{1}{\sqrt{d_k}}$. [3]

(b) Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. [3]

While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. [3]

4. While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k . We suspect that for large values of d_k , the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. (To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k .) To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$. [3]

39.7. Multi-Head Attention (MultiHead(Q, K, V))

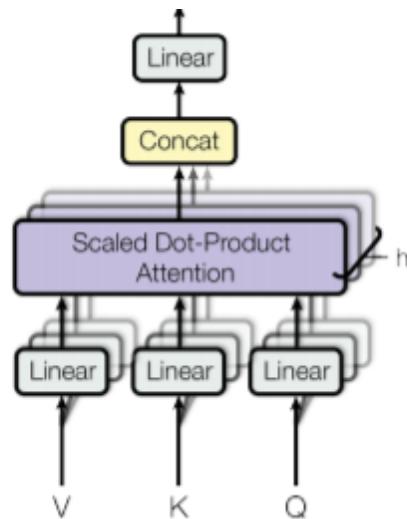


Fig. 39.2: Multi-Head Attention [3]

1. Instead of performing a single attention function with d_{model} -dimensional keys, values and queries, we found it beneficial to **linearly project** the queries, keys and values h times with different, learned linear projections to d_k , d_k and d_v dimensions, respectively. [3]
2. On each of these projected versions of queries, keys and values we then perform the attention function **in parallel**, yielding d_v -dimensional output values. [3]
3. These are concatenated and once again projected, resulting in the final values. [3]
4. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this. [3]
5. $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$ [3]

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. [3]

CHAPTER 40

TRANSFORMER (2017)

ADVANCED ML TECH: PAPER: Attention Is All You Need

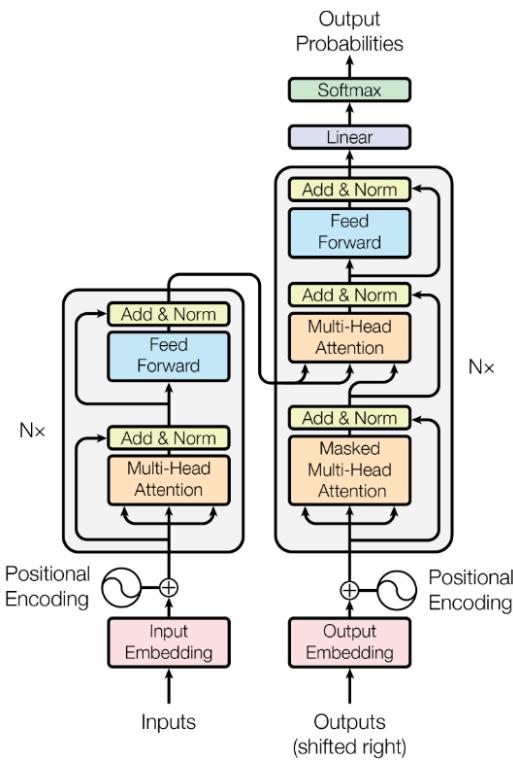
AUTHOR(S): Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

YEAR: 2017

URL: <https://arxiv.org/abs/1706.03762>

1. The (dominant) traditional sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. Some models also connect the encoder and decoder through an attention mechanism. [3]
2. The Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. [3]
3. **Attention mechanisms** have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. In all but a few cases, however, such attention mechanisms are used in conjunction with a recurrent network. [3]
4. Transformer is a model architecture eschewing (= avoiding) recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. [3]
5. the number of operations required to relate signals from two arbitrary input or output positions (tokens) is a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention. [3]
6. End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks. [3]

40.1. Model Architecture (encoder-decoder)



Transformer - model architecture
left - encoder, right - decoder [3]

1. the encoder maps an input sequence of symbol representations (x_1, \dots, x_n) to a sequence of continuous representations $z = (z_1, \dots, z_n)$. Given z , the decoder then generates an output sequence (y_1, \dots, y_m) of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. [3]
2. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decode. [3]

40.1.1. Encoder and Decoder Stacks

Encoder

1. The encoder is composed of a stack of N identical layers. [3]
2. Each layer has two sub-layers. The first is a **multi-head self-attention** mechanism, and the second is a simple, position-wise fully connected feed-forward network. [3]
3. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. [3]
4. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model} . [3]

Decoder

1. The decoder is composed of a stack of N identical layers. [3]
2. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs **multi-head attention** over the output of the encoder stack. [3]
3. we employ residual connections around each of the sub-layers, followed by layer normalization. [3]

4. We modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i . [3]

40.2. Applications of Attention in Transformer

The Transformer uses multi-head attention in three different ways:

1. In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models. [3]
2. The encoder contains **self-attention layers**. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder. [3]
3. Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. [3]

40.3. Position-wise Feed-Forward Networks

1. each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a **ReLU activation** in between. [3]
2. $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$ [3]
3. While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. [3]

40.4. Embeddings and Softmax

1. We use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model} . [3]
2. We also use the usual learned **linear transformation** and softmax function to convert the decoder output to predicted next-token probabilities. [3]
3. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation. [3]
4. In the embedding layers, we multiply those weights by $\sqrt{d_{\text{model}}}$. [3]

40.5. Positional Encoding

1. Since the model contains no recurrence and no convolution, in order for the model to make use of the **order of the sequence**, we must inject some information about the relative or absolute position of the tokens in the sequence. [3]

2. we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. [3]
3. There are many choices of positional encodings, learned and fixed. [3]
4. sine and cosine functions of different frequencies: [3]
 - (a) $PE(pos, 2i) = \sin(pos/10000^{2i/d_{\text{model}}})$ [3]
 - (b) $PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{\text{model}}})$ [3]

where pos is the position and i is the dimension. [3]
5. each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. [3]
6. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k , PE_{pos+k} can be represented as a linear function of PE_{pos} . We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training. [3]

X

APPLICATIONS

CHAPTER 41

AUTOMATIC SPEECH RECOGNITION

XI

APPENDIX

CHAPTER 42

DATASETS

DATASET NAME	SIZE (du -sh)	RECORDS/ ITEMS	SOURCE(S)
Demographics Synthetic	24K	(500,6)	[56]
Face Data	260K	(3628,7)	[56]
High School	2.4M	(50069,13)	[56]
Houses	32K	(546,13)	[56]
Potatoes	876K	(47582,7)	[56]
Voting Demo	36K	(750,7)	[56]
Sampling Plans - Population	8.9M	(50000,25)	[7]

REFERENCES

- [1] ADVANCED ML TECH: BLOG: What is an attention mechanism?
AUTHOR(S): Dave Bergmann, Cole Stryker
URL: <https://www.ibm.com/think/topics/attention-mechanism>
- [2] ADVANCED ML TECH: GFG: ML - Attention mechanism
URL: <https://www.geeksforgeeks.org/artificial-intelligence/ml-attention-mechanism/>
- [3] ADVANCED ML TECH: PAPER: Attention Is All You Need
AUTHOR(S): Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin
YEAR: 2017
URL: <https://arxiv.org/abs/1706.03762>
- [4] ADVANCED ML TECH: PAPER: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks
AUTHOR(S): Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber
YEAR: 2006
URL: https://www.cs.toronto.edu/~graves/icml_2006.pdf
- [5] ADVANCED ML TECH: PAPER: Neural Machine Translation by Jointly Learning to Align and Translate
AUTHOR(S): Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio
YEAR: 2014
URL: <https://arxiv.org/abs/1409.0473>
- [6] ADVANCED ML TECH: PAPER: Sequence Transduction with Recurrent Neural Networks
AUTHOR(S): Alex Graves
YEAR: 2012
URL: <https://arxiv.org/abs/1211.3711>
- [7] AI: OpenAI: ChatGPT
URL: <https://chatgpt.com/>
- [8] ARTIFICIAL INTELLIGENCE: BOOK: Artificial Intelligence: A Modern Approach 3e
SERIES: Prentice Hall Series
AUTHOR(S): Stuart Russell and Peter Norvig
PUBLISHER: Pearson
YEAR: 2010
URL: <https://aima.cs.berkeley.edu/>
- [9] ARTIFICIAL INTELLIGENCE: WIKIPEDIA: Logic
URL: <https://en.wikipedia.org/wiki/Logic>
- [10] ARTIFICIAL INTELLIGENCE: WIKIPEDIA: Syllogism
URL: <https://en.wikipedia.org/wiki/Syllogism>
- [11] BASIC MATHS: WIKIPEDIA: Absolute value
URL: https://en.wikipedia.org/wiki/Absolute_value
- [12] BASIC MATHS: WIKIPEDIA: Error function
URL: https://en.wikipedia.org/wiki/Error_function
- [13] BiRNN: GFG: Bidirectional Recurrent Neural Network
URL: <https://www.geeksforgeeks.org/deep-learning/bidirectional-recurrent-neural-network/>
- [14] DATA: PY-LIBRARY: matplotlib.pyplot: boxplot
URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html
- [15] DATA: PY-LIBRARY: matplotlib.pyplot: pie
URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html

- [16] DATA: PY-LIBRARY: seaborn: boxplot
URL: <https://seaborn.pydata.org/generated/seaborn.boxplot.html>
- [17] DATA: PY-LIBRARY: seaborn: countplot
URL: <https://seaborn.pydata.org/generated/seaborn.countplot.html>
- [18] DATA: PY-LIBRARY: seaborn: displot
URL: <https://seaborn.pydata.org/generated/seaborn.displot.html>
- [19] DATA: PY-LIBRARY: seaborn: histplot
URL: <https://seaborn.pydata.org/generated/seaborn.histplot.html>
- [20] DATA: PY-LIBRARY: seaborn: pairplot
URL: <https://seaborn.pydata.org/generated/seaborn.pairplot.html>
- [21] DATA: PY-LIBRARY: seaborn: scatterplot
URL: <https://seaborn.pydata.org/generated/seaborn.scatterplot.html>
- [22] DATA: WIKIPEDIA: Scatter plot
URL: https://en.wikipedia.org/wiki/Scatter_plot
- [23] DISTRIBUTION: WIKI: Bernoulli distribution
URL: https://en.wikipedia.org/wiki/Bernoulli_distribution
- [24] DISTRIBUTION: WIKI: Beta distribution
URL: https://en.wikipedia.org/wiki/Beta_distribution
- [25] DISTRIBUTION: WIKI: Binomial distribution
URL: https://en.wikipedia.org/wiki/Binomial_distribution
- [26] DISTRIBUTION: WIKI: Chi-squared distribution
URL: https://en.wikipedia.org/wiki/Chi-squared_distribution
- [27] DISTRIBUTION: WIKI: Continuous uniform distribution
URL: https://en.wikipedia.org/wiki/Continuous_uniform_distribution
- [28] DISTRIBUTION: WIKI: Exponential distribution
URL: https://en.wikipedia.org/wiki/Exponential_distribution
- [29] DISTRIBUTION: WIKI: Laplace distribution
URL: https://en.wikipedia.org/wiki/Laplace_distribution
- [30] DISTRIBUTION: WIKI: Log-normal distribution
URL: https://en.wikipedia.org/wiki/Log-normal_distribution
- [31] DISTRIBUTION: WIKI: Multinomial distribution
URL: https://en.wikipedia.org/wiki/Multinomial_distribution
- [32] DISTRIBUTION: WIKI: Multivariate normal distribution
URL: https://en.wikipedia.org/wiki/Multivariate_normal_distribution
- [33] DISTRIBUTION: WIKI: Negative binomial distribution
URL: https://en.wikipedia.org/wiki/Negative_binomial_distribution
- [34] DISTRIBUTION: WIKI: Normal distribution
URL: https://en.wikipedia.org/wiki/Normal_distribution
- [35] DISTRIBUTION: WIKI: Poisson distribution
URL: https://en.wikipedia.org/wiki/Poisson_distribution
- [36] DISTRIBUTION: WIKI: Student's t-distribution
URL: https://en.wikipedia.org/wiki/Student%27s_t-distribution
- [37] GRU: GFG: Gated Recurrent Unit Networks
URL: <https://www.geeksforgeeks.org/machine-learning/gated-recurrent-unit-networks/>
- [38] LSTM: GFG: What is LSTM - Long Short Term Memory?
URL: <https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/>

- [39] MACHINE LEARNING: BLOG: Common Loss functions in machine learning for a Regression model
URL: <https://medium.com/analytics-vidhya/common-loss-functions-in-machine-learning-for-a-regression-model-27d2bbda9c93>
- [40] MACHINE LEARNING: BLOG: Loss Functions in Machine Learning Explained
URL: <https://www.datacamp.com/tutorial/loss-function-in-machine-learning>
- [41] MACHINE LEARNING: BOOK: Pattern Recognition And Machine Learning
SERIES: Information Science and Statistics
AUTHOR(S): Christopher M. Bishop
PUBLISHER: SPRINGER NP EXCLUSIVE (CBS)
YEAR: 2009
URL: <https://www.amazon.in/Pattern-Recognition-Machine-Learning-2009/dp/1493938436>
- [42] MACHINE LEARNING: GFG: Bias-Variance Trade Off - Machine Learning
URL: <https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/>
- [43] MACHINE LEARNING: GFG: Cross Validation in Machine Learning
URL: <https://www.geeksforgeeks.org/machine-learning/cross-validation-machine-learning/>
- [44] MACHINE LEARNING: GFG: Model Selection for Machine Learning
URL: <https://www.geeksforgeeks.org/machine-learning/model-selection-for-machine-learning/>
- [45] MACHINE LEARNING: GFG: Regularization in Machine Learning
URL: <https://www.geeksforgeeks.org/machine-learning/regularization-in-machine-learning/>
- [46] MACHINE LEARNING: GFG: What is Generative AI?
URL: <https://www.geeksforgeeks.org/artificial-intelligence/what-is-generative-ai/>
- [47] MACHINE LEARNING: IBM: What is agentic AI?
URL: <https://www.ibm.com/think/topics/agentic-ai>
- [48] MACHINE LEARNING: WIKI: Regularization (mathematics)
URL: [https://en.wikipedia.org/wiki/Regularization_\(mathematics\)](https://en.wikipedia.org/wiki/Regularization_(mathematics))
- [49] MATHS FOR ML: BOOK: Mathematics For Machine Learning
AUTHOR(S): Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong
PUBLISHER: Cambridge University Press
YEAR: 2020
URL: <https://mml-book.com/>
- [50] MATHS FOR ML: GFG: Computational Graphs in Deep Learning
URL: <https://www.geeksforgeeks.org/deep-learning/computational-graphs-in-deep-learning/>
- [51] MATHS FOR ML: WIKIPEDIA: Gram–Schmidt process
URL: https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process
- [52] MATHS FOR ML: YOUTUBE: Essence of linear algebra (3Blue1Brown) (16 videos)
URL: https://www.youtube.com/playlist?list=PLZHQB0b0WTQDPD3MizzM2xFFitgF8hE_ab
- [53] MATHS FOR ML: YOUTUBE: Gram-Schmidt Orthogonalisation Process - Linear Algebra by GP Sir
URL: <https://www.youtube.com/watch?v=U0ZjINOGLog&t=125s>
- [54] MATHS FOR ML: YOUTUBE: Pavel Grinfeld's Part 1 Linear Algebra: An In-Depth Introduction (MathTheBeautiful) (149 videos)
URL: <https://www.youtube.com/playlist?list=PL1XfTHzgMRUKXD88IdzS14F4NxAZudSmv>
- [55] RNN: GFG: Introduction to Recurrent Neural Networks
URL:
<https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/>
- [56] STATISTICS: BOOK: Statistics for Data Scientists: An Introduction to Probability, Statistics, and Data Analysis
SERIES: Undergraduate Topics in Computer Science
AUTHOR(S): Maurits Kaptein and Edwin van den Heuvel
PUBLISHER: Springer
YEAR: 2022
URL: <https://doi.org/10.1007/978-3-030-10531-0>

- [57] STATISTICS: CLT: Hypothesis Testing: Cheat Sheet
URL: <https://ctl.unm.edu/assets/docs/resources/hypothesis-testing-sheet.pdf>
- [58] STATISTICS: GFG: Marginal Probability
URL: <https://www.geeksforgeeks.org/engineering-mathematics/marginal-probability/>
- [59] TOOLS: Codecogs: Equation Editor
URL: <https://editor.codecogs.com/>
- [60] TOOLS: Draw.io (app.diagrams.net)
URL: <https://draw.io/>
- [61] TOOLS: Matrix Calculus
URL: <https://www.matrixcalculus.org/>